Captcha solver

Alex Colucci[1] and Mario Parisi[2]

[1]Università degli studi del Molise
[2]Politecnico di Bari

February 14, 2019

Abstract

Machine learning model implementation for text based captcha solving.

Project structure

Captchas stand for Completely Automated Public Turing-test-to-tell Computers and Humans Apart.

Captcha is defined as a general task that should be very hard to be solved by a computer but must be very easy to be solved by a human.

The goal of the project is to build software to autonomously solve Captchas. In particular, the Captchas that the software will try to solve are based on the character recognition task. The next paragraphs will describe all the processing applied on an input Captcha in order to retrieve the solution, i.e. the associated text.

The project is logically divided into three parts: pre-processing, model definition for the machine learning part and the GUI to interact with the program. In the following paragraphs, the folders structure is analyzed to provide the reader with an overview of the project.

Dataset folder

Contains the original folder that contains the captcha dataset but simplified (the number of elements is reduced by a factor of 2). At the first run of `initialize_dataset.py` grayscale, binarized and segmented folders will be initialized. Every step is described in paragraph . Pickle folder will contain the splitted dataset after the first run of `train.py`.

Gui

Contains all the resources for the graphical user interface.

History

This folder is initialized the first time after the run of `save_history.py`. The concept behind is to save the current dataset folder with the last model created, this is useful to draw conclusions.

Managers

Contains `image_manager.py` that deals with providing image manipulation features. Than `dataset_manager.py` is used to do all the initialization of the dataset folder and the previous script is used to achieve that. Finally there is `controller.py` that provides a higher interface that is used from the Gui part.

Trained models

Contains all the trained models divided into two sub-folders: neural networks and support vector machine.

Dataset analysis

The main idea of the project is to work on the following captcha dataset. It contains 564 different captchas images with relative solutions to perform supervised training. Every image contains 4 letters with background noise. Moreover, there are letters of the English alphabet missing: i, j, k, l, u, v, x.

After this premise, several steps must be performed to obtain good quality input for the machine learning model. (sol)

From now on the following image will be taken as a reference.

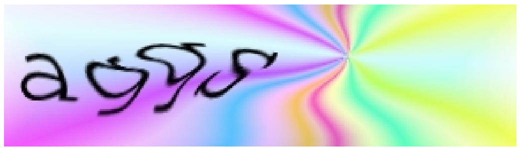

Figure 1: 0.158788403802649.jpg

Step 1 - Simplify dataset

Every image has a name like this `0.158788403802649.jpg` with another file containing the solution and named `0.158788403802649.jpg.txt`. To obtain something that is more comfortable to work with and to reduce the number of the files that must be read, the image in this step is renamed with the solution that is contained in the text file. So the final output will be for example `aggs.jpg`. If there are conflicts, a number will be added at the end of the file's name, for example `aggs(1).jpg`.

Step 2 - Grayscale dataset

Colors are unnecessary and do not provide vital information for the purpose of the problem to be addressed. Thus the image is transformed from RGB to a Gray monochrome, composed exclusively of shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest. (pho).

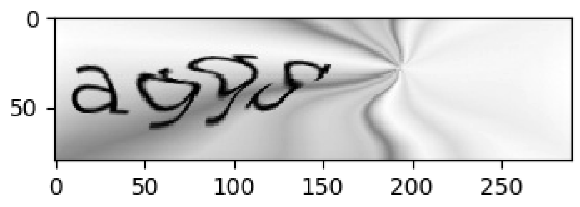

Figure 2: grayscale aggs.jpg

Step 3 - Binarize dataset

After the second passage, there is still a problem to deal with, that is noise. The idea is to set a specific threshold and every pixel that is bigger will be set to white otherwise to black. Translated into code is `binarized = image > threshold`. To set the threshold there are different algorithms and in this project *Minimum thresholding* and *Isodata* are used. The former for instance takes a histogram of the image and smooths it repeatedly until there are only two peaks in the histogram (doc). Unfortunately sometimes the histogram is too flat and in that case, the latter comes into play.

