# Recognition.py Function Descriptions; Updated as of 6/1/2020

This document details all manually created functions for recognition.py. Functions called from python packages are not detailed here but, can be found in either the [Python 3.7 documentation](https://docs.python.org/3/) or the [OpenCV documentation](https://docs.opencv.org/3.4.2/). The functions are listed in order of program execution.

## Class: Recognition

**Class Variables:**

* image\_title; holds the title of the image.
* image; holds the actual image.
* template\_title; holds the title of the corresponding template.
* template; holds the actual template.
* station; the station where the image was captured.
* camera; the camera that took the image.
* date; the date the image was taken.
* time; time of when the image was captured.
* cat\_ID; the corresponding ID to the leopard in the image.

**Class Functions:**

* \_\_init\_\_; called to initiate the object. Sets all variables to an empty string.
* add\_image; adds the image title and the image to the class.
* add\_template; adds the template title and the template to the class.
* add\_title\_chars; adds the title characteristics to the class.
* add\_cat\_ID; adds the cat ID to the class.

## “\_\_main\_\_”

**Inputs:** none.

**Outputs:** none.

**Calls:** get\_path(), init\_Recognition(), add\_cat\_ID(), match\_multi(), normalize\_matrix(), check\_matrix().

**Notes:** The main program driver. Does all system ‘Path’ initiation and calls all major functions. Writes the resulting score\_matrix.csv to the work directory.

## get\_path(work\_directory, folder)

**Inputs:**

* work\_directory is a system ‘Path’ object that details the user specified work directory that holds the image and template folders.
* Folder is the name of the folder Path to be found from the work directory.

**Outputs:**

* Outputs a ‘Path’ object that locates the desired folder. Returns an empty string if the folder cannot be found.

**Notes:** It is used in a loop to find ‘images’, ‘templates’, and ‘destination’ folders on three separate calls. It will create the destination folder if it does not find one. If either images or templates is not found it will return an empty string and main will throw an error and cease running.

## init\_Recognition(paths['images'], paths['templates'])

**Inputs:**

* path[‘images’] holds the system path to the images folder.
* path[‘templates’] holds the system path to the templates folder.

**Outputs:**

* A list of Recognition objects called rec\_list.

**Calls:**

* getTitleChars(t).

**Notes:** This function is called to pair each image with its corresponding template and place them together in a single Recognition object. It uses a parallel for-loop to concurrently iterate through the image and template folders. It adds the image title, the image, the template title, and the template to the object. It also adds the results of getTitleChars(t) to the object.

## getTitleChars(title)

**Inputs:**

* title is the image title.

**Outputs:**

* Four variables that denote the station, camera, date, and time of when/where that picture is taken.

**Notes:** Relies on a very specific image naming convention. This shouldn’t be an issue as long as you don’t change the name of the images.

## add\_cat\_ID(rec\_list, cluster\_path)

**Inputs:**

* The rec\_list that is created by init\_Recognition.
* The system ‘Path’ object specifying the location of the ‘cluster\_table.csv’ file.

**Outputs:**

* An updated rec\_list with the cat\_ID added to each object.

**Notes:** If the cluster\_table.csv table is available, you can run this function and it will add the appropriate cat\_ID to each Recognition object. Can be used to see the average score of an correct match and a correct miss. If the cluster table is not available for a specific image set make sure to comment the function out.

## match\_multi(primary\_images, image\_destination, n\_threads, write\_threshold)

**Inputs:**

* primary\_images is the ‘rec\_list’.
* image\_destination is the system Path object to the destination folder.
* n\_threads is the number of threads to use imported from the config file.
* write\_threshold is the lower threshold to use when determining whether to write an image to the destination folder.

**Outputs:**

* The resulting score\_matrix.

**Calls:** slice\_generator(), match()

**Notes:** Uses multithreading to create multiple threads that all execute the ‘match’ function. Will wait for all children threads to finish before returning the score matrix.

## slice\_generator(num\_pictures, n\_threads):

**Inputs:**

* num\_pictures is derived from the length of the primary images list. Holds the number of images in the dataset.
* n\_threads is pulled from the config file. It’s the number of threads the user wants to use during execution.

**Outputs:** Returns an array of tuples, specifically two integers, representing beginning and ending indices for breaking up the program into multiple processes.

