

**Project Spot Check:**

**Developing a Vision-Based Program for Identifying Individual Snow Leopards from Camera Trap Photographs**

**ECE 20.4’s User Guide to Recognition.py**



****

Contents

[List of Figures 2](#_Toc41941732)

[List of Tables 3](#_Toc41941733)

[Introduction 4](#_Toc41941734)

[Installation 4](#_Toc41941735)

[Setting Up Recognition.py 5](#_Toc41941736)

[Initializing the ‘Images’ and ‘Templates’ Directories 5](#_Toc41941737)

[Installing and Setting up Mask R-CNN 6](#_Toc41941738)

[Understanding and Modifying config.json 8](#_Toc41941739)

[Filling Out easy\_run.py 9](#_Toc41941740)

[Constructing the Full Working Directory 11](#_Toc41941741)

[Running Recognition.py 12](#_Toc41941742)

[Opening the Command Line 12](#_Toc41941743)

[For Windows: 12](#_Toc41941744)

[For Linux/Mac OS: 12](#_Toc41941745)

[Using the Command Line Argument 12](#_Toc41941746)

[Not necessary: 13](#_Toc41941747)

[Necessary: 13](#_Toc41941748)

[Running the Program 14](#_Toc41941749)

[How Recognition.py Works 14](#_Toc41941750)

[Recognition.py Output 15](#_Toc41941751)

[Addendum 17](#_Toc41941752)

# List of Figures

[Figure 1: Ensuring Python is installed correctly 5](https://redhawks.sharepoint.com/sites/panthera2020/Shared%20Documents/Recognition%20Local%20User%20Guide.docx#_Toc41941714)

[Figure 2: Installing OpenCV 5](https://redhawks.sharepoint.com/sites/panthera2020/Shared%20Documents/Recognition%20Local%20User%20Guide.docx#_Toc41941715)

[Figure 3: Location of the three folders needing to be copied 6](#_Toc41941716)

[Figure 4: Location of model.py to be updated 7](#_Toc41941717)

[Figure 5: Modified lines in model.py for Tensorflow 2.2.0 7](#_Toc41941718)

[Figure 6: Screenshot of config.json in Notepad 8](https://redhawks.sharepoint.com/sites/panthera2020/Shared%20Documents/Recognition%20Local%20User%20Guide.docx#_Toc41941719)

[Figure 7: easy\_run.py template to be filled out 9](#_Toc41941720)

[Figure 8: Completed easy\_run.py file with macOS paths 10](#_Toc41941721)

[Figure 9. Example of a working directory 11](#_Toc41941722)

[Figure 10. edited\_photos folder for original copies 11](#_Toc41941723)

[Figure 11. How to open the command line (Windows) 12](#_Toc41941724)

[Figure 12. Example of when not to use the command line argument 13](#_Toc41941725)

[Figure 13. Extracting the work directory address 14](#_Toc41941726)

[Figure 14. Basic command line 14](#_Toc41941727)

[Figure 15. Using the command line 14](#_Toc41941728)

[Figure 16: Elbow graph of the k-Means output 16](#_Toc41941729)

[Figure 17: Markov clustering output 16](#_Toc41941730)

# List of Tables

[Table 1. Templating method from editing "templating" field of config.json 8](#_Toc41941731)

# Introduction

In early 2019, Panthera recommended that team ECE 19.7 move away from the HotSpotter program and that they create their own lightweight, simple version. So, ECE 19.7 and Ross Pitman developed Recognition.py. This is a Python script that is completely automated and has no graphical user interface but, still serves the same functionality as HotSpotter. It has integrated template support, autoquery functionality, and outputs a score matrix used for clustering algorithms. The results of ECE 19.7’s work on Recognition.py is documented fully on the team’s GitHub repository: <https://github.com/devindewitt/SU-ECE-19-7>