The isodata divides the image into object and background by taking an initial threshold, then the averages of the pixels at or below the threshold and pixels above are computed. The averages of those two values are computed, the threshold is incremented and the process is repeated until the threshold is larger than the composite average (197, 1978).

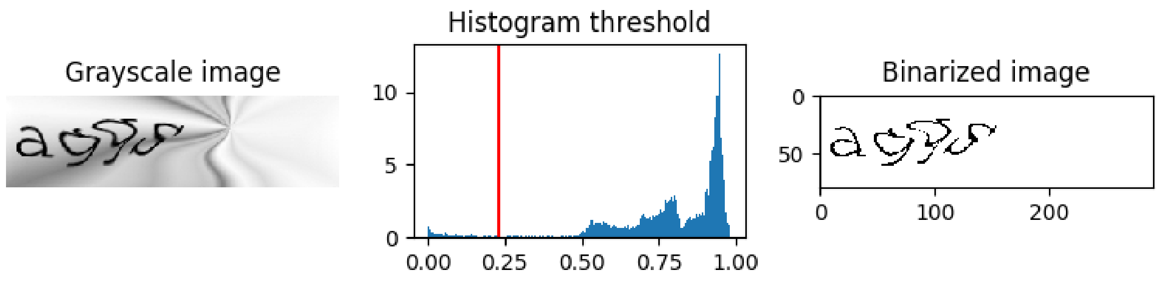

Figure 3: binarized aggs.jpg

Step 4 - Segmentation

This is the crucial step that takes from the captcha image every single letter and transform it in a 50 x 50 image. The first thing to do is to create a matrix called *Markers* (4), which applies a threshold to the binarized image in this way:

```
markers[binarized > 0.1] = 1
```

```
markers[binarized <= 0.1] = 2
```

Secondly is created an elevation map through the research of the edge magnitude using the *Sobel transform* (wik).

Thirdly a *watershed algorithm* is applied on the markers and on the elevation map and the third matrix is obtained.

Finally, the proper matrix that yields to the real segmentation, the labeled. The segments are enumerated starting from 1, while the background is 0.