**Notes:** This function was written by Ross Pitman. It takes the number of pictures and the number of threads and evenly breaks it up into a start and end index that each thread will execute through. The number of threads **must** be less than the number of pictures or the program will throw an unfriendly error.

## match(primary\_images, secondary\_images, image\_destination,

## start\_i, score\_matrix, write\_threshold)

**Inputs:**

* primary\_images is the rec\_list array split into each section for the multithreading process.
* secondary\_images is a deep copy of primary images but is the full list of images regardless of the multithreading that is implemented.
* image\_destination is the system ‘Path’ object that locates the directory to write the matched images to.
* start\_i is the starting index of each thread. This parameter is used to help correctly write the score\_matrix when multithreading is implemented.
* score\_matrix is a reference of the final score matrix that will be returned.
* write\_threshold is the lower bound of when to write an image to the destination folder. It denotes the lower cutoff of the number of keypoint matches

**Outputs:**

* score\_matrix is the final score\_matrix once it has been fully populated by the function.

**Calls:** score\_boosting(), write\_matches(), call\_cluster()

**Notes:** This function executes all of the computer vision and image comparison code. It implements a nested for-loop that iterates through the sliced primary images and the secondary images list. A SIFT object is created and the detectAndCompute cv2 function is called to extract the keypoints and descriptors. It then iterates through the secondary images and extracts those keypoints and descriptors as well. It then implements a knnMatch object and compares the lists of keypoints. Then the score\_boosting function is called and the final score for that image pair is written to the score matrix. Then it checks if the final number of matching keypoints passes the write threshold and calls write\_matches() if the statement is passes. It will throw an error if the final number of keypoint matches between two images is 0. This error is caught so that if it happens the program is not interrupted or compromised.

score\_boosting(primary\_image, secondary\_image, good\_points)

**Inputs:**

* primary\_image is a single recognition object that holds the primary image being compared.
* secondary\_image is a single recognition object that holds the secondary image being compared.

**Outputs:**

* score is the final comparison score between the two images that will be written to the score matrix.

**Notes:** This function implements the score boosting functionality that is present in HotSpotter. When each image is captured by the camera trap, the station number, camera number, date, and time are all recorded and appended to the image name. We can confidently say that there is a higher chance that if two pictures are taken by the same station and camera and on the same date, then there is a higher chance that they picture the same cat, so we boost the score that was received from the image comparison process. Each level of image similarity can be boosted to some degree and those values can be changed via the config file.

## write\_matches(kp\_1, kp\_2, good\_points, primary\_image, secondary\_image, image\_destination)

**Inputs:**

* kp\_1 is a list that holds the keypoints found in the primary image.
* kp\_2 is a list that holds the keypoints found in the secondary image.
* good\_points is the combined list of keypoints between the two images that have been determined to be a match.
* primary\_image is a single recognition object that holds the primary image.
* secondary\_image is a single recognition object that holds the secondary image.
* image\_destiantion is the system ‘Path’ object that holds the location of where to write the final image.

**Outputs:** none.

**Notes:** This function is responsible for writing the matching images to the destination folder. Most of it was written by Ross Pitman.

## normailze\_matrix(score\_matrix)

**Inputs:**

* score\_matrix is the resulting score matrix returned by the match() function.

**Outputs:**

* A new version of score\_matrix.

**Notes:** This function is responsible for normalizing the score matrix. That means that it finds the highest value in the matrix and divides all values by that number. This brings each index to a value that lies in between 0 and 1. This is necessary for the clustering steps to work correctly. Once the score matrix is normalized, the identity matrix is added to it so that the main diagonal is populated with 1’s. This is because the diagonal represents the score when an image is compared against itself which, should receive the maximum score of 1.

## check\_matrix(rec\_list, score\_matrix)

**Inputs:**

* rec\_list is the list of recognition objects that is created by init\_Recognition().
* score\_matrix is the score matrix after it has been normalized.

**Outputs:** none.

**Notes:** If the cluster table is available and you ran the add\_cat\_ID function then you can also run this function. This was more of an experiment on Jack Gularte’s part to see what the average score of a ‘match’ and a ‘miss’ is. Since the score matrix is in the same order as rec\_list, it iterates through each 2D index and checks the two cat\_ID’s from each object and adds the score to the correct variable. It then prints to the terminal the final result.

