Numerical Hand Gesture Classifier

Jose Millan
jampmil@gmail.com
Data Mining & Knowledge Management Masters Program
Università degli Studi del Piemonte Orientale 'Amedeo Avogadro'
Alessandria, Italy

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis tempus lacus eget congue tincidunt. Morbi a velit justo. Maecenas consectetur eleifend gravida. Vivamus hendrerit ante vel est dictum vehicula. Quisque venenatis rutrum massa, at consectetur mauris posuere id. Integer fermentum, nulla ac mollis convallis, tellus nunc molestie nunc, eget vulputate mi odio eget nulla. Nulla facilisi. Phasellus tempus diam ligula, non tempus elit tristique ac. Donec volutpat neque a arcu pellentesque facilisis. Fusce a tempor eros. Integer imperdiet turpis vel lacus porta, vitae convallis dui luctus. In lobortis, ex id malesuada fringilla, augue metus mollis ipsum, sed auctor nisi turpis nec arcu. Nulla diam felis, congue a odio vitae, sodales fermentum turpis. Nunc vel hendrerit metus, ultricies ullamcorper ligula.

1 Introduction

Hand gesture recognition implies the use of image processing along with visual data mining to correctly identify specific gestures that are being shown. The current work presents a simple classifier that using a SVM model, is able to identify between five different gestures (numbers from 1 to 5) when they are shown towards a camera.

2 Acronym List

The following acronyms are used through the current work and are necessary to understand its contents.

- SVM: Support Vector Machine
- RGB: Red, Green, Blue
- HSV: Hue, Saturation, and Value
- OpenCV: Open Source Computer Vision Library
- TODO MORE!

3 Theoretical Assumptions

3.1 Thresholding

TODO!!!

3.2 Hue, Saturation, and Value - HSV

TODO!!!

3.3 Support Vector Machine - SVM

Support Vector Machine (SVM) [1] is a supervised machine learning algorithm. This algorithm that can be used for either classification or regression, nevertheless they are more commonly used in classification problems.

SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes. In the case of a two features problem, as seen in Figure 1, a hyperplane as a line that linearly separates and classifies a set of data. The more features the problem has the dimension of the hyperplane increases, due to this its name [4].

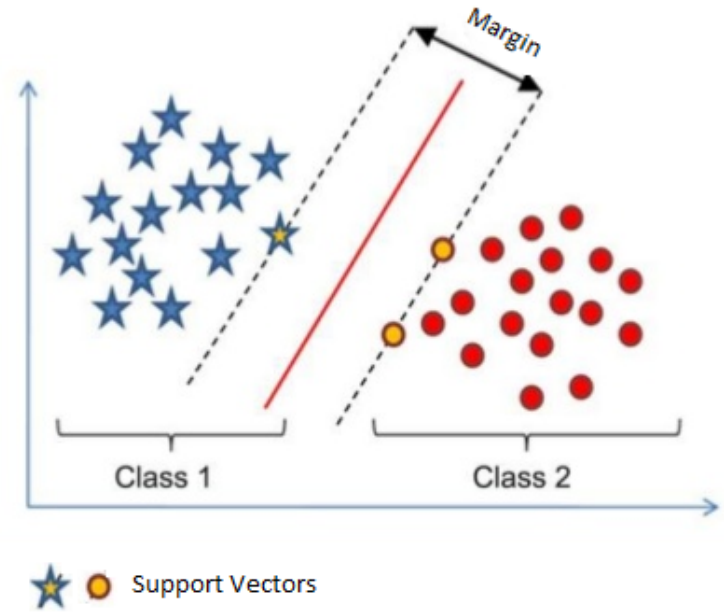

Figure 1: Simple Visualization of SVM

The further from the hyperplane the data points lie, the more confident in their correct classification. Considering this the objective of the SVM algorithm is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.