During fall 2019, team ECE 20.4 was introduced to Recognition.py for further development in methods in increasing accuracy and making the program functional for multiple operating systems. We, ECE 20.4, introduced new functionality for improvement of accuracy through image enhancement, Mask R-CNN templating, and clustering methods. These functions are described in the updated Function Description document for Recognition.py. Another version of the program was also created for use in Azure Databricks, so that larger datasets could be tested. This updated version is publicly available on our GitHub repository finished in June 2020: <https://github.com/caballe4/SU-ECE-20-4>

The following user guide gives a thorough explanation of how to set up, run, and evaluate the output of the Recognition.py Python script. We went through this installation and operation process on Windows and MacOS. You will see in the instructions below that we included the necessary steps for each operating system. This user guide also details how the program functions in the “main” section of the code, and what edits may have to be made to the code to achieve running the program. The outputs of the Recognition.py script will also be explained in this user guide.

# Installation

The first step is to install a Python 3 interpreter. We recommend *Anaconda*. Anaconda is a free distribution of the Python programming language that easily handles all installations, updates, package additions, and provides a multitude of IDE’s (Integrated Development Environments). You can download the installer for your operating system at this link: <https://www.anaconda.com/distribution/#download-section>. Download the 64-bit version. Once the installer downloads, continue with the default installation. OPTIONAL: On the ‘Advanced Options’ page, select the check box for ‘Add Anaconda to my PATH environment variable’. Select the ‘Register Anaconda as my default Python 3.7’ checkbox. This is necessary for Recognition.py to run. Then, finish out the installation, a computer restart may be required.

Next, you need to download Recognition.py and the other files that come with it. We have them stored on GitHub at this link: <https://github.com/caballe4/SU-ECE-20-4>. Go to the webpage and click on the green ‘Clone or download’ button and it will reveal a URL address. Copy that address for the time being. Now, we need to clone the repository to your computer. You may also download the repository ZIP file, if preferred. Inside this folder will also be a link to a Dropbox account. Copy and paste the Dropbox link into your browser and download the ZIP file. This file contains everything necessary to run the Mask R-CNN templating section of the code.

At this point, all necessary software and files should be on your computer. If you chose to add Anaconda to the PATH environment, then on Windows open *PowerShell* and on MacOS/Linux open *Terminal*. Then type ‘python’. You should see a screen similar to what is shown below in Figure 1. Then enter ‘exit()’ to leave the interpreter. If you receive an error, delete Anaconda and reinstall it making sure that you select the ‘Add Anaconda to my PATH environment variable’ box. Otherwise open the Anaconda terminal to use Python.

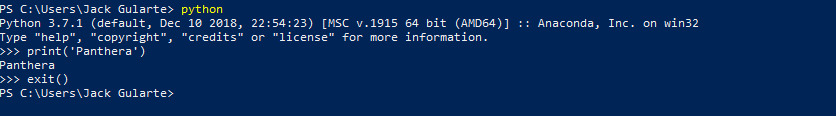


Figure : Ensuring Python is installed correctly

The final bit of set up involves installing OpenCV. This software package holds the computer vision tools that are used to identify the snow leopards in the images. You should see an output similar to Figure 2 after running this command: ***pip install opencv-contrib-python==3.4.2.16***

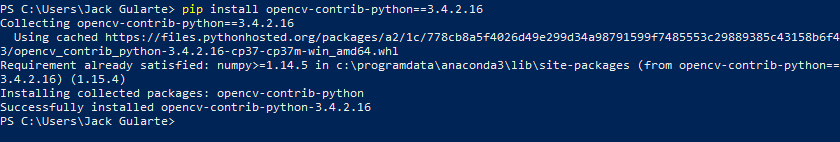


Figure : Installing OpenCV

# Setting Up Recognition.py

Light user setup is required for Recognition.py to run correctly. The next few sections will go over, in detail, how to set up each folder correctly so that no errors occur upon runtime. If you plan on using the image files that were included in the GitHub repository, you can skip to the ‘Running Recognition.py’ section. If you plan on using new images stored on your own computer, please read the following section to ensure the setup is done properly.

