AutoQuerying Dev Guide

# AutoQuerying Overview

Autoquerying is the process where we create a recognition/association matrix that represents the relationships between chips. It consists of recognition and population of a score matrix. We begin by generating a matrix by querying every chip against other chips. When each chip is queried we have to compare that one chip to all other chips in the chip table. This was set up so that it would be in parallel, however this was giving us many errors so we decided to move to doing this in series. We simply forced HotSpotter to perform computations in series, thus serving as a sort of band-aid. We have changed the file structure somewhat since this change, so there is a chance that parallel feature computation is possible now. This computation is called querying in HotSpotter, but is a form of instance recognition, so these terms will be used interchangeably throughout this document and elsewhere.

The recognition algorithm operates on rectangular regions of interest known as chips. Within these chips, it compares elliptical areas of similarity known as hotspots, or more generally, patch-based features.

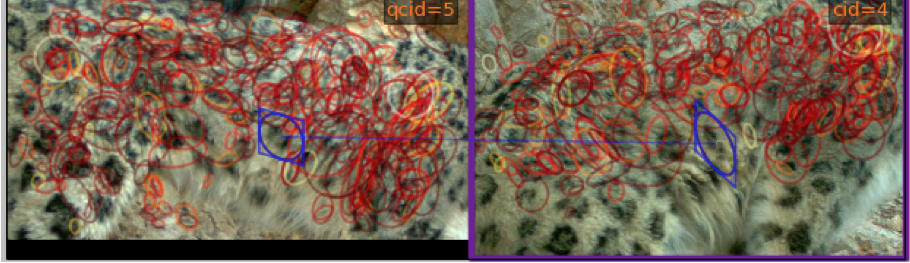


Figure 1: Example of hotspots on the body of a snow leopard

The keypoint associated with each feature contains information about xy-location of the feature within the image, as well as scale, orientation, and shape.[1]This additional information allows the program to account for the size of the feature, as well as different angles of viewing the animal and different poses, effectively allowing recognition between one sighting of an animal and another. However, since it is not desirable that hotspots be allowed to transform freely in reference to nearby hotspots, HotSpotter implements a process known as spatial verification. Spatial verification limits the potential chip-to-chip feature matches by ensuring that hotspots transform similarly to their neighbors. Any feature match that is found to be spatially inconsistent is not considered when computing a similarity score.

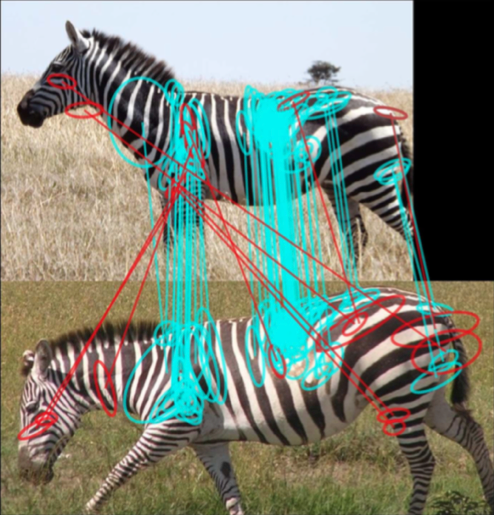


Figure 2: Example of spatial verification on a zebra image. Inconsistent matches are shown in red

Spatial verification works by fitting a rigid projective transformation to the chip. Currently, spatial verification only checks spatial location. The actual recognition is performed using an algorithm similar to Local Naïve Bayes Nearest Neighbor search [1].

The HotSpotter recognition algorithm is well tested on animals with relatively rigid bodies, e.g. zebras. However, snow leopards present a challenge in that their bodies are particularly lithe and flexible, and their fur is long. This fluffiness can sometime occlude identifying features such as spots. The effect these constraints will have on recognition accuracy is unclear, and thus it is necessary to thoroughly test the accuracy of the existing algorithm and determine if modifications are necessary to improve performance. This will involve testing on many cases of snow leopard images (e.g. cats at varying distances from the camera, levels of lighting, body position, etc.) and recording the results. Problem cases will be noted, and from this information, as well as knowledge of the recognition algorithm, the team will determine if modifications to improve the accuracy of the recognition algorithm are possible. Some possible candidates that are under consideration are modifications to the spatial verification, e.g. expanding it to check additional dimensions (possibly scale and orientation), or modifying the transformation it uses to map hotspots, possibly implementing a non-rigid projective transformation.