## test\_accuracy(rec\_list, score\_matrix, threshold)

**Inputs:**

* + rec\_list is the list of recognition objects that is created by init\_Recognition().
  + Normalized/ non-normalized score\_matrix.
  + threshold of matching.

**Outputs:** none.

**Notes:** this function determines the number of correct and incorrect matches in Recognition using an input threshold and a cluster table identifying images in the dataset. The function will iterate through the resulting score matrix and count the true positive, false positive, true negative, and false negative matches and prints the accuracy, recall, specificity, and precision of the matching process.

## kmeans\_clustering(descriptors\_list, k\_range)

**Inputs:**

* + descriptors\_list is a global list that is populated with the descriptors of each image in dataset during the matching process.
  + k\_range is the maximum number of *k* that algorithm will run for clustering.

**Outputs:** none.

**Notes:** an elbow plot will be returned if this function is called to determine the correct number of clusters in the dataset. The elbow of the plot is the predicted number of clusters.

## kmeans\_score\_matrix (score\_matrixCSV)

**Inputs:**

* + score\_matrixCSV is the normalized score\_matrix that generated by the matching process.

**Outputs:** none.

**Notes:** an elbow plot will be returned if this function is called to determine the correct number of clusters in the dataset. The elbow of the plot is the predicted number of clusters.

## Markov\_cluster(mark\_array)

**Inputs:**

* + Square array either of the score matrix or the SIFT descriptors

**Outputs:**

* + The cluster array that was generated by the Markov clustering algorithm as well as the size of this cluster array

**Notes:**

* + The shape of the cluster array is printed and the x value will be the number of clusters that was generated. For example, if the shape is (4,10), four individual snow leopards were identified.

## filter\_images(primary\_images, image\_source, edited\_source)

**Inputs:**

* + Current image that is being uploaded to Recognition, the path from where the image came from, and the path to send original images.

**Outputs:**

* + Enhanced photo, if necessary, written to the original image folder. The unedited photo saved in the “edited\_photos” folder to be kept for later use.

**Calls:** make\_lut\_u(), make\_lut\_v(), edge\_sharpening(), histogram\_equalization()

**Notes:**

* + Cannot be run at the same time as the Mask R-CNN templating because there is an issue that occurs during the neural net templating.
  + Original image must replace the edited image after run of the program to avoid over processing of the image.

## edge\_sharpening(save\_image)

**Inputs:**

* + Copy of the original image to be sent to editing

**Outputs:**

* + Image after the edge sharpening has been performed

## histogram\_equalization(sharp\_image)

**Inputs:**

* + Image that just finished the edge\_sharpening

**Outputs:**

* + Image that had just gone through the histogram equalization process and saved to the image destination folder

## call\_cluster(arr)

**Inputs:**

* + Square array of SIFT descriptor values collected during matches

**Outputs:**

* + Normalized values of SIFT descriptors

**Calls:**

* + markov\_cluster()

mrcnn\_templates(rec\_list, image\_source, snowleop\_dir, weights\_path)

**Inputs:**

* + Rec\_list, Image directory also may have to change path of IMAGE\_DIR, trained snow leopard's directory, path to the snow\_leopard weight ie bottle...h5,

**Outputs:**

* + The rec\_list or recognition objects with inserted Mask RCNN templates

**Calls:** calls the function in the snow\_leopard class to enable Mask\_RCNN is fully loaded also reshapes the images to be the same to keep consistency

**Notes:**

* + Will either be ran or not depending on the configurations decided. The config file of the image set must have “2” for templating for mrcnn\_templates to run.
  + Paths of the folders and weight must be correct for mrcnn\_templates to run properly. The function mrcnn\_templates performs exactly like the add\_templates except it will use the functionaly of Mask RCNN from <https://github.com/matterport/Mask_RCNN>

# Addendum

Changes from ECE 19.7’s Function Description to ECE 20.4’s Function Description:

* Added “mrcnn\_templates” function
* Added “call\_cluster” function
* Added histogram\_equalization function
* Added edge\_sharpening function
* Added filter\_images function
* Added markov\_cluster function
* Added kmeans\_score\_matrix function
* Added kmeans\_clustering function
* Added test\_accuracy function
* Modified match function