## Initializing the ‘Images’ and ‘Templates’ Directories

The program expects that the working directory (the folder the program is run in) has two correctly named folders. Both folder names are self-explanatory; one folder needs to be named ‘images’ and the other is to be named ‘templates.’ The images folder will contain only the images that you want to evaluate and nothing else. The templates folder will only contain the matching templates. Having any sort of errant file within either of these folders will cause errors within the program. When it is first run, Recognition.py will verify that each image has a matching template, so it is important that the pairs are named correctly. For example, if an image is named “\_\_imageOne.JPG” then its matching template name would be “\_\_imageOne.BMP.” Only the file extensions should differ. If the program finds an error between the file names it will print the name of the unmatched image, request that you fix the issue, and then rerun the program.

## Installing and Setting up Mask R-CNN

After the Dropbox link is downloaded and the ZIP is on the computer, there are a few different files that you need to copy into the same folder as the folder downloaded from the Github. After the ZIP file is downloaded, there will be a folder in it named “samples” which is shown in Figure 3. Copy this folder into the “Recognition” folder that was downloaded off the Github. The second folder that needs to be copied over is a folder called “logs”. This will be located in the same place that you were able to find “samples”. Finally the last folder that you need to locate will be called “mrcnn.” “mrcnn’ will also be needed in order for the Mask R-CNN to be able to run. Figure 3 you will be able to see the location of all three of these folders that are to be moved to the “SU-ECE-20-4-master” folder.

A screenshot of a video game

Description automatically generated

Figure 3: Location of the three folders needing to be copied

After this process is complete, next you have to download a software called TensorFlow. The website as well as instructions on how to install the software can be found here: <https://www.tensorflow.org/install>. In order to run Mask R-CNN a version greater than TensorFlow 1.3.0. The files that are in the Dropbox will work for every version of TensorFlow that is greater than or equal to 1.3.0, but older than 2.2.0.

If you are using a version of TensorFlow that is greater than 2.2.0, there will be a few lines that need to be changed. The file that will need to be updated is called model.py and can be found in the “mrcnn” folder. There are only three lines that need to be changed. Figure 4 shows where the model.py file is located.

A screenshot of a video game

Description automatically generated

Figure 4: Location of model.py to be updated

On line 758 of model.py, this needs to be changed from:

*“keep = tf.sets.set\_intersection(tf.expand\_dims (keep, 0), tf.expand\_dims(nms\_keep, 0))”* to

*“keep = tf.sets.intersection(tf.expand\_dims(keep,0),tf.expand\_dims(nms\_keep,0))”*

On line 761 of model.py, this needs to be changed from:

*“keep = tf.sparse\_tensor\_to\_dense(keep)[0]”* to

*“keep = tf.sparse.to\_dense(keep)[0]”*

And finally on line 774, this needs to be changed from:

*“tf.to\_float(tf.gather(class\_ids, keep))[..., tf.newaxis],”* to

*“tf.dtypes.cast(tf.gather(class\_ids,keep),tf.float32)[...,tf.newaxis],”*

Figure 5 shows the newer version’s lines implemented and older version’s lines commented. Once these changes are made, you are ready to move on to setting up the remainder of Recognition.py.

A screenshot of a cell phone

Description automatically generated

Figure 5: Modified lines in model.py for Tensorflow 2.2.0

## Understanding and Modifying config.json

When you first open the files from the GitHub repository, you will see a file named ‘config.json’. Recognition.py has variables that help with the program set up and the image-matching process that you may want to change in-between different runs. This file acts as a quick and simple way for the user to change these runtime variables without having to open and modify the Python program. You can open this file in any text editor (*Notepad* is used in Figure 6), change the variable within the quotations to your new desired value, save the file, and rerun the program with the new parameter set.

A screenshot of text

Description automatically generated

Figure : Screenshot of config.json in Notepad

This is where you are able to change the templating option that Recognition.py will run. If the templating is set to 0, then it will run the manual ROI section of Recognition.py. A 1 will run Recognition.py with the MATLAB-generated templates that have been previously created. Changing this to 2 will run the Mask R-CNN section of the program, and the templates will be generated during the run time of the program.

