AC51007 Report (Postgraduate Computer Vision Assignment 2016)

Authors: Vladislavs Ignatjevs, Yang Xu, Wei Li

1. Intro

The team created a Matlab project during this assignment. To run the project it is select project directory in Matlab and run *"image_converter.m"*.

The program has a graphical user interface that allows user to input target image and select various properties: output path, tile and subtile sizes, as well as an output image size . After pressing “Generate”, the program will classify the target image into manmade class or natural class, select corresponding folder to import composite images. Then, by using RGB comparison it will construct the final output image - tile by tile. The name of the output file is “output.jpg” and it written into “Generated image write path” specified.

2. Software choice

At the first team meeting every team member shared their own idea about the software choice. Pros and cons of each were discussed. This lead into decision to use Matlab development environment for the project.

3. Preprocessing used

3.1 Composite images preprocessing

Composite image preprocessing is done by importing composite images and adjusting them to a same square size along with Gaussian filter to blur each image and collect them into a cell array as ‘image pool’.

```
picCell{i,1}=imread(char(target_array(i)));
gauss = fspecial('gaussian',1,4);
temp1=imresize(picCell{i,1},[height width]);
adjCell{i,1}=imfilter(temp1,gauss);
```

3.2 Target image preprocessing

Target image preprocessing is done by importing target image and dividing it into blocks. Then, dividing target image into small blocks by using *mat2cell()* :

```
chunk=mat2cell(adjTar,repmat(tilesize,[1 ratioH]),
repmat(tilesize,[1 ratioW]),3);
```

Where *repmat(tilesize,[1 ratioH])* means dividing height into *'ratioH'* segments with each segment being *'tilesize'* long; *repmat(tilesize,[1 ratioW])* means dividing width into *'ratioW'* segments with each segment being *'tilesize'* long. In other words, the target image is divided into *ratioH*ratioW* number of blocks.

4. Classification

Classification was done by using K-nearest neighbor classifier to classify target images and judge them to decide whether they belong to manmade class or natural class. Due to fact that the judgement is based on the parameter K and the similarity between two images, the team made an attempt to find a good way to describe this similarity and find a K value to achieve the best accuracy.

Pixel Comparison

Pixel comparison was implemented by using distance between the pixels of two images, a good way to describe the similarity of them. The size of original images was different and some of them had too many pixels, so they were resized via Gaussian filtering which was also used in Part A and obtaining new images of the same size (100-by-100). The team also wanted to use a vector to represent the feature of a image, but such reshaping of 100-by-100 images would lead into the feature vector becoming 1-by-30000, resulting into the next calculation becoming very time-costing. To tackle this problem, decision was made to use PCA to decrease the diamond of feature vectors. With no doubts, by using PCA it is possible to decrease the correlation between different features of a feature vector and choose as many features as it is desired to keep after diamond decreasing. At the end of PCA, it is possible to obtain a matrix which contains the principal component coefficients. Due to fact the there were 1000 training images, the original feature matrix was 1000-by-30000. The coefficients matrix was 30000-by-n; it was possible to choose the value of n according to the number of features needed. After multiplying 1000-by-30000 matrix by the 30000-by-n matrix, the new feature matrix (1000-by-n in size) was obtained. The same method was used to obtain feature matrix of images in test set. By multiplying it by the coefficients of the matrix that was obtained from training set, the new feature matrix (of images in the test set) was obtained. After obtaining these two new feature matrices, it was possible to compare the distance of two different feature vectors and use KNN for classification. This process is shown in the chart below:

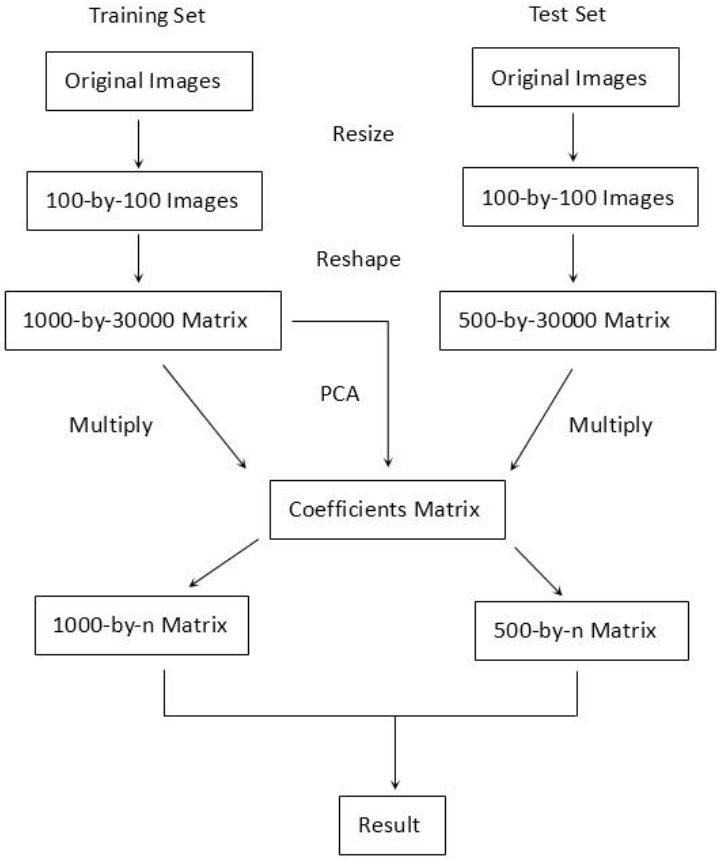

Figure 4.1: Pixel Comparison

Classification was done with n=300. This made it possible to keep 95.16% of the total features. However, the result was far from the initial expectation: the achieved accuracy was only 59.2%. The team expected that overfitting was one of the reasons that lead to this result, so decision was made to decrease the percentage of kept features. When n became 12, 63.91% of the total features was kept, while the accuracy sharply increased to 72.6%. Additionally, it was decided to classify with different size of resized images and use different distance to measure the similarity of two images, but the result did not differ provide a large difference. The team found out that the euclidean is the best measurement for the classification. The detailed statistics of this classification method can be viewed in the table below:

