UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

**Academic Year 2020/2021**

**Master Degree in ICT for Internet and Multimedia**

**02 · 2021**

# 3D AUGMENTED REALITY

# PROJECT REPORT - B.2

**Casagrande Nicola - De Toni Alberto**
**1242363 - 1236657**

**Local feature compression using autoencoders - SfM tests**
Design a compression strategy for local SURF descriptors using autoencoders. Training data can be generated using the images of dataset Portello and Castle. Testing must be done on dataset FountainP-11 and Tiso (available at https://github.com/openMVG/SfM_quality_evaluation/tree/master/Benchmarking_Camera_Calibration_2008 and http://www.dei.unipd.it/~sim1mil/materiale/3Drecon/). Software must be implemented in MATLAB, Keras or Pytorch.
**Testing on 3D reconstruction using SfM:** The reconstructed descriptors (only for the test set) are used to perform a SfM reconstruction using COLMAP (using the two test dataset).
Programming languages: MATLAB/Python/C++.

# 1 Introduction

## 1.1 Descriptors in general

In computer vision, visual descriptors or image descriptors are descriptions of the visual features of the contents in images, videos, or algorithms or applications that produce such descriptions. They describe elementary characteristics such as the shape, the color, the texture or the motion, among others[1]. A feature detector (*extractor*) is an algorithm taking an image as input and outputting a set of regions ("local features" = "Interest Points" = "Keypoints" = "Feature Points"). A descriptor is computed on an image region defined by a detector. The descriptor is a representation of the image function (colour, shape, ...) in the region (typically an array). Two key operations are related to feature extraction:

- Feature detection: extract the features of interest
- Feature description: associate a descriptor to each feature in order to distinguish from the others

There are many pre-defined descriptors available to the users. Some of them are reported in fig.1.

## 1.2 Speeded Up Robust Features (SURF)

In computer vision, Speeded Up Robust Features (SURF) is a local feature detector and descriptor. It can be used for tasks such as object recognition, image registration, classification, or 3D reconstruction. It is partly inspired by the Scale-Invariant Feature Transform (SIFT) descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT[2]. The SURF algorithm

[1] https://en.wikipedia.org/wiki/Visual_descriptor, 01/02/21
[2] https://https://https://en.wikipedia.org/wiki/Speeded_up_robust_features, 01/02/21

| Desc. | Binary | Invariant | Size | Complexity | Robustness | Patents |
|---|---|---|---|---|---|---|
| SIFT | no | Scale, rotation, mild comp. | 128 float= 512 B (128B in some impl) | High | High | Yes |
| SURF | no | Scale, rotation, mild comp. | 64 float = 256 B or 128 float = 512 B (conv. to bytes sometimes) | Medium/High | High | No |
| BRIEF | yes | Scale, rotation | 256 or 128 bits | low | Low | no |
| BRISK | yes | Scale, rotation | 512 bits | low | low | no |
| FREAK | yes | Scale, rotation | 512 bits | very low | low | no |

Figure 1: Common algorithms for feature description.

is based on three main parts:

- Detection: using a blob detector method based on the Hessian matrix to find feature points
- Description: the goal of a descriptor is to provide a unique and robust description of an image feature
- Matching: comparing the descriptors obtained from different images to find the matching pairs

## 1.3 Autoencoders in general

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input[3]. There are two categories of autoencoder:

- Regularized autoencoders: they are divided into three types such as Sparse autoencoder (SAE), Denoising autoencoder (DAE) and Contractive autoencoder (CAE)
- Variational autoencoders (VAE): generative models akin to generative adversarial networks (GAN)

[3]https://https://en.wikipedia.org/wiki/Autoencoder, 01/02/21

A basic autoencoder as shown in fig. 2 has an internal hidden layer and it's made of two main parts which are the encoder that turns a high-dimensional input into a latent low-dimensional code, and the decoder that performs a reconstruction of the input with this latent code. It's also possible to train the autoencoder with multiple layers to exponentially reduce the computational cost of representing some functions and to exponentially decrease the amount of training data needed to learn some functions. The two steps of using an autoencoder are:

- Training
- Testing

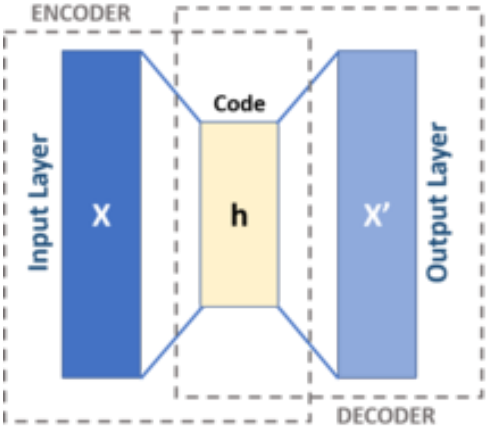



Figure 2: Scheme of a basic autoencoder.

## 1.4 Development software

MATLAB is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages[4]. We decided to use Matlab as Machine Learning tool because we felt more confident, while also it has an extensive documentation and it offers different advantages:

- Implement and test algorithms easily
- Develop the computational codes easily
- Debug easily
- Use a large database of built in algorithms

[4]https://https://https://https://en.wikipedia.org/wiki/MATLAB, 02/02/21

- Process still images and create simulation videos easily
- Symbolic computation can be easily done
- Possibility of calling external libraries
- Perform extensive data analysis and visualization
- Develop application with graphics user interface

### 1.5 Goal of the project

The objective of the project is to develop a compression strategy for local SURF descriptors using an autoencoder and perform a SfM reconstruction using COLMAP software with the obtained data. The matchings and the features must be saved in a .txt file. The autoencoder has to be trained with training data generated from a given dataset and has to be tested on another dataset. The final result should display a worse reconstruction due the compression of the data and the goal is to find a good setting for the autoencoder.

## 2 Data

The data of the experiment are divided into two groups of images, one used for the training of the autoencoder and another one for the testing. The images represent four architectural elements from different points of view. In the training group there are images of Portello and Castle (some of them are reported in fig. 3 and fig. 4), while in the group of the testing images there are images of fountain and Tiso, some of them reported in fig. 5 and fig. 6.

Figure 3: samples from Portello dataset.

Figure 4: samples from castle dataset.

Figure 5: samples from Fountain-P11 dataset.

Figure 6: samples from Tiso dataset.

# 3 Development

## 3.1 Training

The images from the training set are loaded into a single image datastore. After that, a loop for each image is designed in order to:

1. Convert the image to grayscale;
2. Detect the SURF features (using the MATLAB function “detectSURFFeatures”);
3. Extract the features (descriptors) using the function “extractFeatures”. In this case they are represented as a vector of 64 columns and $N$ rows;
4. Stack vertically the descriptors of each image in a single matrix.

The result is a big matrix containing all the descriptors for each image in column. For the next step the autoencoder has been built using the function “layerGraph”, which requires in input a vector “layers” containing the structure of the Neural Network. For this purpose, we chose the following three-layer structure:

- an input layer that receives features of dimension 64 (using “featureInputLayer(64)”);
- two fully connected layers of dimension 6 and 64 (using “fullyConnectedLayer(6)” and “fullyConnectedLayer(64)”), the second one having dimension 64 as will work as output layer;
- a regression layer that will predict the reconstructed values (using “regressionLayer”).

After some testing, we found out that for the inside layer a dimension of 4 was too strict (The reconstructed values were too compressed, i.e. too far from the input ones) while for 8 they were too close to the input ones, so we opted for the compromise of 6. Regarding the training options, we decided to use an *adam* (Adaptive Moment Estimation) optimizer and a maximum number of epochs of 2, since even after one epoch the training loss was pretty stable around 5% (fig. 7). Training the autoencoder for a longer period of time could result in a better performance.