SVM is usually used for text classification tasks such as category assignment, detecting spam and sentiment analysis. It is also commonly used for image recognition challenges, performing particularly well in aspect-based recognition and color-based classification. SVM also plays a vital role in many areas of handwritten digit recognition, such as postal automation services [3].

3.4 Cross-Validatino

TODO!!!

4 Algorithm Description

The developed solution is divided in four different main tasks, each one necessary for the Hand Gesture Classifier. Below there is a description of each task and its the pseudo-code.

4.1 Preprocess Images

The first task that is necessary for the Hand Recognition Classifier is to be able to preprocess any image so afterwards its suitable for the model that is being used. For this, an ideal scenario was set, where each hand gesture is shown to the

camera against a dark color background.

With this conditions, the image is converted to HSV and then a threshold over its HSV values is applied to the image in order to get a black and white image with the contour of the gesture, where finally a blur effect is applied to soften and improve the contours. With this processed image it is ensured that the model will produce better results compared with an unprocessed image.

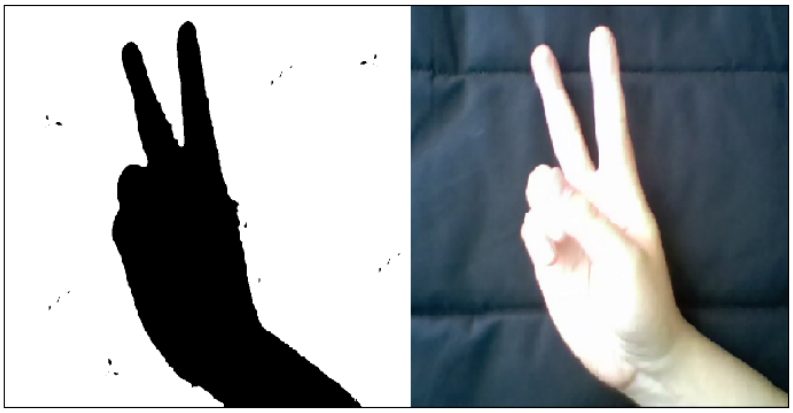

Figure 2: Final Preprocessing of the a Hand Gesture Image

Figure 2 shows the processed image with its correspondent original image, where the contour of the hand is clear for the model to use the processed image. Algorithm 1 shows the steps to process the images and to get the result shown, which in the implementation refers to the function `Mat processImage(Mat img, int* hsvConfig)`.

Algorithm 1: Process an Image

Data: original image, hsv configuration
Result: processed image

begin
- convert image to hsv;
- threshold original image using the configuration;
- apply blur;
- resize image to 640x480;

4.2 Create Datasets and Train/Test SVM Model

The next task of the Hand Gesture Classifier is to be able to train a SVM model based on a given dataset of hand gesture images, with their respective class, as it can seen in Algorithm 2. For this, the program will loop through the folders in the `images` folder importing and preprocessing the images (using each folder name as the class of the images it contains).

With the images processed then the algorithm divides them in the training and testing sets (with a ratio of ~30%), the first one of those is used to create and train an SVM model (using a linear kernel), storing the model in a file (`HandNumbersClassifier_01.dat`) and then its performance is tested using the testing set. Finally the algorithm prints the results of the process.

This task is implemented in the function `void trainSVM()` which itself calls the function `void createData( Mat& trainData, Mat& trainClasses, Mat&testData, Mat& testClasses, Ptr<TrainData>& trainingInData)`.

Algorithm 2: Create Datasets and Train/Test SVM Model Algorithm

Result: Trained and Tested SVM Model

begin

```
read images folder;
process images;
divide images in training and testing set;
create and train SVM model with training set;
write SVN model in file;
test model with testing set;
print test results;
```

4.3 Configure Camera

The objective of the configure camera task is to find and store the correct configuration of the HSV values for preprocessing the images afterwards. For this two windows are displayed to the user, one with the current video taken from the camera, and another, with sliders for configuring the HSV values, that shows the frames of the video processed with the current HSV values. The user is able to move the sliders to find the right configuration, and when it has been found, pressing any key will finish this task, by storing the HSV values in a configuration file (`hsv.config`).