Computation Time	Error of Manmade	Error of Natural	Overall Accuracy
0.101s	32.4%	22.4%	72.6%

Table 4.1: Result of Pixel Comparison

RGB Histogram Comparison

It was decided to replace 100-by-100 image with the RGB histogram of original. 256 bins for each channel of RGB were and then the frequency percentage for each level of each channel was calculated. Then, each image was transformed into a 768-diamond vector and

the following matrices were obtained: a 1000-by-768 matrix for training set and a 500-by-768 matrix for test set. The use of PCA was not deprecated by the team. Instead, it was used to decrease diamonds of obtained matrices. The process is shown in the chart below:

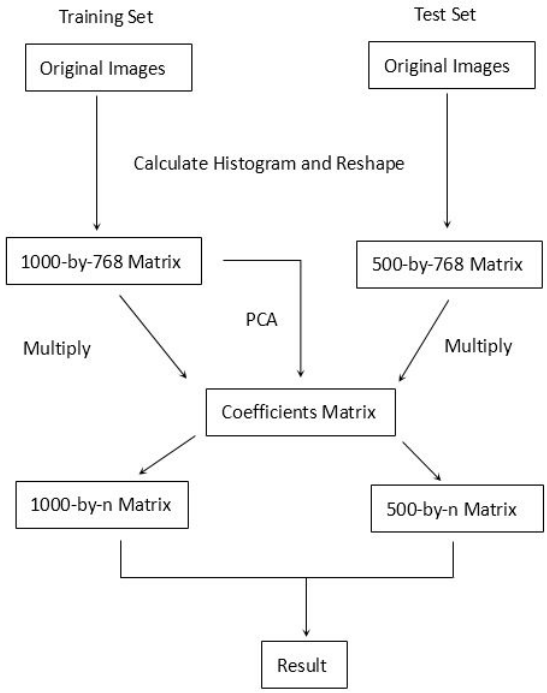

Figure 4.2: RGB Histogram Comparison

Finally, n was set to n=13 (n decided the percentage of kept features). This produced the a satisfying result shown in the table below:

Computation Time	Error of Manmade	Error of Natural	Overall Accuracy
0.095s	19.6%	40.0%	70.2%

Table 4.2: Result of RGB Histogram Comparison

Edge Comparison

Due to calculated result not being satisfactory enough, the decision was made to select another feature for classification. After observing images from manmade class and natural class, the team found out that there were more obvious straight lines in the images from manmade class than in natural class. This lead to extracting edges and transforming each image into a binary image and comparing two binary images to judge if they were from the same class. The method was similar as before. The process is shown in the flowchart below:

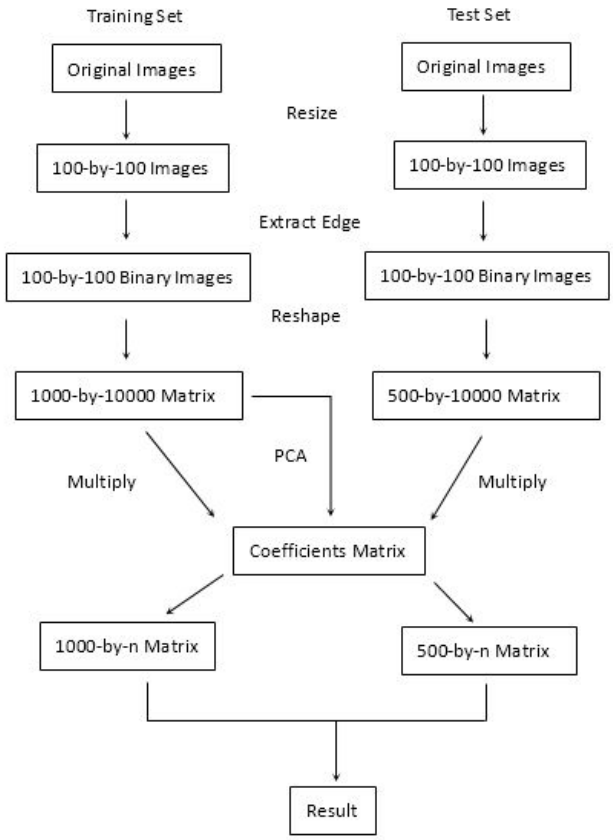

Figure 4.3: Edge Comparison

By trying this out, the team found out that except the value of n and the value of K in KNN classifier, the value of thresholding has effect on classification accuracy as well. The decision was made to extract edges of an image with different thresholding value. This produced the best result, which is shown in the table below:

Computation Time	Error of Manmade	Error of Natural	Overall Accuracy
0.142 s	11.2%	36.0%	76.6%

Table 4.3: Result of Edge Comparison

Competition Images Classification

The team tested competition images on three kinds of classification that were implemented. The best result was produced by using for Edge Comparison described above. The classification accuracy is shown in the table below:

Computation Time	Error of Manmade	Error of Natural	Overall Accuracy
0.115 s	32.8%	16.0%	75.6%

Table 4.4: Result of Edge Comparison on Competition Images

Conclusion
After measuring performance of all the classification methods, the team received interesting results. After classification method comparison it was possible to see how Edge Comparison was more effective and accurate than others. This happened because edges in images belonging to different classes were obviously different from each other.

Moreover, by using Hough transform to detect how many straight lines there are in the images, it would be possible to improve the classification even further. There are other possible ways to improve it. For example, by looking for more features (such as trees, skies and clouds) when distinguishing images from these two different classes. These are more common in the natural images than in the manmade images. But using this method would potentially increase the complexity of both training process and classification process.

Besides, is it also possible to employ a Neural Network because it does not need the artificial feature, as it learns the abstract feature from images of the training set. It is also possible to decrease the diamond of images as well by Neural Network. In addition, It is also possible to use some other classifiers for the classification process, such as SVM and SoftMax.