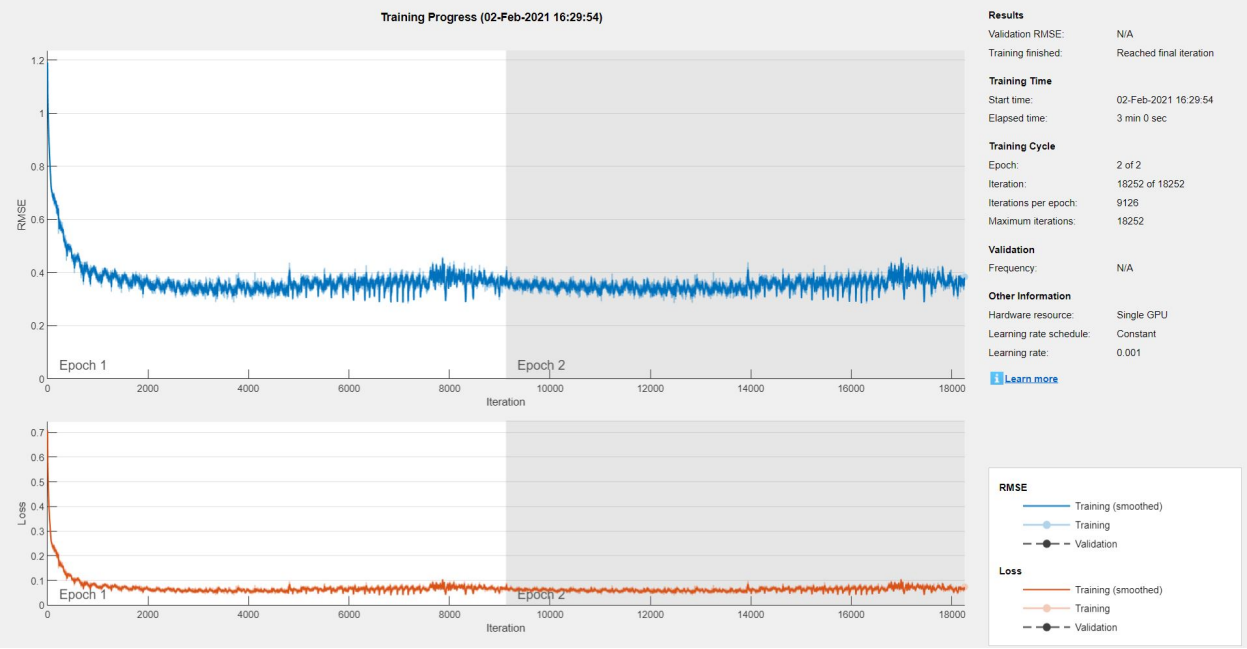



Figure 7: Training results after 2 epochs (18252 iterations).

The autoencoder is then succesfully trained (using the function “trainNetwork”) in order to be ready to be used for the testing phase. A useful trick in order to

fasten up the operation of testing and not doing everytime the training (which could require some time if the number of epochs is increased) is to save the workspace at the end of the training section and load it at the beginning of the testing one, so that everytime that section is called the autoencoder will remain trained just like it was the first time.

## 3.2 Testing

For the phase of testing the image datasets must be loaded separately (one for each architecture). To be clearer, this section simply called two functions, one for the features extraction, and one for the matchings calculations. This was done because COLMAP requires specific type of files in order to perform a reconstruction (will be better explained in sec. 4).

### 3.2.1 Function FEATURES

This function aims to reconstruct the features of each image with the autoencoder and the original ones in a .txt file for each image in the form "location_x location_y scale orientation" followed by 128 zeros since COLMAP works with SIFT features, which are of that size. The workflow of the function is the following:

- The path containing the features is cleaned from older files containing the features;
- For each image in the image datastore considered, which is given in input along with the autoencoder:
    1. The image is converted to grayscale;
    2. The features are detected and then extracted like for the phase of training, only this time they won't be stacked in a single matrix but the features **extracted** are going to be fed into the autoencoder and be returned from the function while the **detected** ones are going to be divided in Location, Scale, Orientation in order to be printed in a .txt file of the form

$$
\begin{aligned}
&N\ 128\\
&location_{x0}\ location_{y0}\ scale_0\ orientation_0\ 0\ ...(\text{x}128)\ 0\\
&location_{x1}\ location_{y1}\ scale_1\ orientation_1\ 0\ ...(\text{x}128)\ 0\\
&...\\
&location_{xN}\ location_{yN}\ scale_N\ orientation_N\ 0\ ...(\text{x}128)\ 0
\end{aligned}
$$

    where $N$ is the number of features.

