Donadel Denis
Mat 1206744
Computer Vision (Final Project)

REPORT OF COINS DETECTION PROJECT

Introduction

The task to be done in this project is not so simple and it can be divided into two main different steps. Firstly it's necessary to detect the shape and the location of coin from the image representing many different of them in order to be able to identify one coin at the time. After that, we have to use some technique to recognize the value of each coin. It can be done via different techniques like using a pattern matcher (for instance, SIFT) or using an Artificial Neural Network. I decide to use Machine Learning because I studied it and I want to try it in a real project.

Coin Detection

In order to detect coins from images I decide to use OpenCV and try some techniques learned during the course.
First of all I convert the image to a grayscale and I apply to it a *GaussianBlur* to smooth the imperfection of the picture.

After that I need a threshold for apply *Canny Algorithm* and find edges. In the beginning I tried with *adaptiveThreshold* using a Gaussian threshold but after some time I decide to use threshold with *Otsu* that provide significantly better results. An example of the applications of this threshold is shown in Figure 1.

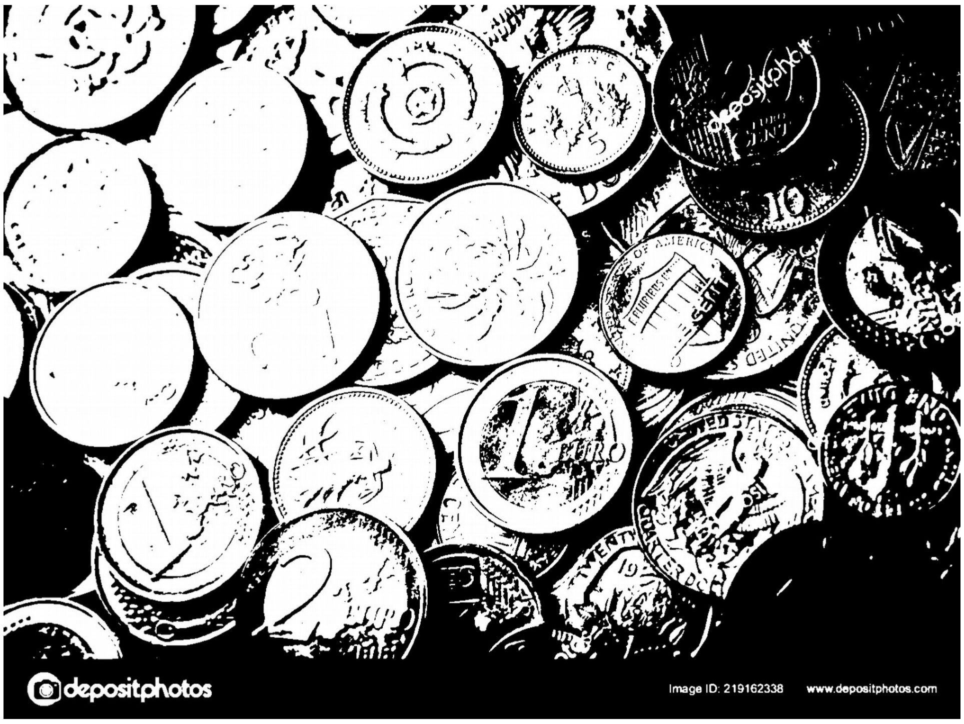

Figure 1: Otsu Threshold

After the application of Canny I apply some *morphological operations* in order to try to make shapes in the image more clear. I discover a good solution applying erosion followed by some iterations of dilation with a rectangular kernel in order to try to “complete” circles also when them was covered by other things. An example of this operations applied is shown in Figure 2.