5. Constructing the mosaic

5.1 Basic algorithm design

The mosaic contraction algorithm is based on RGB color comparing. The difference between two images can be evaluated as Euclidean distance among RGB channels. This is done by assuming that the target block image is *tar* while a composite image candidate (regarded as 'tile' in composite image) is *tile.* Then, Each RGB channel can be represented as tar tar_R, tar_G, tar_B, $tile_R$, $tile_G$ and $tile_B$. Thus the Euclidean distance is shown in the following equation:

$$\mathrm{D} = \sqrt{\sum_{i}^{n} \left(\left(tar_R^{(i)} - tile_R^{(i)} \right)^2 + \left(tar_G^{(i)} - tile_G^{(i)} \right)^2 + \left(tar_B^{(i)} - tile_B^{(i)} \right)^2 \right)}$$

The smaller the D is, the closer color of two images would be. In order to find a better composite image, an image which has the smallest D with target image among all the selected composite images (also can be regarded as 'image pool') should be taken.

5.2 Shortcomings

Although technology progress comes in hand and nowadays almost everybody has an access to high computational powers, attempting to calculate every pixel is still exhausting and is a huge burden to the computer. This lead to increased calculation time. It is only possible to imagine how longer the process would take 10 years ago from today.

5.3 Developed solution

The team found the way of solving the problem by dividing each tile again and splitting it into smaller cells. It lead into calculating mean values of each channel of each block. The same process was repeated on target block image (dividing into the same size and amount of cells). Finally, by using these 'enlarged pixels', Euclidean distance of each channel between *tile* and *tar* was calculated. The calculation was much more efficient, comparing to the previous original method.

5.4 Implementation

The implementation of created solution was done in the following steps:

1. Comparison of each block with 'image pool' using Euclidean distance among RGB channels.
    1.1. Generating color 'fingerprint' of each image from 'image pool': *colorFeat.m* did the job by dividing input image into user-specified number of smaller cells using *mat2cell()* and by using *mean2()* to calculate mean value of each RGB channel of each cell, followed by output of the generated mean color 'fingerprint'. Applying every image from 'image pool' to *colorFeat.m* to get a cell array *'colorCell'* that would contain every 'fingerprint'
    1.2. Applying target block images to *colorFeat.m* as well and getting return value to 'comp1', getting the difference between 'comp1' and each element from 'colorCell':

```
diff=comp1-colorCell{m,1};
```

    Then getting Euclidean distance using *'diff'*, storing results and indexing into an array *'c'*.
2. Taking 20 images that have the minimum distance from 'image pool' and randomly selecting one of them for each block.
    2.1. Sorting the array using *sortrows()*, that would sort base on first column where the Euclidean distances are stored and index numbers follow the sorting movement of Euclidean distances. Then taking 20 smallest values into *'minSorted'*.

```
sorted=sortrows(c);%sort the results
minSorted=sorted(1:20,:);%get 20 smallest values
```

2.2. Random picking one among ‘minSorted’ as chosen one:

```
num=randi(20);
which2get=minSorted(num,:);%random pick one
get=temp{which2get(2),1};%get it from ‘image pool’
get=imresize(get,[tilesize tilesize]);
```

3. Converting the chosen image into HSV color space and adjusting the Value and Saturation to target block image’s Value and Saturation (taking ratios of the means of the chosen image’s Value and Saturation over the means of target block’s Value and Saturation, then setting chosen image’s Value and Saturation matrix to fit brightness in attempt to enhance the performance of the final output).

3.1. Converting both chosen image and target block image to HSV space:

```
get=rgb2hsv(get);
origin=chunk{i,j};
origin=rgb2hsv(origin);
```

3.2. Getting mean value in Value space and Saturation space and applying ratio adjustment to the chosen image:

```
ratio=mean2(origin(:,:,3))/mean2(get(:,:,3));
get(:,:,3)=get(:,:,3)*ratio;
ratio2=mean2(origin(:,:,2))/mean2(get(:,:,2));
get(:,:,2)=get(:,:,2)*ratio2;
```

3.3. Converting the chosen image back to RGB color space:

```
get=hsv2rgb(get);
 output{i,j}=get;
```

**i, j locate the position of target block which is processed.*

4. Output of the final image.
Using *imwrite()* to output the final image.

5.5 Testing

Below it is possible to view the original target image:

Figure 5.5: Original image

Below are the first examples of the generated output of the algorithm when it was still under development. Some of the images are the output that was generated using alternative methods:

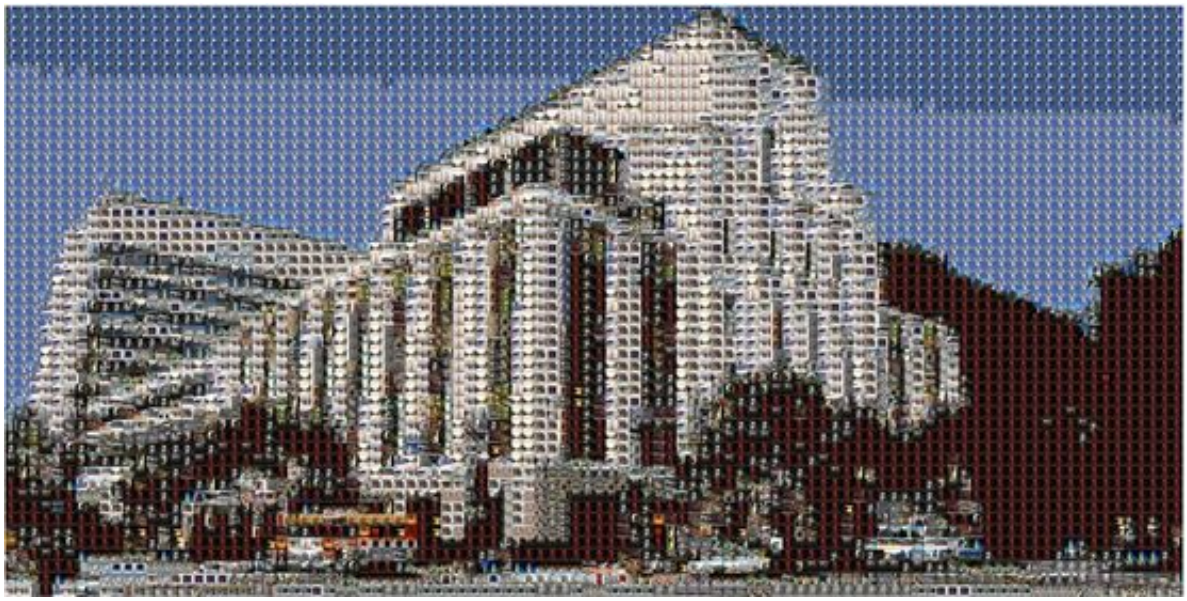

Figure 5.5.1: Using just the closest images as tiles, although the objects like the bus on the bottom left and two cars on the bottom right can be seen, but too reduplicative and roughly.