### 3.2.2 Function MATCHINGS

The function, given in input the image datastore considered and the features (compressed or not) matches the features of each couple of image $i, j$. To avoid repetitions ($i, j$ and $j, i$ have the same matchings) the MATLAB function "matchFeatures" is called for every $i, j$ for $i$ going from 1 to the last feature -1 and $j$ going from $i + 1$ to the last feature. The matchings are all stacked in a single .txt file of the form:

$$
\begin{aligned}
&image0.jpg\ image1.jpg \\
&matching1_0\ matching1_1 \\
&matching2_0\ matching2_1 \\
&... \\
&matchingN_0\ matchingN_1 \\
& \\
&image0.jpg\ image2.jpg \\
&matching1_0\ matching1_2 \\
&...
\end{aligned}
$$

# 4 COLMAP

Having all the required files the performances of the compression system can be tested on a 3D-Reconstruction software, like COLMAP (https://colmap.github.io/). This software was meant for using SIFT features (see fig. 1), which are heavier and more performing in terms of quality but also less robust to image transformations (which must be accounted since the images might be taken with a mobile phone) and patented. A trick that can be used is to use SURF features (64 float) and faking a larger dimensionality (128 float) by adding zeros. This way COLMAP recognizes the right dimensionality of the files while using SURF features instead of SIFT. In order to start the reconstruction, COLMAP will need three things:

1. A set of images of the same scenario in different perspectives;
2. A .txt file for each image containing the features, in the format specified in sec. 3.2.1. The title of the file must be the name of the image followed by .txt (*e.g: image001.jpg.txt*);
3. A .txt file containing the matchings between each image in the format specified in sec. 3.2.2.

Not having these will result in an error inside COLMAP which won't start the reconstruction.

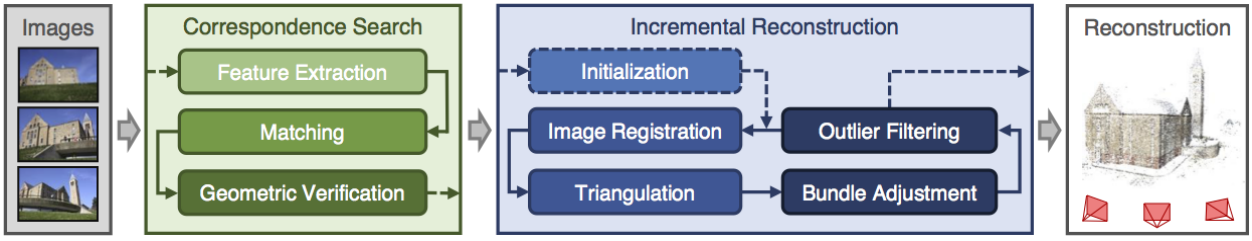
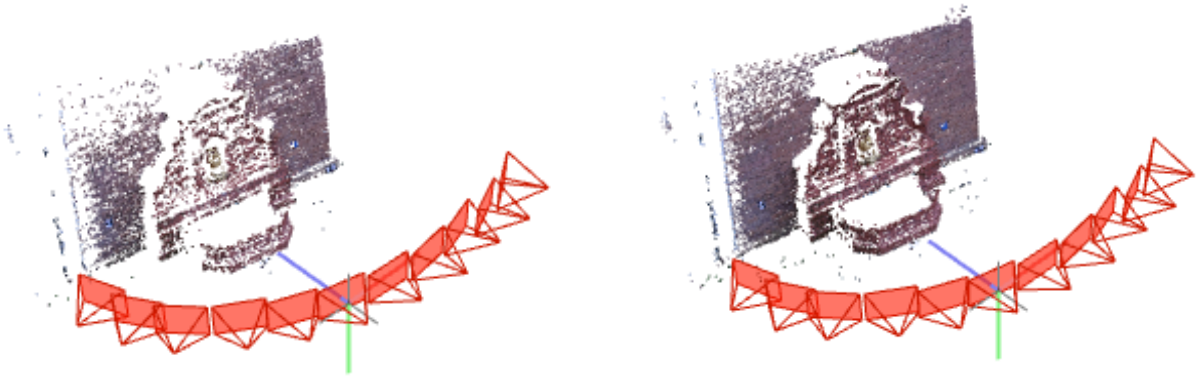



Figure 8: COLMAP's incremental SfM pipeline.

