

Part A

In part A I was tasked with creating software capable of creating target composite images given a set of source images. The main function which is used to output the two composite images for “mosaic\_target1.jpg” and “mosaic\_target2.jpg” is contained within the main.matlab file.

**Image Preparation**

As the first step of the software, it prompts the user to enter the desired size of the composite image. It then proceeds to resize the source image by resampling it using the matlab method *imresize.*

Unlike normal resizing, this method uses bicubic interpolation to add pixels to images when increasing their size , thus reducing the loss of quality of due to resolution change.

When shrinking the image the method applies aliasing to reduce artifacts which appear , due to the fact that there are now fewer pixels in the image.

In addition to resampling , the application applies Gaussian smoothing filters to reduce the amount of noise.

The application then proceeds to request the number of horizontal and vertical tiles the composite image should be consisted of. Next thing which needs to be specified is the folder with the images used to create the tile

The image is then split into tiles and the set of source images is resized to match the tile size. After which the program starts computing histograms.

Histograms and similarity measure

The software evaluates image similarity by measuring the similarity of histograms. Firstly, for every image a 3-dimensional histogram is constructed ,i.e. for each RGB channel.

Secondly, a similarity measure is established between a given tile in the target image between all the source images, this is quite a heavy process because if there are a lot if images it has to perform computations for each one of them.

The similarity is measured by calculating the Euclidean distance for each RGB channel and then producing the final similarity measure by using the formula for the weighted colours.



By balancing the weights of the colours its possibly to find a better match for an image if , for example you know its red.

Finally, the similarity measures are stored in a special array on which the max function is used to find the index of the cell which has the highest similarity to the destination tile , when the index is found the image residing in that position replaces the tile in the composite image. This process is repeated until there are no unprocessed cells left in the target image.

Here are the images for the outputted composite images

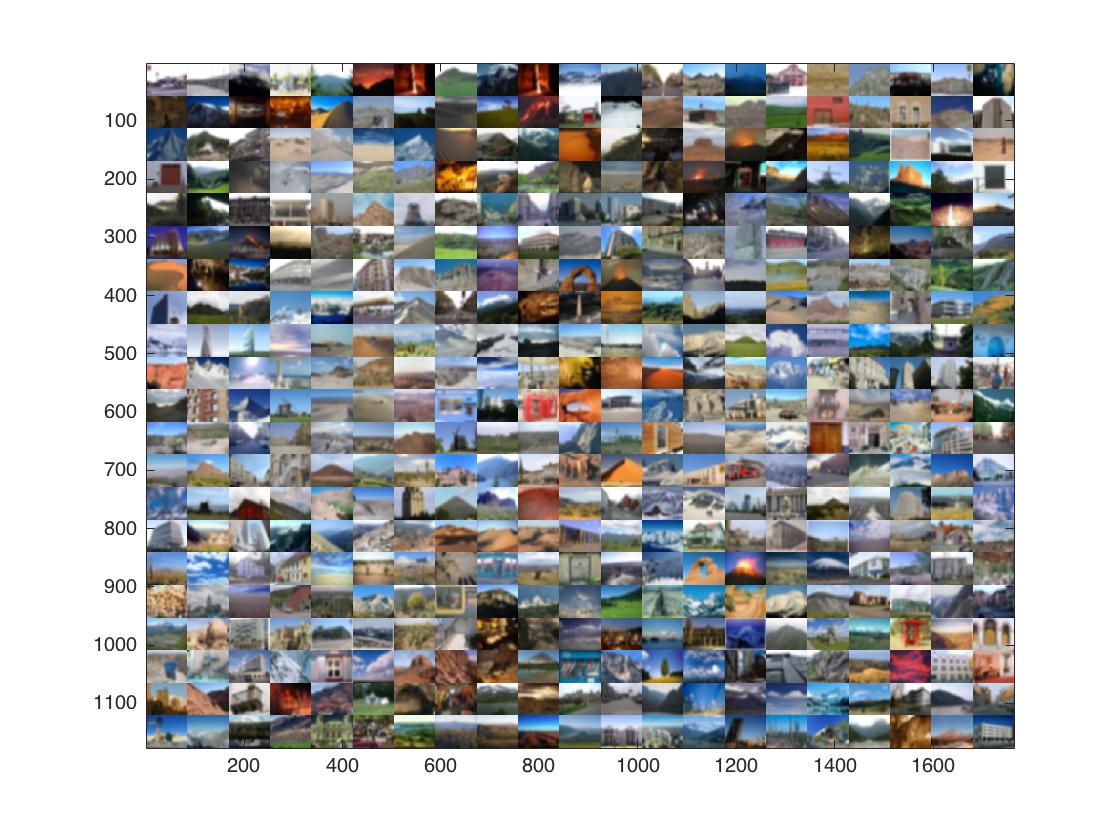


Figure Mosaic target 1



Figure Mosaic target 2

Part B

For part B ,the main function is located in mainB.matlab. The program reads all the images from the folders which contain the learning set images , both the manmade and the natural ones. The program however assumes that these folders are in the same folder as the script.

The program then proceeds to compute the histograms, as described in the previous sections , for both the manmade and the natural images , all while storing the results in designated arrays.

# k-nearest neighbor algorithm

After calculating and storing the histograms for the learning set of images ,we now proceed to read the images from the test folders. The software calculates the similarity between every test image and the test image in the form of Euclid distances and sorts in ascending order , thus the most similar ones are in the beginning of the arrays.

Following that , the software calls the function where the k-nearest neighbor algorithm is implemented. The user is also prompted to enter the value he wishes to have as k , when the program receives the input it looks at the arrays which contain the Euclid distances for the manmade and natural images. It then finds the shortest distances and depending records which array they belong to : if the source is the array with the manmade image similarities then it classifies the image as manmade , and vice versa for natural.

When going through the test images the program records whether there were any mistakes made and calculates the error rate as well as execution time.

Whats the k value for k-nearest?

|  |  |  |
| --- | --- | --- |
| K – value | Computation Time | Overall Accuracy |
| 1 | 6.20s | 60.6% |
| 2 | 5.55s | 44.60% |
| 3 | 5.81s | 63% |
| 4 | 4s | 52% |
| 5 |  | 66.20% |