Figure 5.5.2: Adding a random function based on Image 5.5.1 that would randomly choose one from the 10 closest images as tiles, hence the output will be different for each run which also solve the problem from Image 5.5.1. However, the result appeared a little 'plain' or 'pale'

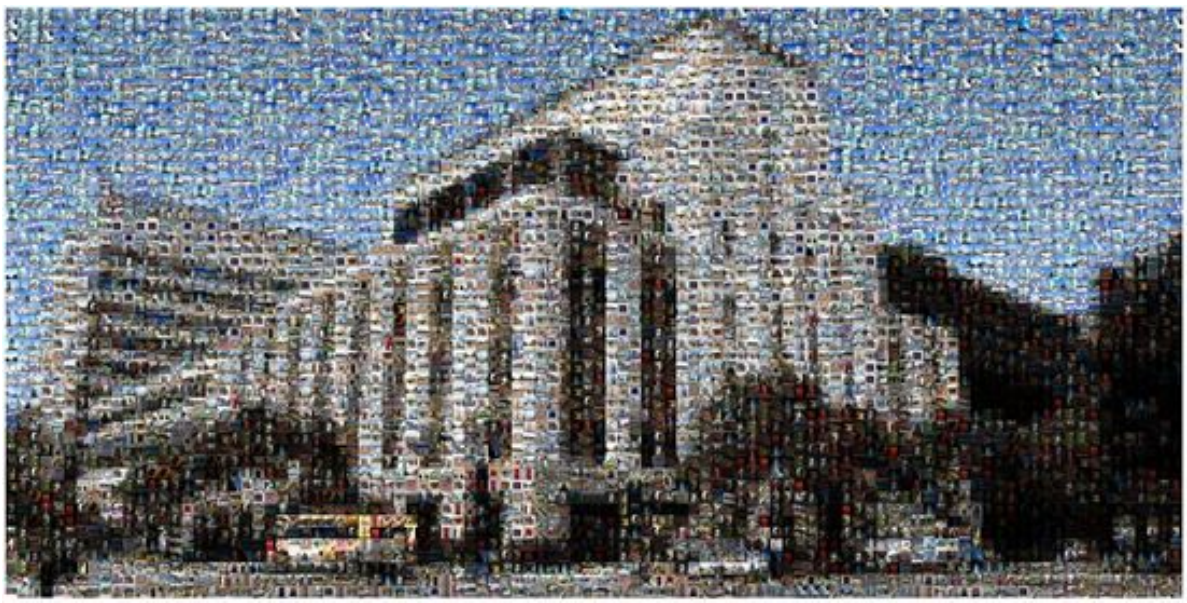

*Figure 5.5.3.1: Adding brightness adjustment, the result looks more vibrant and more details can be seen in comparison to the previous output. The output size is 2000*1000 pixels, tile size is 20*20 pixels and the number of smaller cell (sub tile) for each tile is 2*2, which means that 4 smaller areas in each tile (target block) are taken into calculation.*

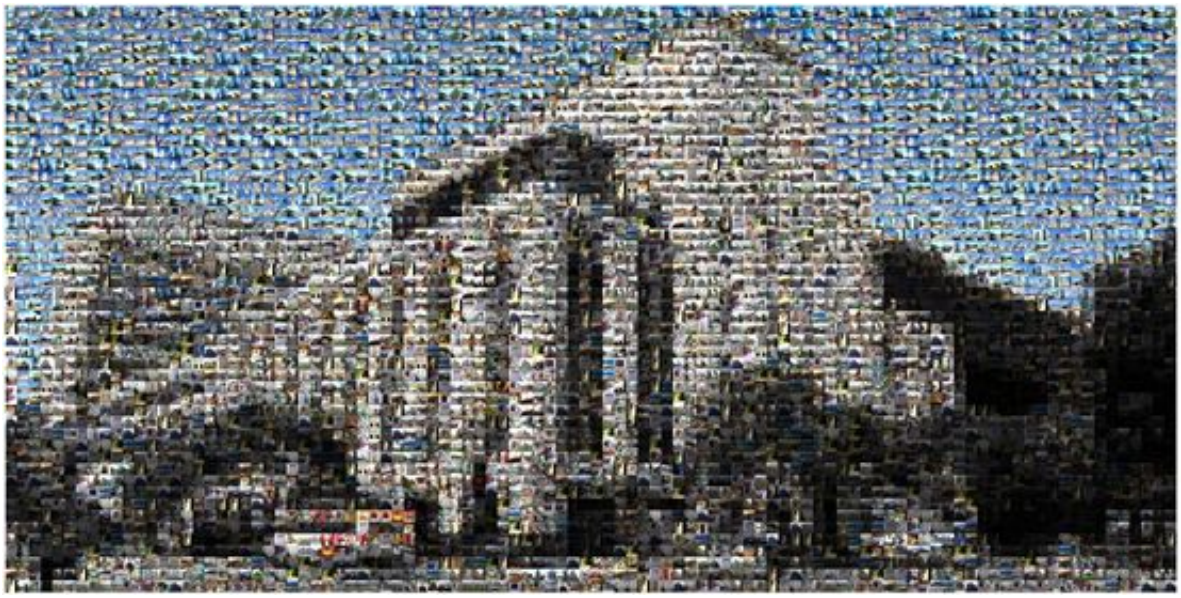

*Figure 5.5.3.2: However, if smaller cell is set to 1*1 (which means operating no division on each tile) of each tile, the generated output looks more noisy than the previous output.*

In attempt to address the points described above, Edge matching using Hausdorff distance was taken into consideration.

Hausdorff distance between two curves A and B can be described as the following equations:

$$h(A,B) = \max_{a \in A} \{ \min_{b \in B} \{ d(a,b) \} \}$$

$$H(A,B) = \max \{ h(A,B), h(B,A) \}$$

Where d(a, b) means the distance between point a and b. And a is the point on curve A, b is the point on curve B.