# 5 Conclusions

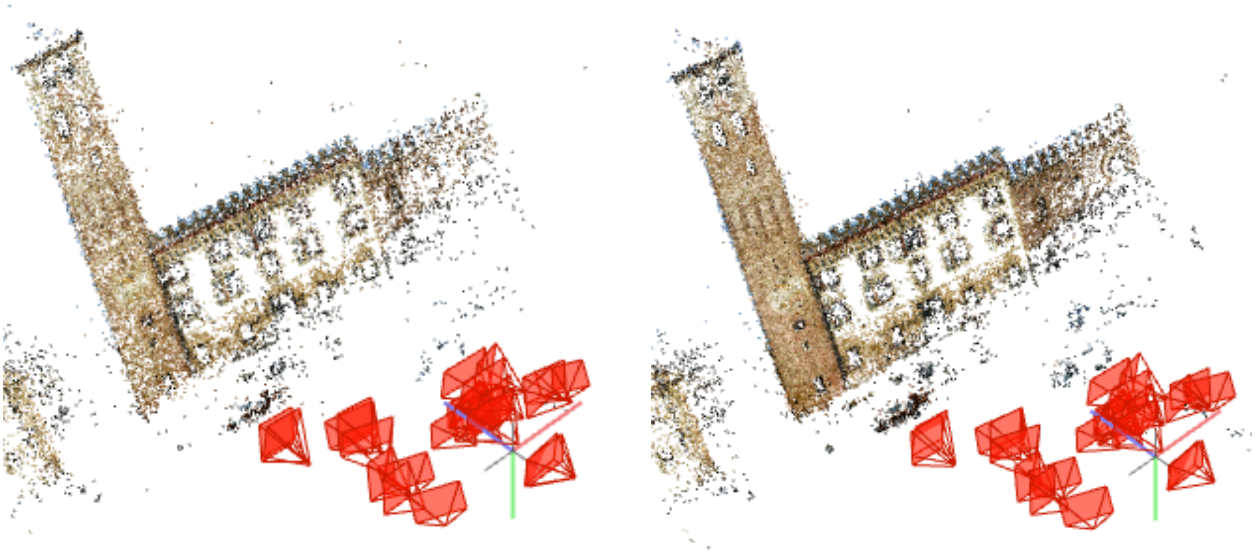
By performing a reconstruction in COLMAP, a final result is obtained and in this case, as shown in fig. 9a and 10a, the fountain and the building are rebuilt in a more sparse form due to the smaller amount of information provided to the software. Fig. 9b and 10b refer to the final result without the use of the autoencoder and therefore all the features and matchings obtained by the SURF algorithm are supplied to COLMAP without regression. Comparing the two results, significant differences are visible in the reconstruction quality of the subjects.



(a) Fountain reconstructed with the autoencoder.

(b) Fountain reconstructed without the autoencoder.

Figure 9: Fountain reconstruction using COLMAP.



(a) Tiso reconstructed with the autoencoder.

(b) Tiso reconstructed without the autoencoder.

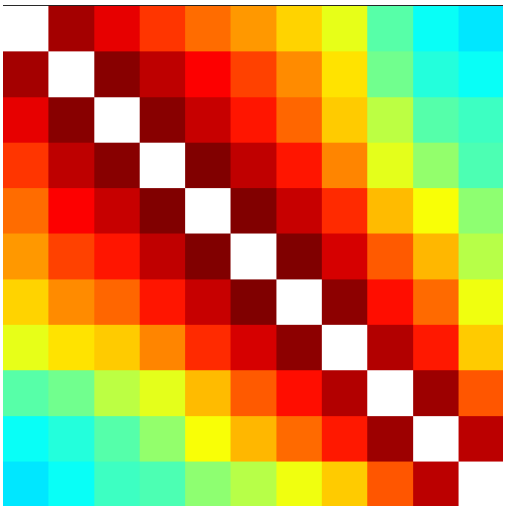
Figure 10: Tiso reconstruction using COLMAP.

From the parameters obtained after the reconstruction, reported in the table 1 regarding the Fountain-P11 dataset, there is a mean observations per image equal to 4751.36 and a total number of observations equal to 52256, in the case of the compression strategy. Data are almost a third lower than those obtained in both cases with the SURF algorithm without using the autoencoder. Also the other parameters have substantial differences such as the number of points that is half between the compared data.