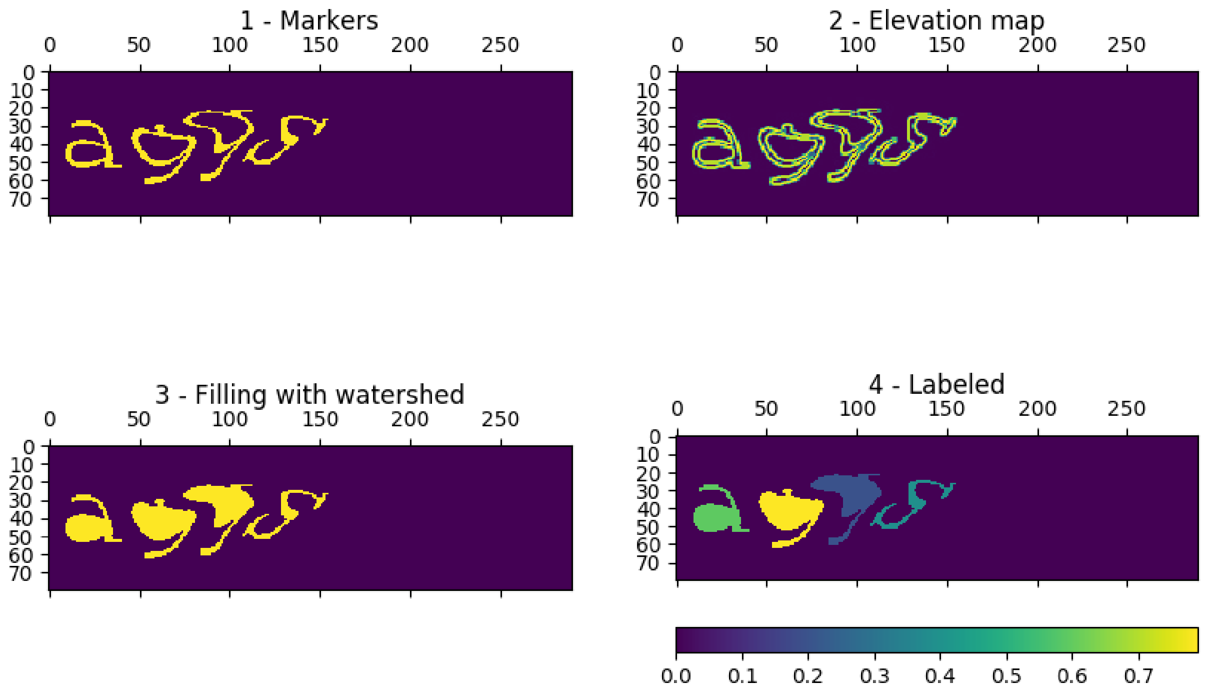

Figure 4: Graphs of the matrices markers, elevation map, watershed and labeled

Nevertheless, it is still necessary to make small adjustments and the reason why is the possible noise like in figure 5 . The solution is rather simple, the segment is kept only if the area is above a certain threshold. In the case of this project, the number 15 has opted for the job.

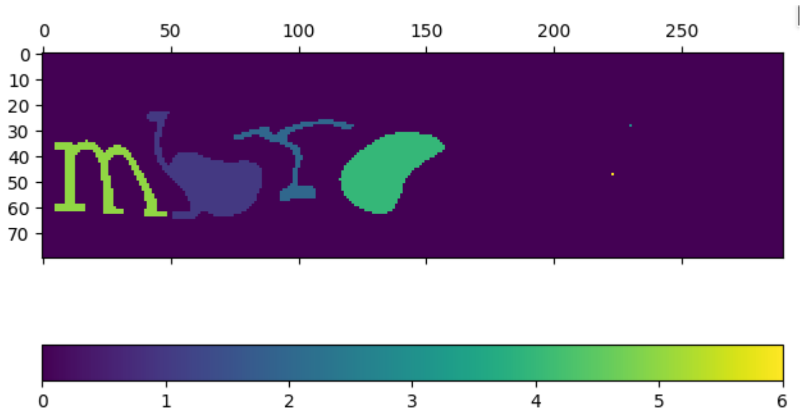

Figure 5: More than four segment because of the noise

However, the last is the best case scenario. Unfortunately cases with more segments whose area is bigger than the determined threshold or with fewer segments than 4, that is the number of letters, have to be taken into account.

The former is solved by doing the erosion (ski). Which consists of the disappearance of white from the border of an object. The algorithm is applied until less than four segments can be found.

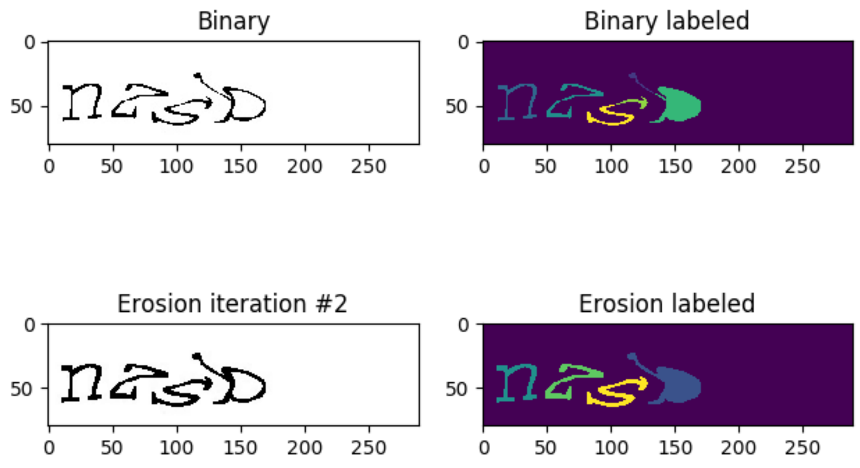

Figure 6: Erosion algorithm to decrement the segments

This brings to the latter, the case where less than four segments are presents, because of the erosion algorithm that can overshoot or because there are images that have letters attached together. The following algorithm has been thought:

1. find the segment with the maximum width;
2. build a vector *sum* that contains for each column the sum of all the rows;
3. build a vector *weight* that contains the distance from the middle point multiplied by 0.7;
4. subtract the weight from to the sum vector;
5. find the column with the maximum value and cut there, that will be the one with more white.