The method was used properly after choosing 10 closest images, calculating Hausdorff distances between each image with target block image. However, it required too much computation and the effect is was not too significant if edge points were simplified. The following image shows the result of using this method with simplification (when each tile is converted into an 8*8 edge matrix using function *edge()* and then compared with target block image's 8*8 edge matrix using Hausdorff distance, followed by edge points being part-selected so that for every edge matrix there are no more than 15 points). Although it has already been simplified, it took almost twice the time than the original algorithm. And the difference was not substantial:

Figure 5.5.4: Output generated from edge matching using Hausdorff distance

Figure 5.5.5: Output generated using Original Algorithm

In this scenario, the number of tiles was decreased deliberately because of total-computation-time consideration

Figure 5.5.6: Adding brightness adjustment only still produces less colorful result. To adress this, saturation adjustment was added and the output looks even more vibrant and colorful than Image 5.5.3.1

5.6 Future potential expansion

The future tests were done by testing other manmade or natural images. Below it is possible to see the output from testing another man-made target image:

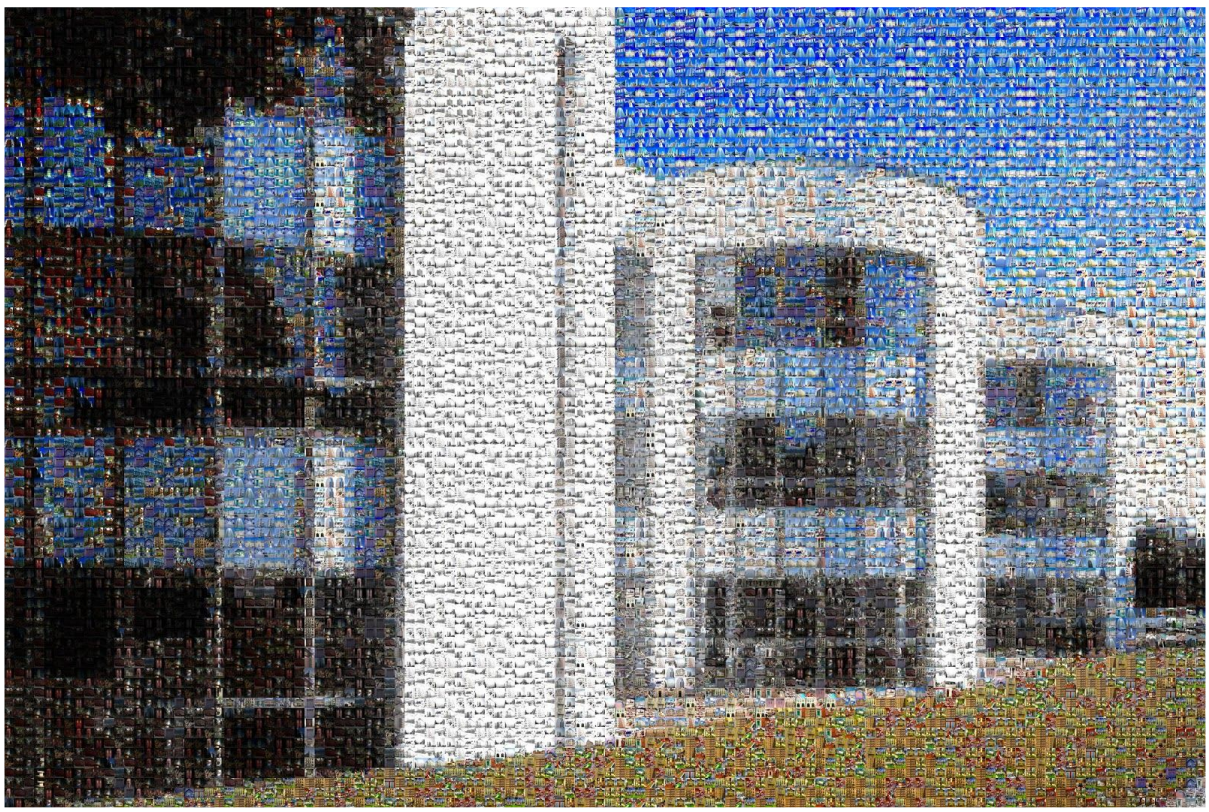

Figure 5.6: Generated image

Figure 5.7: Original image

Other manmade images were also considered and tested:

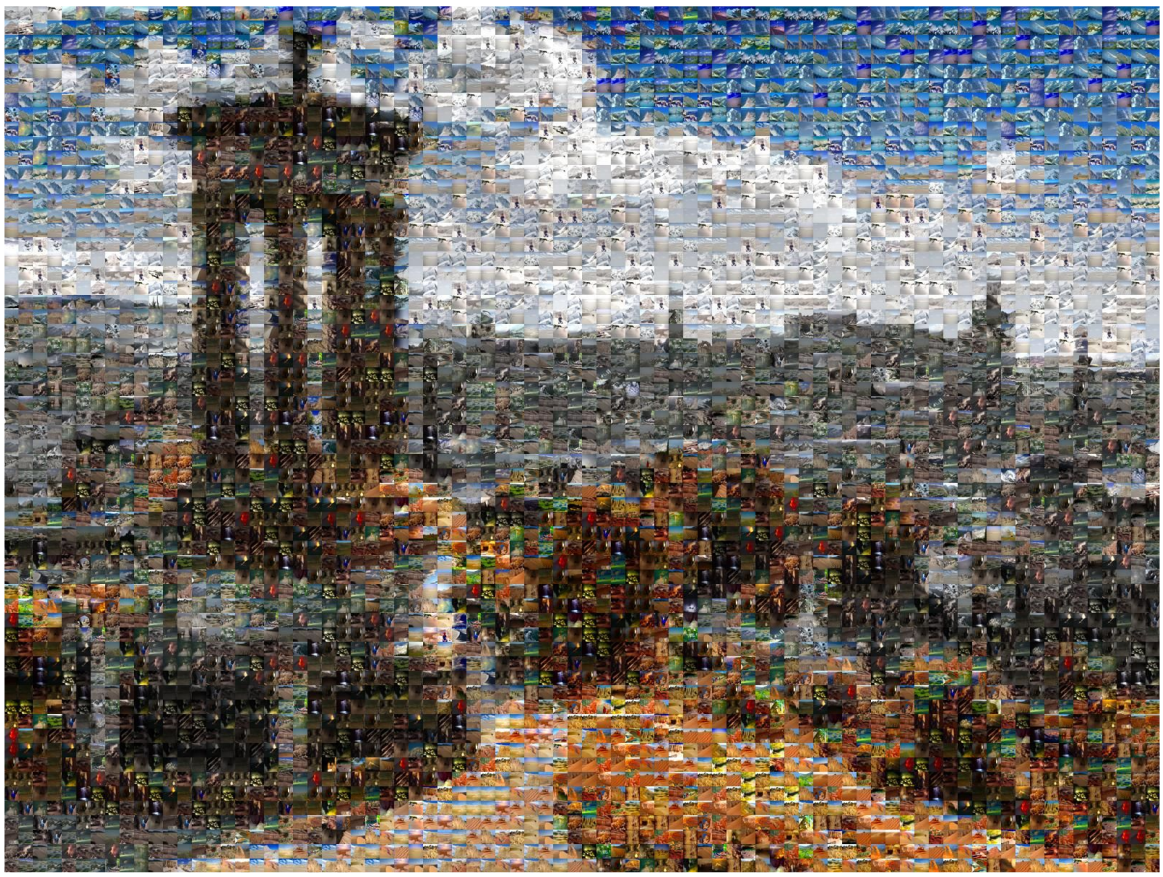

Figure 5.8: Generated image

Figure 5.9: Original image

Points noted in the result of testing:
Grass seemed to be turning into autumn colors, however the sky looked fine. By looking closer into the output of the natural image, most of the grass parts were replaced by desert scenes. The only possible explanation would be that fact that the desert RGB color is more close to the light green from original image, than other images containing darker green from ‘image pool’.

One quick alternative improvement would be blending the output with original target image. The following picture is the result of blending 80% of the output and 20% of the original:

Figure 5.10: the grass looks more green than the previous output with the help of original target image as one of the components. By increasing the proportion of the original target image, it would appear more green and closer to the original target image.

However, the team believed that the method which uses blending with original target image is called ‘fake mosaic’ which seemed to be ‘unfair’. The team decided that for the means of this assignment that would not be a perfect solution to this problem.

Other possible solutions:
Use of color recognition could be a good alternative. A dominant color of each chunk can be assigned. When choosing the closest image as tile, if the dominant color is different, then it should be taken out of consideration.

Solution 1:
Deciding dominant color can be done by counting number of big values of each channel (where value which is greater than 200, for instance). Then, it would be required to assign each channel with 1 or 0 whether the number is greater than the threshold or not. For instance, Taking 1 in red channel means this tile’s dominant color has red component. So in addition, there would be eight possible combinations. Which means eight different dominant colors can be assigned.

This solution has been tried out and the results were not satisfying. If the big value and the threshold were set too high, the output looked pretty much the same as the original output. If

they were set too low, the output would not even provide the same level of efficiency as the original output:

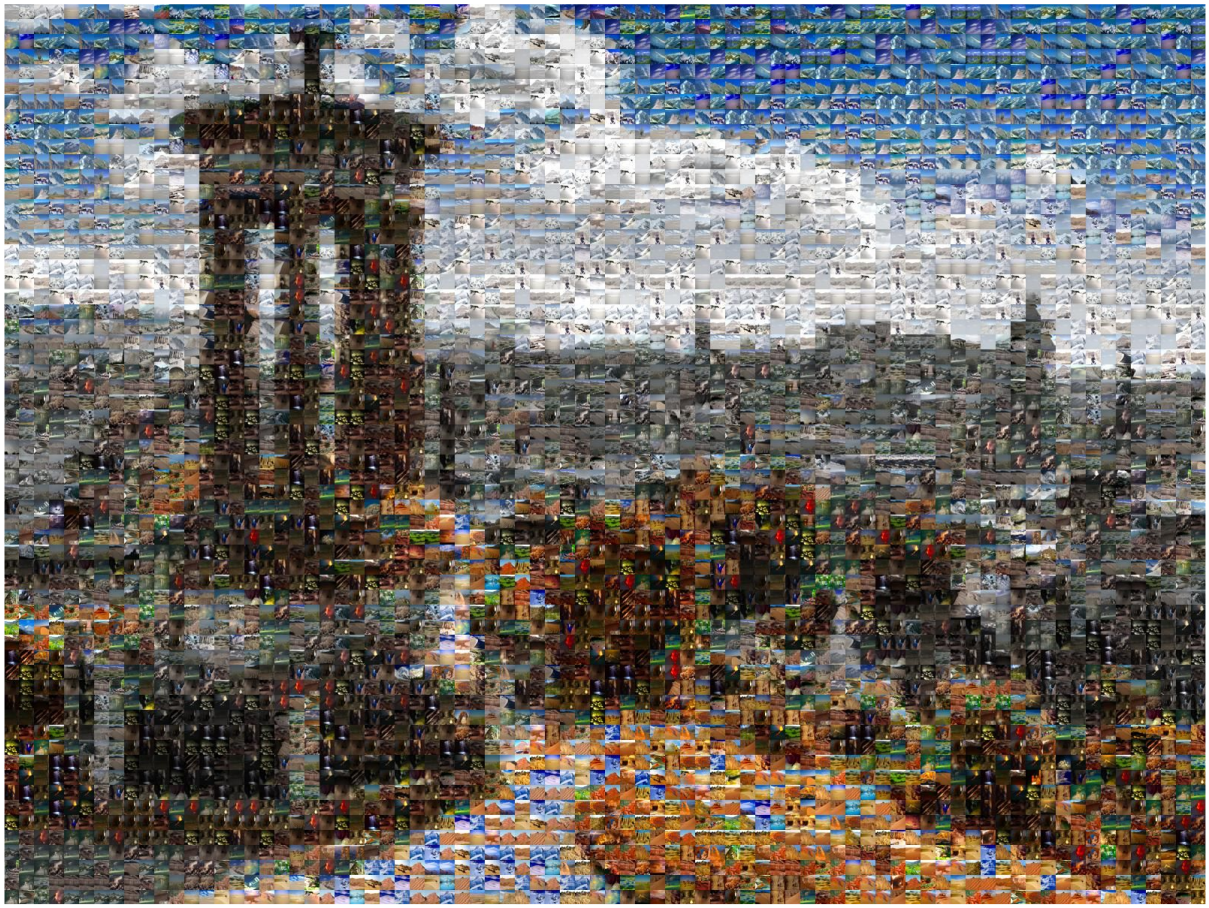

Figure 5.11: The output is clearly worse than the the original output. At the bottom of this image, some of the ground was replaced by images of skies.