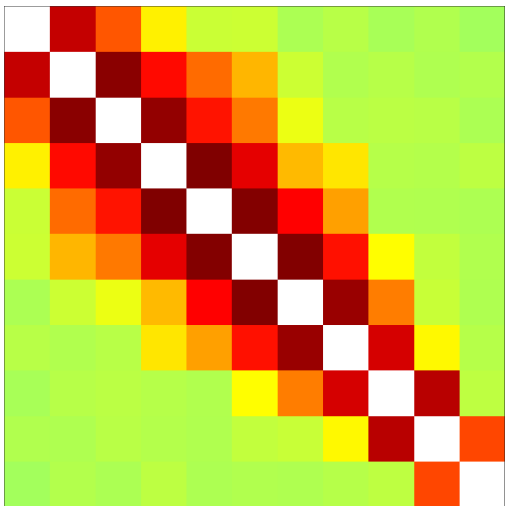
| **Parameters** | **No autoencoder** | **Autoencoder** |
|---|---|---|
| Cameras | 11 | 11 |
| Images | 11 | 11 |
| Points | 30620 | 14674 |
| Observations | 139909 | 52265 |
| Mean track length | 4.5692 | 3.56174 |
| Mean observations per image | 12791 | 4751.36 |
| Mean reprojection error | 0.49785 | 0.428602 |

Table 1: Result comparisons for Fountain-P11 between the no-autoencoder and autoencoder strategies.

Fig. 11 reports the difference between the match matrices in the case of regression or not and displays a qualitative measure of the matchings in the reconstruction process in COLMAP. Figure 11b displays a smaller variance, i.e. a smaller number of matchings per feature, due to the compression.



(a) Match matrix without the autoencoder.



(b) Match matrix with the autoencoder.

Figure 11: Match matrix comparison.

This type of strategy is surely a *lossy* compression, which sacrifices part of the original information during the decompression phase of the data in order to reduce the computational complexity and/or the original dimensions. It can be useful in environments where the quality of reconstruction is not important, for example in object detection, where is not crucial to exactly reconstruct the object with respect to the detection result "true/false". It can be seen from the compressed images that the edges of the objects are mostly preserved while straight planes are "thinned out", i.e. represented with less points. This means that the most important part of the original information (larger variance) is preserved and classifies the autoencoder compression strategy as a useful tool in many appliable fields.

# 6 MATLAB Code

```
%% TRAINING
clear all; close all; clc;

imdsTraining = imageDatastore(...
    'images\Training', 'IncludeSubfolders', 1);

SURF_features=[];
for i = 1:numel(imdsTraining.Files) % for each image

    images{i} = rgb2gray(readimage(imdsTraining, i));

    [points_training{i}, valpoints] = extractFeatures(images{i}, ...
```

```
        detectSURFFeatures(images{i}, 'NumOctaves', 4, ...
        'MetricThreshold', 50), 'Upright', true);

    imshow(images{i}); hold on;
    plot(valpoints); hold off;

    A = double(points_training{i});

    SURF_features = [SURF_features; A];
end

close all; clc;

layers = [
    featureInputLayer(64,'Name','IN')
    fullyConnectedLayer(6,'Name','mid2')
    fullyConnectedLayer(64, 'Name', 'mid4')
    regressionLayer( 'Name', 'OUT')];

lgraph = layerGraph(layers);

figure; plot(lgraph);

options = trainingOptions('adam', ...
    'MaxEpochs',2, ...
    'Shuffle','never', ...
    'Verbose',false, ...
    'Plots','training-progress');

compressed=SURF_features;
autoenc = trainNetwork(SURF_features,compressed,lgraph,options);

save('Workspace_autoenc_trained.mat');

%% TESTING

clc; clear all; load('Workspace_autoenc_trained.mat');

imdsPortello = imageDatastore(...
    'images\Training\portelloDataset', 'IncludeSubfolders', true);

imdsCastle = imageDatastore(...
    'images\Training\castle', 'IncludeSubfolders', true);

imdsFountain = imageDatastore(...
    'images\Testing\fountain-P11', 'IncludeSubfolders', true);

imdsTiso = imageDatastore(...
    'images\Testing\tisoDataset', 'IncludeSubfolders', true);


% MATCHINGS(imdsFountain,  FEATURES(imdsFountain, autoenc));
MATCHINGS(imdsTiso, FEATURES(imdsTiso, autoenc));