Algorithm 3 shows the pseudo-code of this task which was implemented in the function `void configureCamera()` of the developed solution.

Algorithm 3: Configure Camera Algorithm

Result: HSV Configuration

begin

```
read stored hsv configuration;
configure window with hsv sliders;
activate camera;
while no key is pressed do
    process frames of camera with current hsv configuration;
    show original camera frames;
    show processed camera frames;
close windows showed to user;
write current hsv configuration;
```

4.4 Read From Camera and Classify Gestures

The final task of this work consists in allowing the user perform the hand gestures in front of the camera, where the program will show the user the prediction of which gesture is being presented, while typing the value of the gesture on the console. In order to allow the user change between signs and avoid possible miss interpretations, after a sign has been recognized the program will not attempt to recognize any hand gesture for the next three seconds.

For this the first step is to load the previously created SVN model, along with the stored HSV configuration. Then the camera is started and then the program shows the user three windows: one for the current camera capture, another one with the camera frames processed and finally one that shows the user which gesture is being shown to the camera. Every time the program recognizes a gesture, its value is typed in the console so the user can use it later. Finally the program stops when the user presses any key.

This task was implemented in the function `void readCameraAndPredict()` and the pseudo-code for this can be seen in Algorithm 4.

Algorithm 4: Read from Camera and Predict Algorithm

Result: predictions of the users gestures

```
begin
    load SVM model;
    read stored hsv configuration;
    activate camera;
    while no key is pressed do
        process frames of camera with current hsv configuration;
        show original camera frames;
        show processed camera frames;
        predict processed camera frame with SVM model;
        if hand gesture recognized then
            show recognized number image to the user;
            print recognized number in console;
            sleep hand gesture recognition for 3 seconds;
        else
            show no prediction image to the user;
    close windows showed to user;
```

5 Implementation

The program written for this work was developed in `C++`, using the computer vision and machine learning software library OpenCV [2] version 3.1. All images captured for this program were taken using a standard webcam of a laptop (USB2.0 HD UVC WebCam) .

As mentioned in Section 1, the current work includes five specific hand gestures of the first five numbers, along with the empty background. A sample of this gestures can be seen in Figure 3 and would be the only gestures the program can recognize. Additionally, in Figure REF!!!, the final user interface for the Configure Camera task (left) and the Read From Camera and Classify Gestures task (right) can be appreciated.

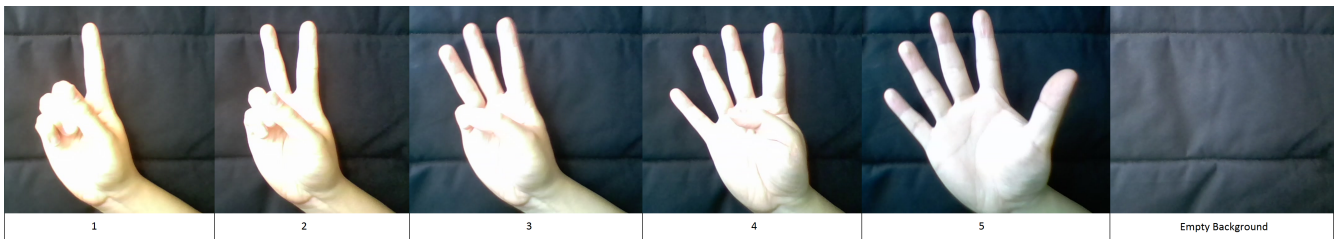

Figure 3: Hand Gestures of Numbers from 1 ot 5

The dataset of these gestures is composed of 926 images (around 150 images per gesture), which were frames taken from videos performing the gesture, by three individuals. These images are stored in the `images` folder in the root of the program.