Solution 2:
The following solution was just a product of ‘brainstorm’ of one of the team members. It was not yet been implemented and fully tested. However it has a very large possibility to perform in the right way, according to what was learned from the lectures so far.

The idea was to still use the color recognition, driven by the neural network. The procedure would be to take different color dominant images selected by human as training set and train the program to justify whether the image is red-dominant, green-dominant or blue-dominant. And set degrees of each dominants (indicating how much it dominates). The chosen tile is then selected within a range of degrees inside one of the dominants.

6. User interface

During one of the group meetings it was agreed that it will be essential to have some sort of the user interface to avoid use of command line multiple times. It was agreed that having GUI as such will improve overall user experience of the final solution and will make it usable for people who are not familiar with Matlab. As the result, the developed solution does not

require the use of the command line from the user. Moreover, the solution can be deployed as an executable file that makes it possible to run the application without launching Matlab. The user interface offered source folder selection, target image selection and output folder selection. This was implemented in such way that only image file can be selected as a target (custom file types, multi selection off), only folders can be selected as source/target paths. The additional parameters included: size of output image (width and height), tile size, subtile size. It was decided to trigger image generation when user presses “Generate” button. For visibility, the program outputs displays “Image generated” when the process is finished. It was agreed to disable user input into text fields after the image generation is executed to prevent errors. For better user experience, clear error messages were displayed if any of the required fields were left empty.

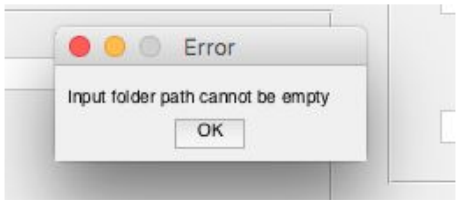

Figure 6.1: User input evaluation

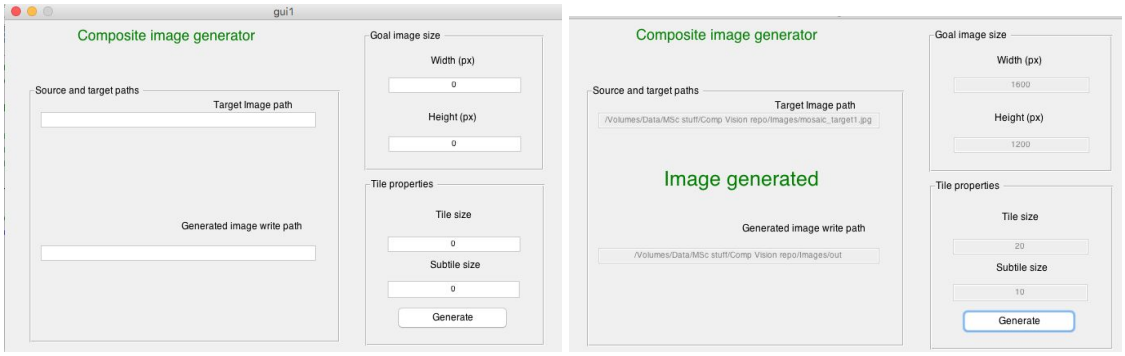

Figure 6.2,6.3: User interface

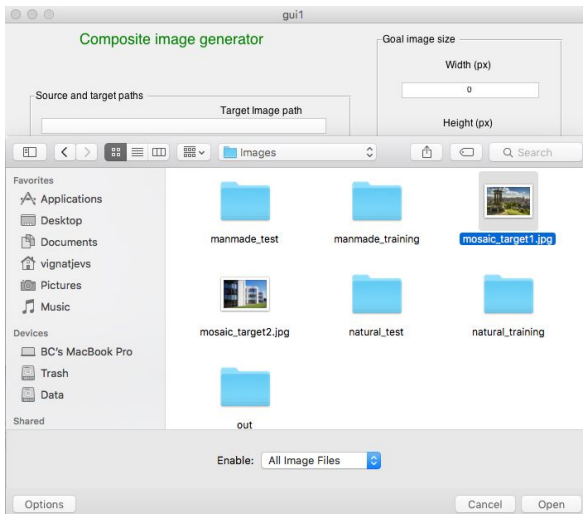

Figure 6.4: Using custom file types

7. Meetings

In overall, the team meetings happened one to two times per week during the project time. During the first meetings, the tasks were clearly divided between team members to allow everybody to focus on their own part. The team discuss different option for software, project management and agreed on necessary tools which will be used for the project. It was also agreed to set up a private group on social media to allow out of hours communication. This group was used mostly for updating each other, assigning meetings, leaving links to related books/papers or asking questions to each other. At the certain stage it was decided to meet once per week and assign more meetings as per requirement. The project was “semi-agile” [8], during every meeting each of a team was spending 2-3 minutes to update the rest with their amendments to the project.