Table 1. Templating method from editing "templating" field of config.json

|  |  |
| --- | --- |
| **String in “templating” field** | **Templating method during run of Recognition.py** |
| “0” | **Manual ROI** drawn by user at runtime based on “images” folder |
| “1” | **Pre-generated templates** in “templates” folder |
| “2” | **Mask R-CNN templates** generated at runtime based on “images” folder |

## Filling Out easy\_run.py

One of the files that is in the Recognition folder from GitHub is called easy\_run.py. This file is where all path information will be filled out that Recognition.py uploads and uses. These paths are extremely important for the correct operation of Recognition.py. A template of easy\_run.py will be in the downloaded Github folder and will look like Figure 7 below.

A screenshot of a cell phone

Description automatically generated

Figure 7: easy\_run.py template to be filled out

The only changes that need to happen to this folder is to replace the tilde (~) with the correct path information. Wherever these files are located, is what should be placed into the str() command. Python may run differently in terms of how it determines the path on macOS vs Windows. Windows may use either the “\” or “//” to separate the folders in the path, while macOS can use “/”. A version of a filled out easy\_run.py is in Figure 8.

A screenshot of a cell phone

Description automatically generated

Figure 8: Completed easy\_run.py file with macOS paths

This is the completion of easy\_run.py. After this we must ensure that the folder is set up correctly in order for Recognition.py to run.

## Constructing the Full Working Directory

Now that we have covered the individual entities within the folder, we will now show you how to set up the full working directory so that Recognition.py works smoothly. Below, Figure 9 shows a correctly initiated working directory in MacOS.

A screenshot of a cell phone

Description automatically generated

Figure 9. Example of a working directory

You can see that all folders and files are contained within the working directory called ‘set\_1’. The ‘images’ and ‘templates’ folders are both correctly named and ‘config.json’ is present. You will also notice the folder called ‘destination’ and the file called ‘score\_matrix.csv’. Both of these entities are created when the program is run so you do not need to have them included prior to running the program. Also, if you desire, you can store whatever other files and folders in this outer folder. Recognition.py simply looks for the folders named ‘images’ and ‘templates’ and the file ‘config.json’ and it will ignore everything else. Finally in order for the image processing to write files correctly, there needs to be a folder within the image directory that you’re working in. The folder should be named ‘edited\_photos’. This is where any original image that was edited will be saved. In order for one image to be processed too much and lose its quality, the original image must replace the edited image after each run. Figure 10 will show the folder, within the ‘images’ directory, that needs to be present for the image processing to be run successfully.

A screenshot of a cell phone screen with text

Description automatically generated

Figure 10. edited\_photos folder for original copies

# Running Recognition.py

## Opening the Command Line

For Windows:

Open *File Explorer* and navigate to the folder that holds Recognition.py. Hold the ‘shift’ and right-click in any whitespace then select ‘Open PowerShell window here’. In Figure 11 below, the orange line highlights the presence of Recognition.py and the blue line shows the correct option in order to open PowerShell.