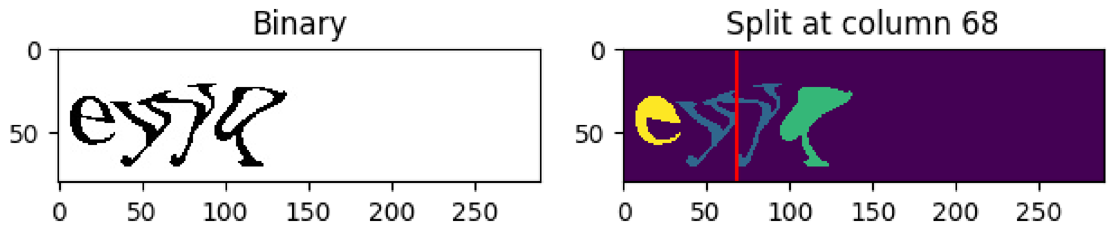

Figure 7: Split of an image with 3 segments

Project implementation

To implement the software Python has been used because of the several scientific capabilities offered by different libraries dedicated to math and machine learning but even for its ease of use. TensorFlow, Keras, and Scikit-learn frameworks have been used to implement the different machine learning algorithm. To process the images scikit-image has been used. Furthermore, Numpy has been used to handle the dataset in a vectorized way. To solve the captcha recognition problems, the following methods have been implemented:

- Neural-Network with REctified Linear Unit(ReLU) activation function
- Neural-Network with Hyperbolic tangent(tanh) activation function
- SVM

Neural network introduction

A Neural network is a machine learning model, used in supervised learning, that try to simulate the structure of biological neurons and their interconnections. A neural network has one input layer, with a number of neurons equal to the number of features of the example to be classified. After the input layer, the network can have one or more hidden layer composed of a variable number of neurons. Finally, the last layer of the network, called the output layer, must have a number of neurons equal to the number of different classes that have to be classified.

Every neuron has an activation function, generally ranging from 0 to 1, or from -1 to 1. Often the neurons of the last layer have an activation function ranging from 0 to 1, in order to express the probability of the input of being of the type associated with the specific neuron.

In our scenario, the first layer of the neural network will always contain 2500 neurons, one for every pixel in the segmented image representing a single letter. Instead, the last layer will be composed of 19 neurons, one for each possible letter. Furthermore, the activation function for the neurons of the output is a softmax function, which values ranges from 0 to 1.

Neural Network with ReLu activation

The first kind of neural network implemented is composed of three layers. The only hidden layer is composed of 128 neurons with a ReLu activation function. We obtain a mean of the accuracy equal to 86.98% with a standard deviation of 2.25%

Trying to add a second hidden layer composed of 32 neurons to the two neural networks, the results obtained are worse and more dependent on the particular train and test set selected. Indeed, the mean of the accuracy given by the 10-fold validation method is 77.97% with a standard deviation of 7.51%.

Instead, if we add a second hidden layer of 50 neurons, we obtain results very similar with the single layer architecture. The mean accuracy returned by the 10-fold cross validation is 85,96% with a standard deviation of 2,08%

Neural Network with Hyperbolic tangent

The second kind of neural network has an architecture that has been demonstrated to be particularly useful for the pattern recognition task (Bostik 2018). This network uses the Hyperbolic tangent sigmoid function for the neurons of its only hidden layer. In this case, the following results have been obtained. Mean accuracy k-fold = 87,43% (2,24% standard deviation). Adding a second layer of 50 neurons we obtain a little bit worse result with a mean of 84.31% with a 3.03% standard deviation.