6 Tests description

In order to find the best solution for the Hand Gesture Classifier, two different tasks of the process had to be refined: the preprocessing of the images and tuning the SVM model's parameters.

1. Preprocessing Methods:
   The first type of tests consist in preprocessing the images before they are passed over the SVM model. The different approaches are described below and a visual sample of them can be appreciated in Figure 4.

(a) No change: This is the default setting for the different tests. It simply consists in no performing any kind of preprocessing to the images.

(b) Gray Scale: Consists in converting the image into its gray scale representation. For this the default values of OpenCV where used, where $gray\_value = 0.30xR + 0.59xG + 0.11xB$

(c) Reduce Color Space: Three different images to test where produced by this method where the representations of each RGB channel was used.

(d) Thresholding: A simple threshold was applied to the image, optimizing its HSV values as explained in Section 4. For this the image was transformed from RGB to HSV and then the function `inRange` of the OpenCV library was used.

(e) Thresholding + Blur: A last approach was implemented where the threshold process was performed to the image as described before, and afterwards a noise reduction effect was added in order to reduce the empty spaces within the image. For this a median blur was used using the OpenCV function `medianBlur` with `blurSize = 5`.

2. SVM Model Tunning:
For the SVM model implemented only a liner kernel was tested (due to satisfactory final results). The challenge then was finding the correct values for the parameters C (error penalty) and γ (the linear kernel parameter). In here different values for the parameters were used, restricting the search so $0 < C, \gamma <= 10$ with intervals of 0.5. It is important to understand that for finding the correct values of these parameters, the images used were not processed at all before passing them to the SVM model.

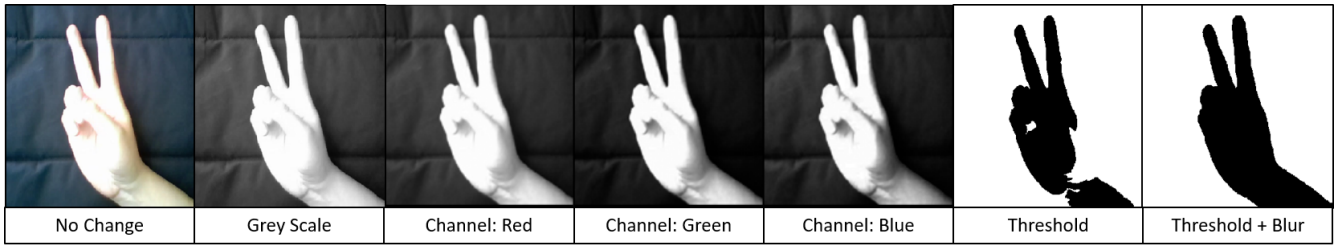

Figure 4: Final Preprocessing of the a Hand Gesture Image

In order to perform the tests described before, the dataset was divided into a training and a testing sets, with a radio of around 30% for the testing set. Finally, when the best combination for the two tasks was found, a 3-fold cross-validation was performed over the results of the trained SVM model, in order to discard any kind of over-fitting in the final model. The results for this process can be seen in Section 7.

7 Results

The results of the tests described in Section 6 are used to find the best process for the Hand Gesture Classifier. Table 1 shows the results for the different preprocessing methods, whereas Tables 2 and 3 show the results of the SVM model tunning process, which can be graphically appreciated in Figure REF!!.

