REPORT

PROJECT: Underwater Target Detector

(Computer Vision Course)

Authors

David BENSABAT
Xavier YVONNE

Teacher

Laurent BEAUDOIN
Loïca AVANTHEY

February 8, 2019

Table des matières

1 Overview

This project deals with computer vision on underwater images. Some images contain a 9x10 checkerbard and/or some square targets with a digit (or the number 10) on them, while other don't. We do the following:

• **Classification / Detection:** find which images have at least one checkerboard and/or a target.
• **Localization:** Among the images that do have one, find the position of targets.

Reading the number written on a target, while greatly appreciated, was beyond our scope.

We were given a dataset consisting in 69 images (format: JPG, resolution: 3840x2880 or 3860x2760 pixels), 46 of them containing at least one checkerboard and/or a target, and 23 images without particular contents.

Our project is fully written in a Python 3. We used in particular the following libraries:

• NumPy (image storing as NumPy arrays)
• Matplotlib (image display)
• Skimage (geometric transformation on images)
• OpenCV 3+ (image processing, machine learning)
• Scikit-learn (machine learning)
• Keras (deep learning) (with TensorFlow backend)

The code is written in a Jupyter Notebook. Just open it in a web browser that supports it, and run all cells.

2 Preprocessing

2.1 Data Augmentation

Our dataset (69 images) is too small. To make it larger, we applied various geometric transformations on the given images: rotations, filps (left/right or upside/down) or both. We finally came with 690 images.

2.2 Labeling

In order to localize the targets, we annotated by hand (basically by drawing a box on them) their position on the original images, by giving the center of targets and its width. Then we derived, using a handful of math formulas, the position of the targets on all generated images of our augmented dataset.

2.3 Image Processing

The images from our dataset take much memory space and are noisy. To address this, we did the following steps:

• Convertion to grayscale, as color does not matter a lot here (but contrast does!)
• Rescaling: shrink all images to 368x276 pixels.

• CLAHE (Constrast Limited Adaptive Histogram Equalization) , see e.g. https://en.wikipedia.org/wiki/Adaptive_histogram_equalization. This enhances the contrast with slightly better performances for our models than the regular AHE seen in courses.

• Apply a median (blurring) filter to reduce the noise.
• Use Sobel operators (magnitude of the gradient) to enhance the edges and facilitate the detection. This single step improved the accuracy of our detector by 23%!

2.4 Image storing

Once the preprocessing is done, all images are encoded in a NumPy array in the naive way: 1 row per image, each entry being the intensity of a pixel.
Despite the large size of the resulting array (690 x 101568), we never had any problem to train our models on it and getting fair performances (see Section 3).

3 Machine Learning

3.1 Classification with SVM

To find which images have at least one checkerboard and/or a target in them, we trained an SVM (Support Machine Vector) using the OpenCV library. The trainAuto() method suggested to take C=0.1 as an optimal value for the C parameter. We measured the performance of our classifier using k-cross validation with k=5.

We got 68% as accuracy score on our training dataset.

3.2 Object Bounding Box Detection

To draw a box around each detected target, we trained a Convolutional Neural Network with two heads, one using classification to distinguish checkerboards from targets, and one using regression to predict the position of the box.

4 Conclusion

This project was deceitfully simple. It was quite clear for us how to do the job, but we stumbled on some very noisy images, impacting the overall performances. We are confident that a good image preprocessing and well-chosen descriptors are the key for efficient classification and detection. To get better descriptors and in particular to reach invariance through translation and rotation, we considered a while using HOG or SIFT descriptors. While such an approach would undoubtedly lead to far superior results, we were unfortunately lacking of time to fully undestand the underlying theory and implement it.

We found this project quite difficult, but very challenging.