To automate HotSpotter ECE 17.7 created created functionality in autoquery.py that walks through the whole chip table and queries a chip against every other chip. This is done using functions listed in the MCL/Parsing ICD. However if there are no keypoints in a chip then it will return a string for the match between the two chips. We correct for this by assigning all strings to zero. HotSpotter however does not give the same scores from chip to chip. For example when querying chip 1 we might find chip 2 matches with a score of 500, but when chip 2 is queried we may find that chip 1 gets a matching score of 50. To fix the issue of asymmetrical scoring, ECE 18.7 chose to take the maximum score between two chips, which proved to increase accuracy especially with non-boosted scores. To cluster these chips we need an undirected graph. We create an undirected graph by averaging the scores from the queries and putting them into the score matrix. We must also take into account that a high score in one query may not correlate to a better match than a lower score in another query. To adjust as best that we could we decided that we would normalize all scores from an individual query based on the highest score from that query. That way we would not have such large changes in values as we would get if we did not do this. After talking to our faculty advisor Rana Bayrakçismith, we were told that we can know with a fairly high certainty that if images were taken within 15 minutes from each other that the snow leopard in that image are the same snow leopard. With this knowledge we decided that we would add to the recognition scores if the chips were taken from images within 900 seconds (15 minutes) or if chips came from the same image. The image name we are provided with from Panthera has metadata that is explained in MCL/parsing ICD. We make sure however that by adding this we do not let the score go above 1 because we want the entire matrix to be normalized. The parameters for same set and same image can be changed under options -> edit preferences. More information about AutoQuerying and ECE 18.7’s proposed changes to AutoQuerying can be found in the AutoQuerying Dev Guide.

Hitherto, we can see that AutoQuerying is comparing all the chips together without the knowledge of the perspective of each chip. This means that AutoQuerying is spending a lot of time comparing chips of different flanks to each other, for example, chips from the head to chips from the tail, chips from the left flank to chips from the right flank, and so on. This process is not only unnecessary, because snow leopards are not patterned symmetrically, but it also requires more processing power and time, and raises the probability of false positives.

Querying by orientation is a new feature that ECE 18.7 experimented in order to improve the performance and accuracy of AutoQuerying process. The proposed method only compares chips that are from images that capture the same flank of snow leopards. For example, chips from images that capture the left flank will be compared together, chips from images that capture the right flank will be compared together, and so on. One disadvantage is that HotSpotter would need user assistance to identify what side of the snow leopard is being shown in each image, thus reducing the autonomy of the program. However, there is an opportunity to try and train a machine learning algorithm to do this recognition.

# Classifying Images by Orientation and Number of Snow Leopards

To acquire the knowledge of the orientation of a snow leopard captured in each image, we propose adding an additional module to prompt input from user. This module could be named ClasssifyByFlank, and would be launched automatically after the user has imported all the desired images. Once all the images are successfully added into image\_table.csv, ClasssifyByFlank would pop up as a window and guide the user through each of the imported images. A mock-up of this user interface is shown in Figure 1. The window would show an image alongside a prompt asking the user to choose the side of the snow leopard that the image captures. Options to choose for the sides are: left flank, right flank, front/face and back/tail. The user can choose more than one orientation if multiple angles are pictured.