```

```
67
68 %% FUNCTIONS
69
70 function features = FEATURES (imds, autoenc)
71
72     delete('features/tiso/*.txt'); delete('features/fountain/*.txt');
73     delete('features/portello/*.txt'); delete('features/castle/*.txt');
74
75     for i = 1:numel(imds.Files)
76
77         images{i} = rgb2gray(readimage(imds, i));
78
79         [descriptors{i}, valpoints] = extractFeatures(images{i}, ...
80             detectSURFFeatures(images{i}, 'NumOctaves', 4, ...
81             'MetricThreshold', 50), 'Upright', true);
82
83         A=double(descriptors{i});
84
85         Y_test = predict(autoenc, A); % WITH AUTOENCODER
86 %         Y_test = A; % WITHOUT AUTOENCODER
87
88         %plot of the images%%%
89         imshow(images{i}); hold on;
90         plot(valpoints); hold off;
91         %%%%%%%%%%%%%%%%%%%%%%
92
93         A =[double(valpoints.Location), ...
94             double(valpoints.Scale), ...
95             double(valpoints.Orientation), ...
96             zeros(length(valpoints.Location),128)];
97
98         [path,name,ext] = fileparts(string(imds.Files{i}));
99         if contains(path,'fountain','IgnoreCase',true)
100             filePath = 'features/fountain/' + string(name)+string(ext) ...
101                 +'.txt';
102
103         elseif contains(path,'tiso','IgnoreCase',true)
104             filePath = 'features/tiso/' + string(name)+string(ext) ...
105                 +'.txt';
106
107         elseif contains(path,'portello','IgnoreCase',true)
108             filePath = 'features/portello/' + string(name)+string(ext) ...
109                 +'.txt';
110
111         elseif contains(path,'castle','IgnoreCase',true)
112             filePath = 'features/castle/' + string(name)+string(ext) ...
113                 +'.txt';
114         end
115
116         features{i} = single(Y_test);
117
118         fileID = fopen(filePath, 'w');
119         fprintf(fileID, string(length(A))+' 128\n');
120         fclose(fileID);
```

```
121
122         writematrix(A, filePath, 'WriteMode', 'append', ...
123             'Delimiter', 'space');
124
125     end
126
127     close all;
128 end
129
130
131
132 function MATCHINGS (imds, features)
133
134     delete('matchings/tiso/matchings.txt');
135     delete('matchings/fountain/matchings.txt');
136     delete('matchings/portello/matchings.txt');
137     delete('matchings/castle/matchings.txt'); clc;
138
139     for i=1:length(features)-1
140         for j=i+1:length(features)
141
142             matchings = matchFeatures(features{i}, features{j}, ...
143                 'MaxRatio', .9, 'Unique',  true, 'Method', 'Approximate')-1;
144
145             [path,name_from,ext_from] = fileparts(string(imds.Files{i}));
146             [¬,name_to,ext_to] = fileparts(string(imds.Files{j}));
147
148             if contains(path, 'tiso', 'IgnoreCase', true)
149                 filePath = 'matchings/tiso/matchings.txt';
150             elseif contains(path, 'fountain', 'IgnoreCase', true)
151                 filePath = 'matchings/fountain/matchings.txt';
152             elseif contains(path, 'castle', 'IgnoreCase', true)
153                 filePath = 'matchings/castle/matchings.txt';
154             elseif contains(path, 'portello', 'IgnoreCase', true)
155                 filePath = 'matchings/portello/matchings.txt';
156             end
157
158             title = string(name_from)+string(ext_from) +' '+ ...
159                 string(name_to)+string(ext_to) + '\n';
160
161             fileID = fopen(filePath, 'a');
162             fprintf(fileID, title);
163
164             writematrix(matchings, filePath, 'WriteMode', 'append', ...
165                 'Delimiter', 'space');
166
167             fprintf(fileID, '\n');
168             fclose(fileID);
169
170         end
171     end
172
173 end
```