Support vector machines

SVM is a machine learning model that, tries to find the hyperplane that best separate the input examples belonging to different classes. Some input dataset cannot be separated linearly, in these cases SVM can still find the right hyperplane thanks to kernels. A kernel transforms a given input space in a higher dimensional space. Doing so, SVM can separate the data in the higher dimension space. We tested the following three kernel on our dataset, getting the results shown below:

- Linear kernel : $K\left(x,\ x_i\right) = \Sigma\left(x \cdot x_i\right)$

Accuracy mean 93,54% with a standard deviation of 1,89%

- Polynomial kernel : $K\left(x,\ x_i\right) = 1\ + \Sigma\left(x \cdot x_i\right)^d$

Accuracy mean: 82.70% with a standard deviation of 2.21%

- Radial basis function kernel : $K\left(x,\ x_i\right) =\ e^{\left(-\gamma \cdot \Sigma\left(x\ - x_i^2\right)\right)}$

Accuracy mean: 85.11% with a standard deviation of 2.02%

Fitting evaluation

To check if our models were overfitting or underfitting the dataset, we have compared the values of the mean accuracy on the train data and test data returned by the 10-fold cross-validation method. Using several hyper-parameter configurations, we obtained the following notable results:

Model name	Hyper-parameters	Train set accuracy	Evaluation set accuracy
Neural network	Epochs = 5	83.06%	75.97%
Neural network	Epochs = 10	92.41%	82.52%
Neural network	Epochs = 20	97.82%	87.16%
SVM	Linear kernel, C = 0.001	91.87%	87.16%
SVM	Linear kernel, C = 0.005	98.96%	92.86%
SVM	Linear kernel, C = 1	100%	93.53%

As we can see the models that perform best on test data, are the same ones that perform best on the training data, even if the models are clearly overfitting the train data set. This is an unusual result because overfitting should lead to worse performance on unseen data. The explanation of this behavior is that the test data are too similar to the training data.

Conclusion

The best Machine learning for this particular Captcha dataset resulted to be SVM with a linear kernel and a regularization parameter C equal to one. However, we noticed that all the models that overfit the train set give better results than the models that do not. This makes us think that the training set and the test set are too similar. Hence a good candidate as next classifier to try is the nearest neighbor classifier.

However, to find a model that can solve a more general captcha problem, a bigger dataset is needed.

As a future work would be interesting to test the trained algorithms on new test sets, and trying to find a better tuning for the Hyper-parameters of the models used.

References

Thresholding — skimage docs. http://scikit-image.org/docs/0.13.x/auto$_{e}xamples/xx_{a}pplications/plot_{t}hresholding.htmlbimod$
$histogram.URL$. Accessed on Tue, February 12, 2019.

Stephen Johnson on Digital Photography. https://books.google.com/books?id=0UVRXzF91gcC&pg=PA17&dq=grayscale+
and-white-continuous-tone&ei=XlwqSdGVOILmkwTalPiIDw&redir$_{e}sc = y.URL$. Accessed on Tue, February 12, 2019.

Morphological Filtering — skimage. http://scikit-image.org/docs/dev/auto$_{e}xamples/xx_{a}pplications/plot_{m}orphology.htmle$
Accessed on Wed, February 13, 2019.

Character Segmentation for Automatic CAPTCHA Solving. https://benthamopen.com/ABSTRACT/COMPSCI-1-1. URL https://benthamopen.com/ABSTRACT/COMPSCI-1-1. Accessed on Wed, February 13, 2019.

Sobel operator - Wikipedia. https://en.wikipedia.org/wiki/Sobel$_{o}perator.URL$. Accessed on Tue, February 12, 2019.

Picture Thresholding Using an Iterative Selection Method. *IEEE Transactions on Systems Man, and Cybernetics*, 8(8):630–632, 1978. 10.1109/tsmc.1978.4310039. URL https://doi.org/10.1109%2Ftsmc.1978.4310039.