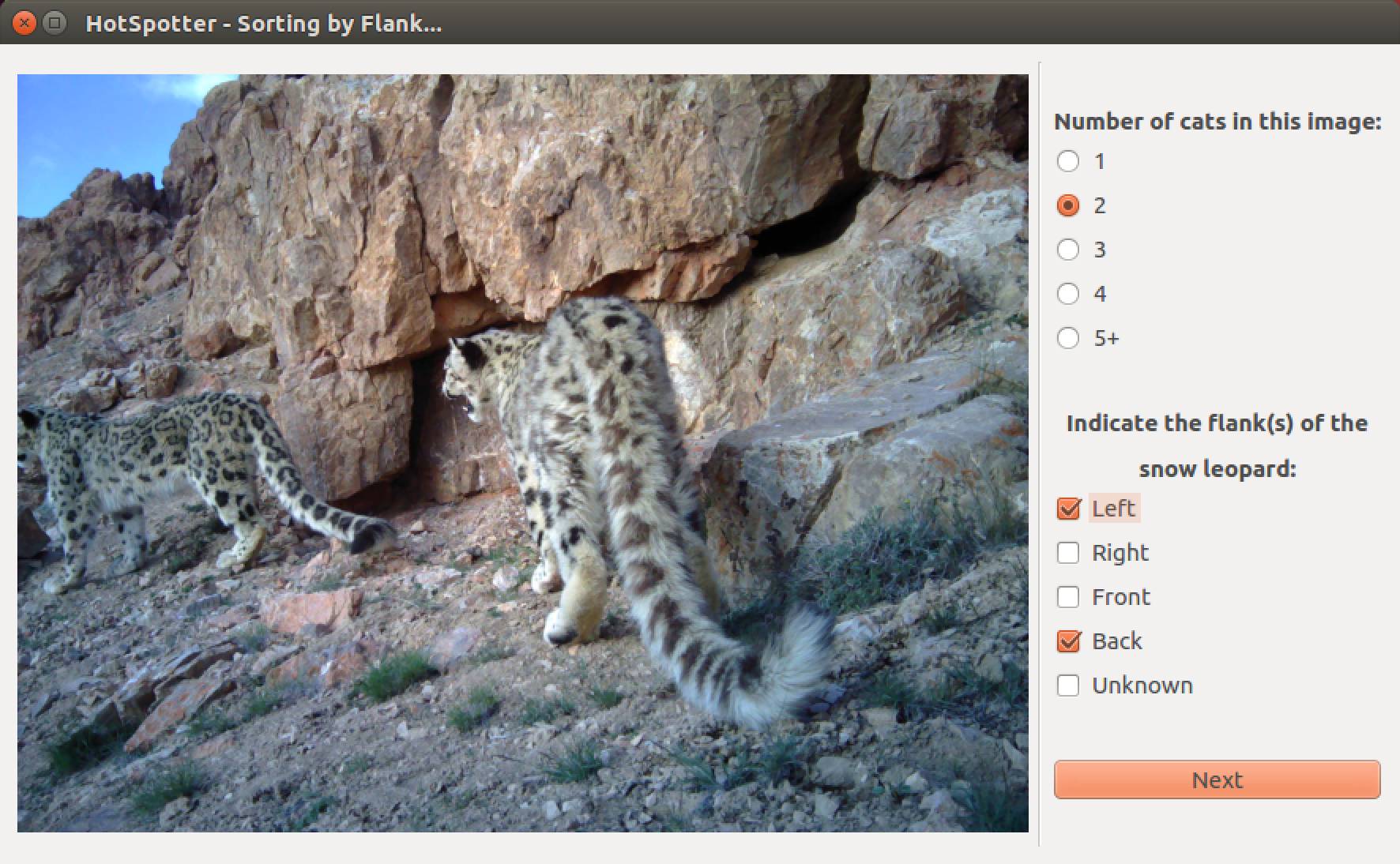


Figure 1 Pop-up window of the proposed ClassifyByFlank module.

In addition to the prompt for the flank of snow leopard shown, there would be also a prompt for the number of cats in the image. By knowing that there is more than one snow leopard in an image, we have the option to set these images aside from AutoChipping, AutoQuerying and Clustering, since these processes currently don’t work well for images with multiple snow leopards.

Once the user has checked all the imported images, ClasssifyByFlank will create two additional columns in the image\_table.csv, and save the orientation labels and number of snow leopards in each image under these two columns according to data recorded from user’s input.

# Accommodate the Knowledge of Orientation into AutoChipping, AutoQuerying and Clustering

Figure 2 New strategy for accommodating flank data into existing process of HotSpotter

Figure 2 shows how current algorithm of HotSpotter can be adapted to make use of the knowledge of orientation/flank to improve the recognition accuracy. AutoQuerying can also utilize the knowledge of what side is associated with each chip to only query chips that have the same orientation label. Each time AutoQuerying compares chips of same orientation, it creates a separate score matrix and saves it to disk as scores.csv. At the end of AutoQuerying, there would be four different score matrices and four different scores.csv files, with each score matrix and scores.csv corresponding to one orientation. Clustering then clusters each of the four score matrices and identifies the number of distinct cats in each orientation set. An additional process, Cross Compare, would run once Clustering is done to link chips from different sides of the same cat together. This additional process can compare the image metadata (location, date and time) of each image associated with an identified cat in the set of each side images to the data of location, date and time of each image associated with an identified cat from the other sets of similarly orientated images. If these images were taken at the same location, and the time these images were taken differ by less than 15 minutes (time adjustable by user), these images would be grouped together and labeled as one cat. This process would allow HotSpotter to identify multiple sides of some snow leopards thus providing a more realistic population estimate