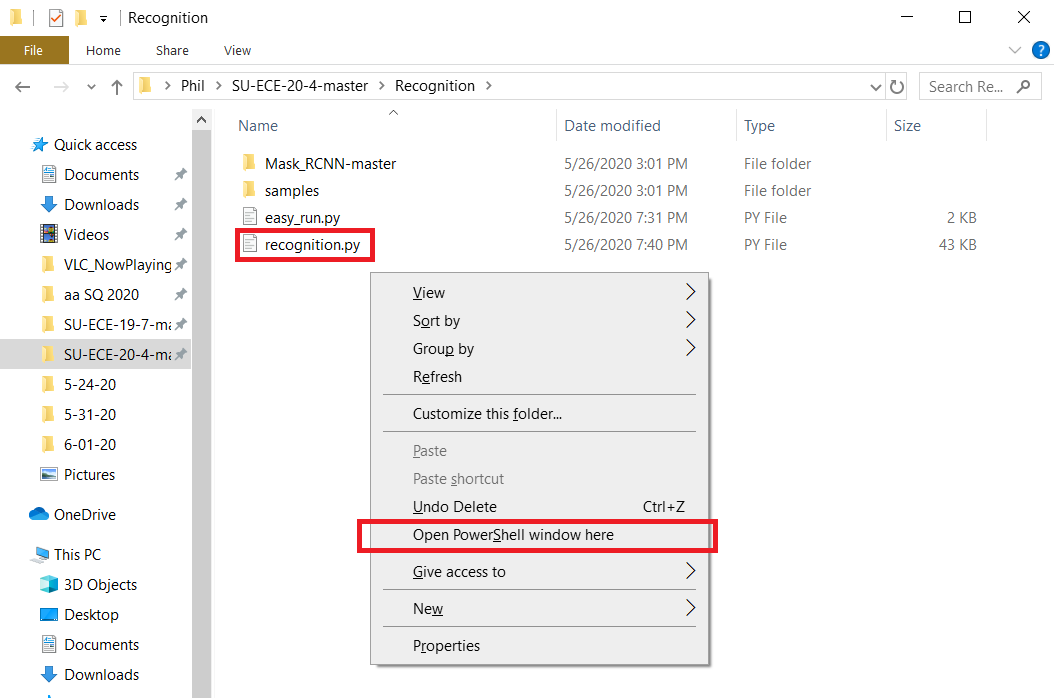


Figure 11. How to open the command line (Windows)

### For Linux/Mac OS:

Open terminal. Navigate to the Recognition Folder. Then type python easy\_run.py.

## Using the Command Line Argument

Recognition.py has a single, optional command line argument. It allows the user to specify the desired working directory to use for that run of the program. You may use this argument if the Recognition.py file is not in the same directory as your image and template folders. The below examples will better explain when to use the command line argument.

Not necessary: If no command line argument is given, the program will set the working directory to the folder that Recognition.py currently resides in. Figure 12 shows that Recognition.py is present in the same folder that holds the images, templates, and config files. If this is how you choose to configure your working directory, then you do not need to use the command line argument. To run the program, you would simply type **python recognition.py**

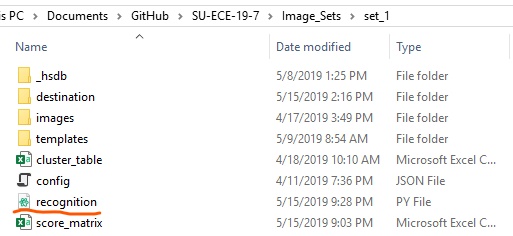


Figure 12. Example of when not to use the command line argument

Necessary: If you plan on making your own custom changes to Recognition.py, you won’t want to have multiple different copies of the Python script in each separate image set. Tracking each file and the different changes would quickly become a hassle. In this case, you can have a singular copy of Recognition.py and use the ‘work\_directory’ command line argument to dynamically change the location of the image set to be worked on. In this case, you will still open PowerShell in the folder that holds Recognition.py. Then use File Explorer to navigate to the desired work directory. Right-click in the address bar and select ‘Copy address as text’, Figure 13 gives a visual representation. Then, return to PowerShell. To run the program, you would type **python recognition.py -work\_directory “\_\_xx\_\_”** . Copy and paste the directory address in-between quotation marks and do not include the period at the end. This will correctly point the program towards the correct working directory.

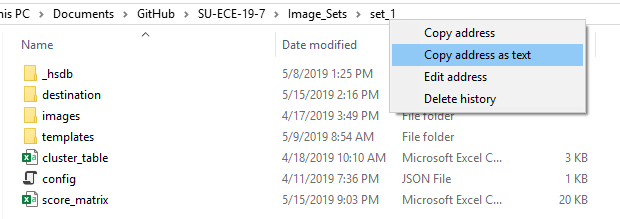


Figure 13. Extracting the work directory address

## Running the Program

As discussed above, there are two ways to run the program. The first scenario does not utilize the command line argument. Once all the proper set up has been completed and the command line window is open, type **python recognition.py**. An example is seen below in Figure 14.