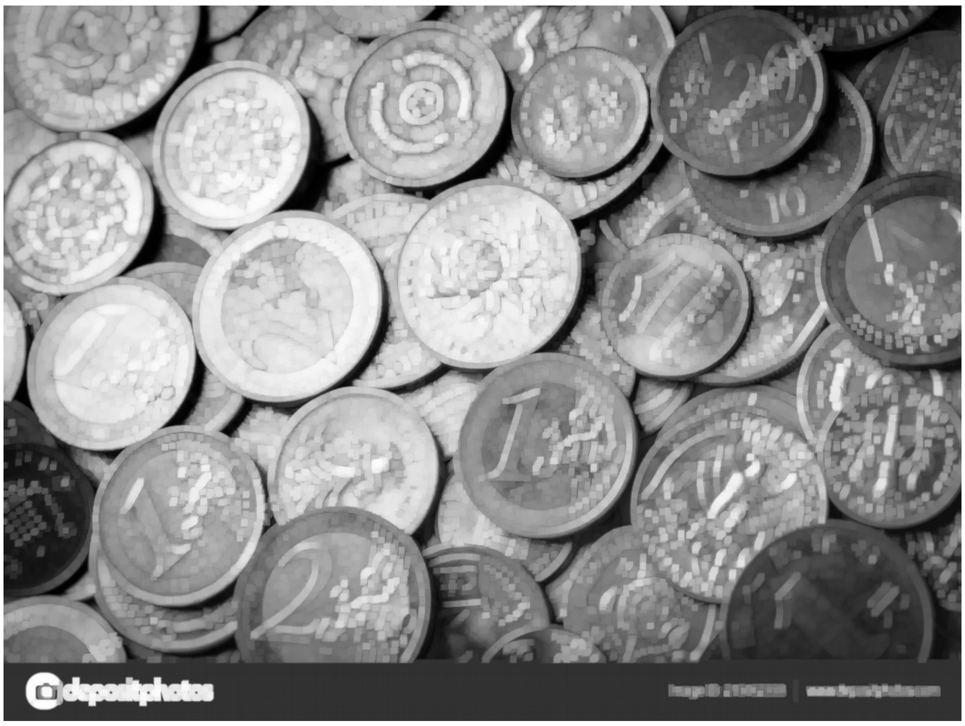

Figure 2: Morphological Operations

Finally, I use *HoughCircles* to find circles in the image. Before I tried using *findContours* function but, except some particular cases, the results were poor.
A good feature of *findContours* function was that it can recognize also ellipses and so coins photographed from an unusual prospective, but I discover that *HoughCirles* generally detect anyway this case of coin ass a circle a little bit bigger than the ellipse.
HoughCircles get many parameters and find good ones for the many different images that the software can get in input was a big challenge.

I applied a limitation to the number of images that can get in output in order to delete the last circles detected that in general are not so strong and so it’s useful to get a prediction on them. I decide also to discard circles that go too much outside the frame calculating the area of them and using a variable (epsilon) to set the threshold.

Every images is saved as file and a vector with coordinates of circles was maintain in the object. We use it later to print the predictions.
After this process is completed, the software call “tester.py” Python script that guess the value of the single coin pictures.

Artificial Neural Network
For the single coin identification I decide to set up an Artificial Neural Network.
At the beginning I try to model some network from scratch following some papers found online about this topic but the results were not so good. So I decide to move to Transfer Learning and use a power and pre-trained network such as *InceptionV3*, a network is used also by big company like Google.

I set up a model with this net changing the last 1000 neurons fully connected layer to a list of two fully connected layers which have the last one (a *softmax*) with 5 output that indicates the four different coins (e.g. 20 cent, 50 cent, 1 euro, 2 euro) and an unknown category. Finally, I lock the weights of the first layers and I train changing only the weights of the last ones.

I create the dataset used for training by my own merging some dataset and generating new images. The base of the dataset is the one provided with the specifications (https://github.com/kaa/coins-dataset). This were the training data for the correct coins. For the *unknown* category I use the remaining data of the coins that I don't have to discover (e.g. 1 cent, 2 cent, 5 cent, 10 cent), a dataset of Roman coins and many other images generated in the following way: I collect from Google some images representing a large number of coins, so I apply to it the process explained in the previous section but using a very permissive parameters of *HoughCircles* in order to generate by every image about 300/400 sub-images representing single coins, many coins together, details or in some cases just noise. I clean up this large set and I use it for training.