Table 1: Preprocessing Methods Test Results

Preprocessing Method	Accuracy on Test
No change	94.245
Grey Scale	94.964
Reduce color space: Red	94.964
Reduce color space: Green	94.604
Reduce color space: Blue	95.324
Threshold	96.043
Threshold + Blur	96.403

γ	**Accuracy on Test with**
0.5	92.446%
1	92.806%
1.5	92.806%
2	93.165%
2.5	93.885%
3	94.245%
3.5	93.885%
4	94.604%
4.5	94.245%
5	94.604%
5.5	94.964%
6	94.604%
6.5	94.245%
7	94.245%
7.5	93.885%
8	94.245%
8.5	93.885%
9	93.525%
9.5	92.806%
10	92.446%

Table 2: γ value vs. Accuracy on Test with C = 0.5

C	**Accuracy on Test with $\gamma = 0.5$**
0.5	94.604%
1	95.324%
1.5	94.964%
2	95.324%
2.5	95.683%
3	95.324%
3.5	94.964%
4	94.604%
4.5	93.885%
5	94.245%
5.5	93.165%
6	93.525%
6.5	92.806%
7	92.446%
7.5	91.727%
8	91.007%
8.5	91.367%
9	90.647%
9.5	90.288%
10	90.288%

Table 3: C value vs. Accuracy on Test with $\gamma = 0.5$

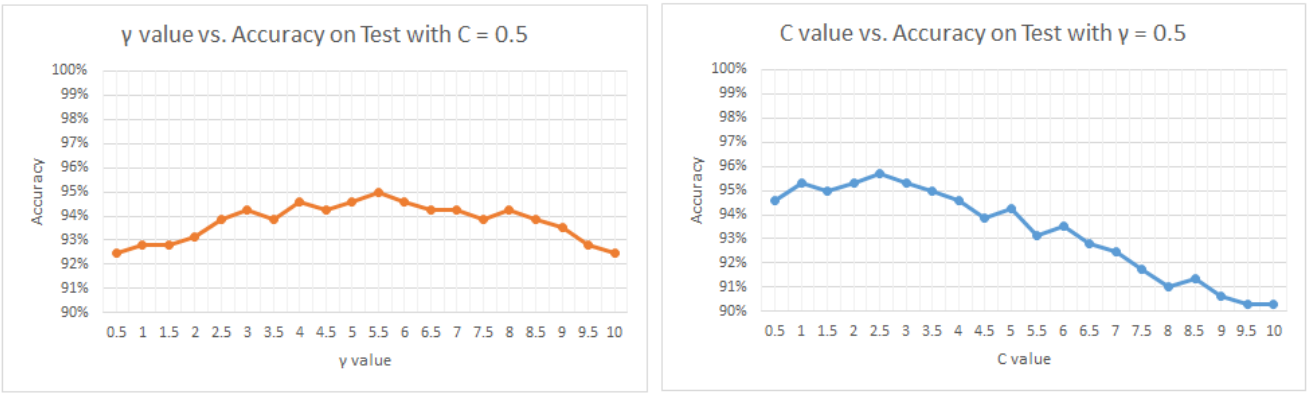

Figure 5: SVM Parameters Tunning Results

With this results the best configuration was defined as using the thresholding + blur method for preprocessing and the parameters $\gamma = 0.55$ and $C = 2.5$ for the SVM model. As mentioned in Section 6, a 3-fold cross-validation was performed over this final process, whose results can be seen in Table 4.

Table 4: 3-Fold Cross Validation for the SVM Model

Configuration	Accuracy in Training Set	Accuracy in Testing Set	Average Accuracy
thresholding + blur	99.845	98.201	98.107
$\gamma = 0.55$	100	97.942	
$C = 2.5$	99.746	98.178	

8 Conclusions and Future Work

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis tempus lacus eget congue tincidunt. Morbi a velit justo. Maecenas consectetur eleifend gravida. Vivamus hendrerit ante vel est dictum vehicula. Quisque venenatis rutrum massa, at consectetur mauris posuere id.

References

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[2] Itseez. OpenCV. `http://opencv.org/`, 2017. [Online; accessed 25-01-2017].

[3] KDnuggets. Support Vector Machines: A Simple Explanation. `http://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html`, 2016. [Online; accessed 25-01-2017].

[4] Wikipedia. Support vector machine. `https://www.wikiwand.com/en/Support_vector_machine`, 2017. [Online; accessed 25-01-2017].