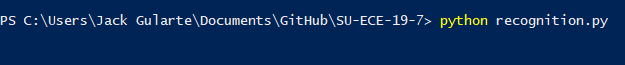


Figure 14. Basic command line

The other option is to use the command line argument. The command line argument is labeled ‘-work\_directory’. The correct way to run the program is shown in . Keep in mind that this example is on a Windows system which uses forward slashes while Linux and Mac OS use back slashes.



Figure 15. Using the command line

# How Recognition.py Works

The basic idea of Recognition.py is to take the camera trap images and, through various methods, determines if a pair of images contains same snow leopard. By inserting an image set into the program, the program determines “keypoints” on each image using the SIFT algorithm (scale-invariant feature transform). The program compares each image’s keypoints to each other image to test if two snow leopards are a “match,” producing a “score matrix” to list how many matches were between each image.

ECE 20.4 set out to enhance the Recognition.py program with new functionality. Before matching an image set, the program checks if the image needs to be enhanced. Mask R-CNN allows for runtime templates to be generated, eliminating the previous dependence on hand-drawn or MATLAB-generated templates. The two clustering techniques are integrated into the runtime of Recognition.py and upon conclusion of the program, two outputs are generated.

The layout of the program through “main” shows the functionality and expected output on an image set.

* First, the program uses the paths specified by easy\_run.py and the templating and other options in config.json.
* Images in the image set are checked to see if editing is needed, if the image is blurry or taken at night.
* Then, a list of Recognition objects is made. A Recognition object contains the image related to a snow leopard image from the image set and holds the extra locational information from the image title.
* A template is added to the Recognition object, based on the choice of templating. For Manual ROI templating, draw a box around the part of the image to mask and press ‘C.’ For pre-generated templates, the add\_templates function looks to the “templates” folder. For Mask R-CNN templating, the Mask R-CNN weights and snow leopard class are loaded, then masks are drawn and shown in windows as they are made, which can be closed normally. The Manual ROI and Mask R-CNN methods write these templates to their own folders in the working directory.
* The matching process begins by assigning slices to a number of threads (specified in config.json), then generates SIFT keypoints and comparing between each image. While the SIFT keypoints are being used, SIFT descriptors are used as inputs to a clustering function that guesses the number of snow leopards in the image set.
* After the multi\_match function is completed, the score matrix is generated and written to the destination folder.
* Clustering functions that use score matrix values (of the number of keypoint matches between image pairs) are called to guess the number of snow leopards in the image set.

# Recognition.py Output

The main output of Recognition.py is called the ‘score\_matrix’. It is a comma-separated values (.CSV) document that can easily be opened in Microsoft Excel or a similar program. The second output to Recognition.py will be the clustering results. One of these results will be a graph that is displayed after the runtime. This graph will be the elbow graph generated by k-Means clustering.

A close up of a map

Description generated with very high confidence

Figure 16: Elbow graph of the k-Means output

The second clustering output will be displayed on the terminal and it will be the cluster graph as well as the shape of the cluster graph that was generated. The x value n this shape will equate to the amount of clusters that was generated. Figure 17 shows this output.



Figure 17: Markov clustering output

# Addendum

Changes from ECE 19.7’s User Guide to ECE 20.4’s User Guide:

* Edited section “Installation” to include ECE 20.4’s GitHub repository of Recognition (<https://github.com/caballe4/SU-ECE-20-4>) instead of ECE 19.7’s GitHub repository of Recognition (<https://github.com/devindewitt/SU-ECE-19-7>)
* Added section “Installing and Setting Up Mask R-CNN”
* Edited section “Understanding and Modifying config.json” by adding “templating” field table
* Edited “Filling Out easy\_run.py” section based on Windows/Mac differences
* Added section “How Recognition.py Works” section that walks through the parts of “main”
* Edited section “Recognition.py Output” to add clustering function outputs