I use a pre-process function provided by *InceptionV3* together with some data augmentation. So I split the images in two different set: 80% for training, 20% for validation.
I try to compile and train with SGD (*Stochastic Gradient Descent*) but I found better results with RMS (*Root Mean Square*) and so I keep it as optimizer.

I training for about 7 epochs obtaining quite good accuracy (0.9396).
For the train I decide to not use my laptop that has not a dedicated GPU but instead try to use the "power of Google". I discover *Google Colab*, a free platform for running snippet of code online with GPU/TPU acceleration and integration with *Google Drive* for retrieve dataset and save results. It permits me to try some different Neural Network in a suitable (but anyway long) time.
Finally I saved as a Json and .h5 files the trained network, ready to be used on my laptop.

Results
A "tester.py" Python script load the pre-trained network and predict the images generated before. Saves the result to a .csv file and return the flow to the C++ application.
It read the .csv and print circle around the well classified images with the name of the coin.
An example is in the following Figure 3.

The results are good but not perfect. The main problem was the different size of the coins between different images: in some of them there are a few big coins, in other ones the coins are many and really small. Set parameters of *HougeCircles* such that it can recognize all the coins in all the picture is a really hard challenge. With a perfect ANN it can be possible to try different parameters and to maintain the ones that provide more coins classified correctly (so, not unknown), but just with a little error rate it can provide several wrong results and also it might takes a lot of extra computational time.
Finally, I'm not an expert of Neural Network and so the accuracy is not satisfying for a real use but I think that it's possible to improve it using, for example, larger datasets.

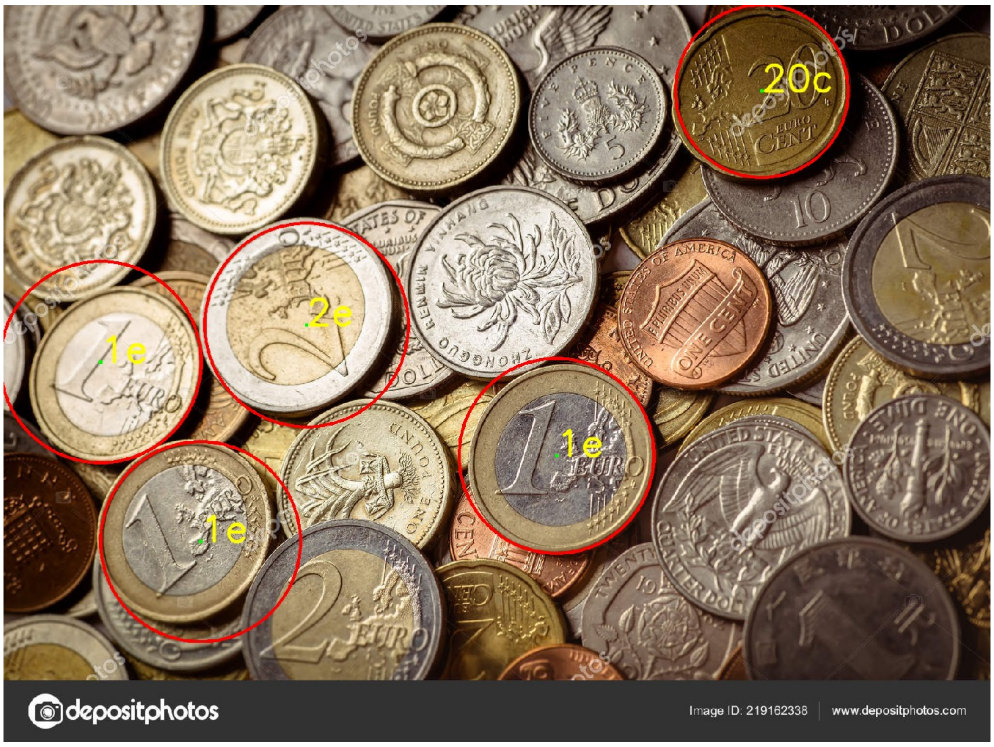

Figure 3: Final result