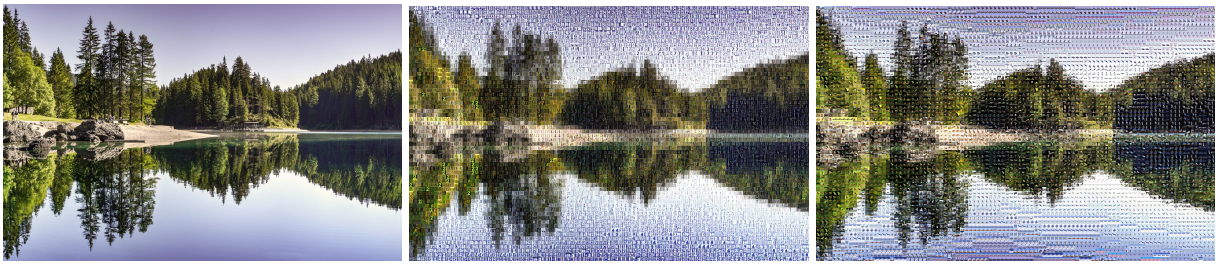
SCHWEIZER JUGEND FORSCHT
LA SCIENCE APPELLE LES JEUNES
SCIENZA E GIOVENTÙ
SCIENZA E GIUVENTETGNA

PROJECT REPORT
STUDY WEEK "FASCINATING INFORMATICS"

Autonomous Image Mosaicing



Julia Gschwind[1], Pascal Burkhard[2] & Dario Ackermann[3]

[1] Kantonsschule Solothurn, Solothurn, Schweiz
[2] Gymnasium Neufeld, Bern, Schweiz
[3] Kantonsschule Sargans, Sargans, Schweiz

Supervised by: Adrian Wälchli, Computer Vision Group, University of Bern
Date: 14 September 2018

Abstract
We develop an intelligent algorithm to convert a given input image to a mosaic version of it. The code covers several aspects, an analysis of the average patch color using vectors as well as specific training using neural networks.

1. Introduction/Question
Image Mosaicing is nowadays mainly known from ancient times. However, to form a visual appealing mosaic manually, it often implies a great deal of effort and is therefore a costly and laborious task. Our research focuses on the following points:

- Which different approaches are feasible to dynamically create mosaics?
- What's the quality of the output? (e.g. similarity, edges, shapes…)
- How long does this process take, and, which of the approaches is the most efficient one?
- Is it possible to utilize neural networks to generate a significantly better image compared to other approaches?

2. Materials & Methods

For our project we used the following materials:

- Python as a programming language with numpy as matrix library
- PyCharm Professional Edition as a programming environment
- Git as our version control system
- Cifar 10 and svhn dataset

The challenge of the week was to find the most efficient way to generate a image mosaic. Therefore we had to try different methods to find the fastest option.

First approach: Nearest neighbor search combined with a k-dimensional tree based on average color feature

The first approach consists of calculating the average color feature of each patch in the dataset as shown in Fig. 1.

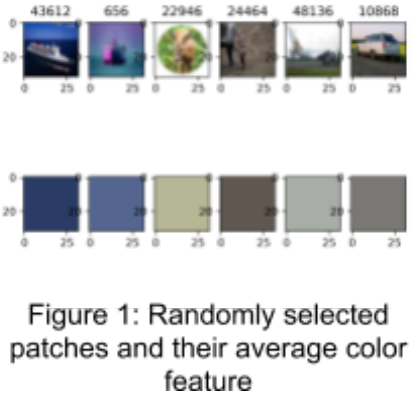


Figure 1: Randomly selected patches and their average color feature

Afterwards, the gathered data is saved into a k-dimensional tree (k-d-tree). A k-d-tree helps finding points in a multidimensional space more efficiently. Every patch of 32x32 pixels represents one out of 256^3 possible colors after calculating the average feature.

All this data could also be stored in a normal, multidimensional array. However, searching data points in it would be highly inefficient and is not feasible with a dataset of 50'000 patches.