8. Using git for teamwork

The team decided to use a version control system to avoid version conflicts and allow working on different file revisions at the same time. Nowadays, use of VCS is considered as essential in software development projects. On the first meeting it was decided to use GitHub because it offered different advantages, such as branching, access to its full version for students. Apart from that, GitHub offered transparent user interface which was beneficial both for the people who are not very familiar with command line and for those who were. Issues section was another great option that served as additional tool for reporting bugs. As the result of these decision, the project all the code was maintained using Git. [9]

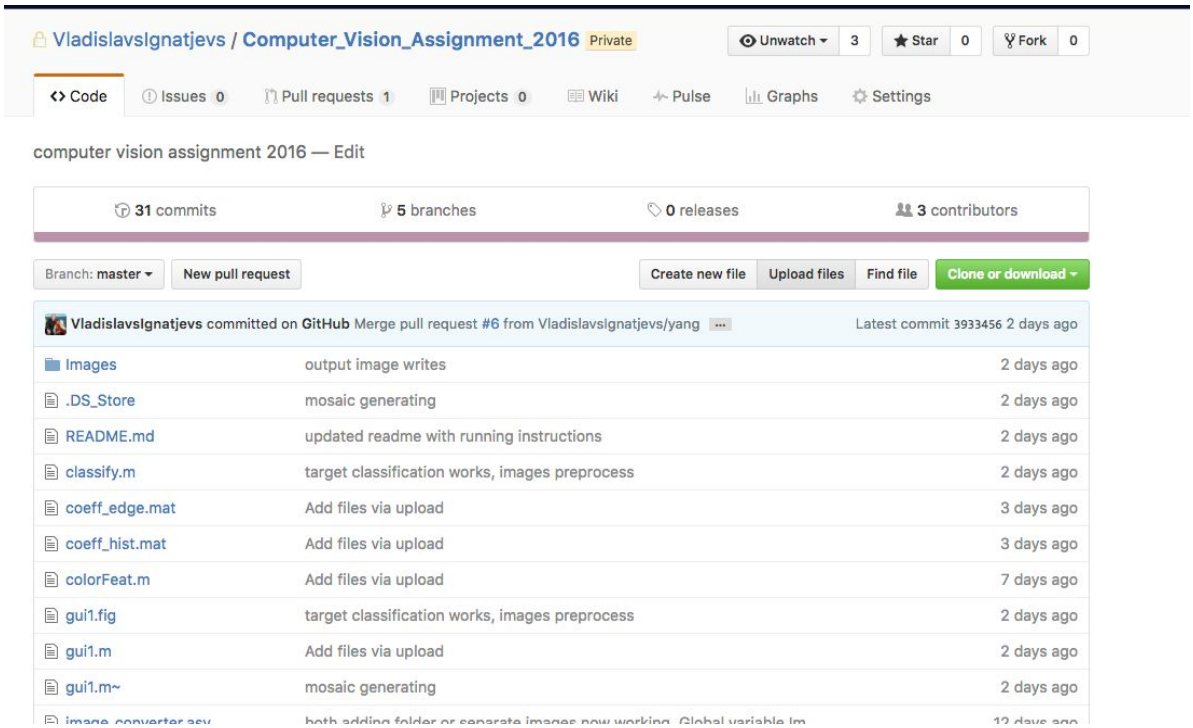

Figure 6: Use of GitHub to improve teamwork

9. Coding

Making sure that the code is readable for other team members is very important in every every software development team. Every member of the team followed the best coding practices [10] while working on the project. The code was always well maintained (refactored), commented, indented to ensure maximum readability. Continuous integration and regular testing was ensured along with some sort of a pair programming . This reduced the amount of time spent in the end of the project, when all the parts were put together.

10. Conclusion

Part A. Photo mosaic

As discussed in 5.6, the RGB comparison has one disadvantage of using euclidean distance which is the fact that it would discard, no matter if whether the difference is positive or negative, so closest distance would be chosen even if it means a different color choice (as per grass turning into autumn colors). So a color detector would be needed in order to make the final result looking closer to the original target image.

Part B. Classification

As aforementioned, exploring good features is the key to achieve a good classification accuracy. From the result we can see that edge is more effective than other two features. More features are helpful to promote the accuracy but too many features are potential to result in overfitting. Besides, there are some other ways to extract features from images such as Neural Network. It is worthwhile to have a try in the future work.

11. References

[1] Lee, H.-Y. (2014) *Generation of Photo-Mosaic Images through Block Matching and Color Adjustment. International Journal of Computer, Electrical, Automation, Control and Information Engineering, 8(3), p. 457-460.* Available at: http://waset.org/publications/9997759/generation-of-photo-mosaic-images-through-block-matching-and-color-adjustment (Accessed: 24 November 2016).

[2] Shah, J., Gala, J., Parmar, K., Shah, M. and Kambli, M. (2014) *Range Based Search Algorithm For Photomosaic Generation*. Available at: http://www.ijarcce.com/upload/2014/february/IJARCCE4D___a_krunal_Range.pdf (Accessed: 24 November 2016).

[3] Veltkamp, R.C. (2001) *Shape Matching: Similarity Measures and Algorithms*. Available at: http://www.staff.science.uu.nl/~kreve101/asci/smi2001.pdf (Accessed: 24 November 2016).

[4] Blasi, G.D. and Petralia, M. (2005) *Fast Photomosaic*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.3800&rep=rep1&type=pdf (Accessed: 24 November 2016).

[5] Tran, N. (1999) *GENERATING PHOTOMOSAICS: AN EMPIRICAL STUDY*. Available at: http://www-cs.ccny.cuny.edu/~wolberg/capstone/photomosaics/EmpiricalStudySAC99.pdf (Accessed: 24 November 2016).

[6] Finkelstein, A. and Range, M. (1998) *Image Mosaics*. Available at: http://kucg.korea.ac.kr/seminar/2006/tr/PA-06-30.pdf (Accessed: 24 November 2016).

[7] Blasi, G.D., Gallo, G. and Petralia, M.P. (2006) *Smart Ideas for Photomosaic Rendering*. Available at: http://gianpierodiblasi.xoom.it/pubblicazioni/eg_it2006_3.pdf (Accessed: 24 November 2016).

[8] SJSU (no date) *Semi-Agile Model-Driven Development (SAMDD)*. Available at: http://www.cs.sjsu.edu/faculty/pearce/cs160/SAMDD.htm (Accessed: 24 November 2016).

[9] Ignatjevs, V., Xu, Y. and Li, W. (2016) *Computer Vision Assignment 2016 repository on GitHub*. Available at: https://github.com/VladislavsIgnatjevs/Computer_Vision_Assignment_2016 (Accessed: 24 November 2016).

[10] VersionOne (2016) *Agile programming best practices*. Available at: https://www.versionone.com/agile-101/agile-software-programming-best-practices/ (Accessed: 24 November 2016).