As soon as the k-d-tree is built and all the average color features are stored, the input image, specified by the user is split up into tiles of 32x32 pixels. The same procedure applies: the average color feature of each tile is calculated and then searched in the k-dimensional tree with a so-called "Nearest neighbor search" - the algorithm tries to find the most similar patch from the dataset.

Our nearest neighbor search works by selecting the patch with the shortest distance to the tile of the image. In order to calculate the distance, we used the following formula, x and y being the average color of the original patch and the replacement:

$$d(x,y) = \|\bar{x} - \bar{y}\| = \left\| \frac{1}{N} \sum_{i=1}^{N} x_i - y_i \right\|$$

After having selected matching patches for the tiles, the image is reassembled and saved to the local disk. This approach generates very pleasing results. After some optimisation done by Pascal, this approach turned out to be quite efficient in time.

Figure 3. Original and a mosaiced image.
The tiles in this image are selected randomly out of the five nearest neighbors.

Second approach: Clustering

Before optimising the above approach, the clustering approach was intended to be faster at the cost of the output image. While training the model, the cluster algorithm tries to classify and group similar patches (and their vectors) together. This method is intended to speed up the imaging process, because no more nearest neighbor searches are necessary.

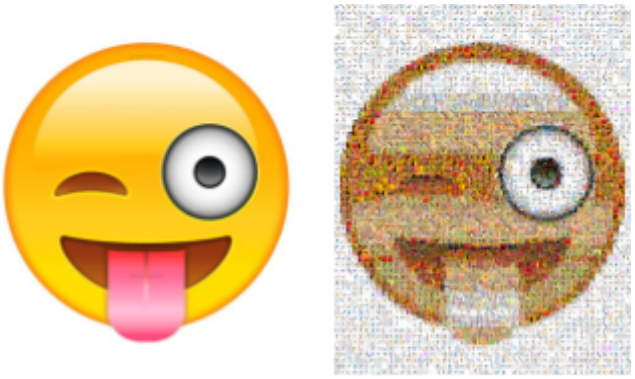


Figure 4. Mosaiced image using 800 clusters and a resolution of 62 patches in height

3. Extensions

Blending

To make it clearer what the picture displays we implemented the blending method. One mixes the original input with the mosaic version of it considering the adjustable intensity (a) of the first picture by using the following formula.

$$a \cdot image_1 + (1 - a) \cdot image_2$$

Video

We applied our first approach on a sequence of frames; a video. To not fill our Random-Access-Memory (RAM) with large video files, we only read one frame at once, processed it to a mosaic and saved it in an output file. Then we took the next frame and so on. Although we can find more than 100 fitting patches per second in a mosaic, it takes several minutes up to hours to create a processed video of a few seconds duration.

Web Interface

The aim is that a user is able to visit a webpage and upload a image which is then converted to a mosaic and provided to download.
Link to our webpage: www.cvg.unibe.ch/mosaic

4. Results

During the week we created a large number of image mosaics using the different methods and compared them. It was clear that the nearest neighbor method produced the best results as the colors were closest to the ones in the input. When we were testing the clustering method we expected a decrease in quality but a significant improvement in efficiency. However, the quality was low as expected but it was even less efficient (about 5-10%) than the nearest neighbor method. Therefore it was clear that nearest neighbor was our preferred method overall.

5. Discussion

We have seen that there are different approaches to image mosaicing and that some produce a qualitative better output than others and are more efficient in time. Clustering turned out to be inefficient. We tried to use neural networks, however, these results were not completely satisfying. More customisation would be needed in order to improve the quality of the output image.
While working on our project we learned a lot about how a image is built up. We gained better programming skills and once we were familiar with the project it was easy to find more methods. Furthermore, we were introduced to Kmeans.
As there were big differences in experience between the team members our tutor gave us tasks which correspond to our knowledge so nobody was over- or underwhelmed and everyone could learn as much as possible.

6. Acknowledgements

First of all we would like to thank our main sponsor of the week, the Hasler Stiftung, as well as all the other sponsors as this project would not have been possible without their support. Also we would like to thank our supervisor Adrian Wälchli who helped us with problems and patiently guided us through the project, the University of Bern and the organiser Dario Moser.