

MACHINE LEARNING ALGORITHMS FOR FACE RECOGNITION

Participants: Filip Šklebar, Iris Martinović, Marko Šegon

Leader: Dražen Lučanin

[1. INTRODUCTION](#)

[2. ACHIEVEMENTS THROUGH THE PROJECT](#)

[3. TECHNOLOGY](#)

[4. METHODOLOGY](#)

[4.1. Pattern Recognition](#)

[4.1.1. The perceptron algorithm](#)

[4.1.2. SVM](#)

[4.1.3. Haar's floating window](#)

[4.2. Testing and Exploration](#)

[5. RESULTS](#)

[6. Conclusion](#)

1. INTRODUCTION

The goal of this project was to understand the basics of pattern recognition in a face recognition system. In this project we had a chance to learn the basics of programming in Python and working in Ubuntu. Apart from theory, we had a practical assignment - creating a face recognition program, which we managed to finish successfully. The project took place in several scientific fields such as informatics, engineering and mathematics.

2. ACHIEVEMENTS THROUGH THE PROJECT

There were several areas the project was connected with. The first step was setting up the environment for work where we learned some informatics skills.

After that, we took exercises in the Python programming language which introduced program engineering to us. Some of the aspects of Python we learned are:

- branches
- loops
- string operations
- functions
- files
- image processing using PIL
- object oriented paradigm basics - using objects

Finally, after implementing the different algorithms needed to achieve face recognition, we connected everything and built our program. As a result we got a program which can take a photo of a person with a web – camera and as output give the person's name from the database.

3. TECHNOLOGY

Ubuntu OS was the best choice for programming in Python because of great support for various kinds of developing libraries (python-imaging, python-opencv, v4l2capture, numpy, libsvm) which can be easily installed through the official Ubuntu software repository. We were using PyDev for developing our application. Pydev is a plug-in for programming in Python that extends the

Eclipse IDE.

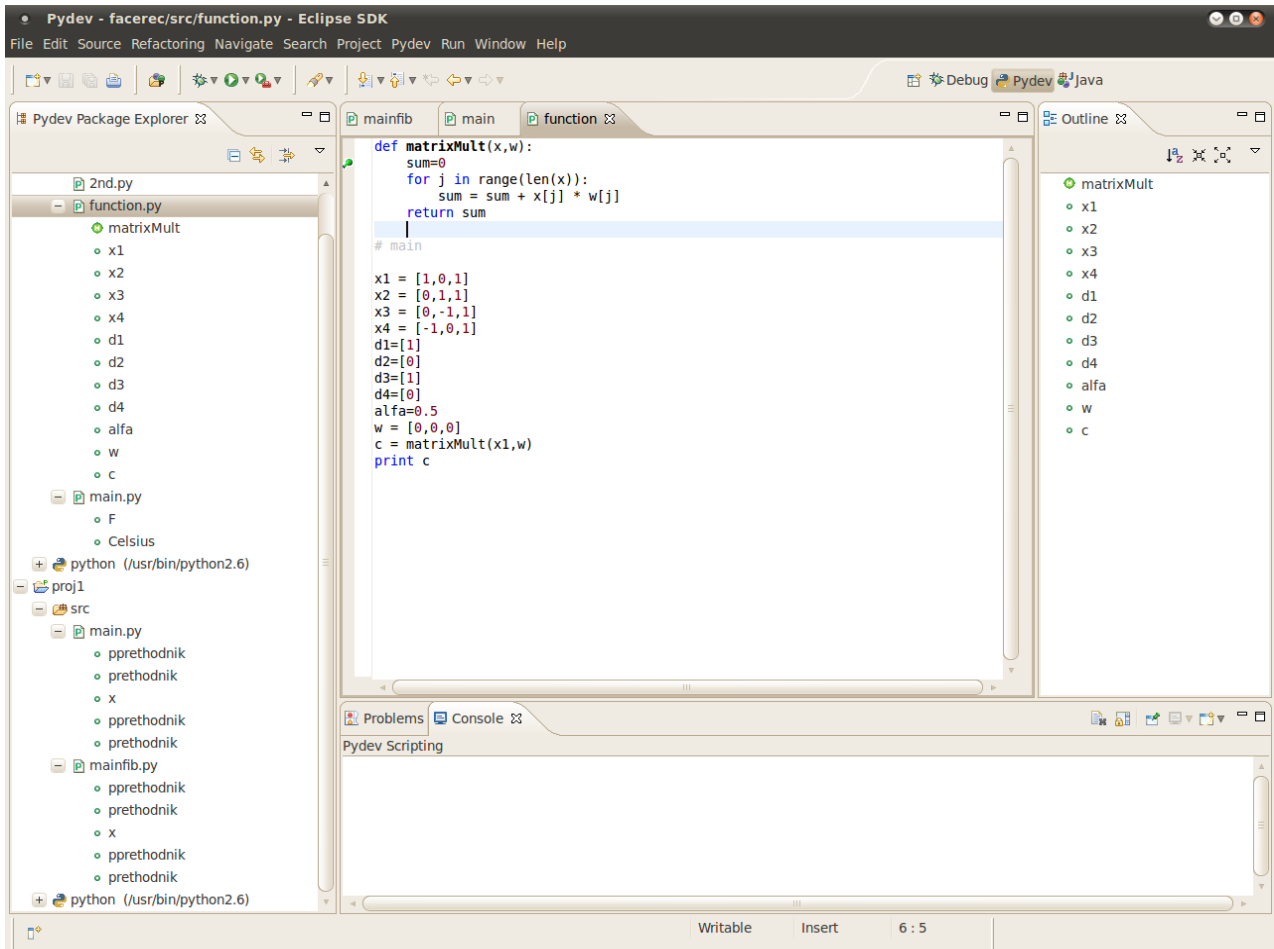


Figure 1. Screenshot of Eclipse IDE with the Pydev plugin

To get the photographs of the people, a web camera was used (over the video4linux interface) and additionally a database free for academic research of face recognition was used.

4. METHODOLOGY

After the engineering part of learning the basics of programming was over, we examined some pattern recognition theory that was necessary for face recognition.

4.1. Pattern Recognition

To discern a person from a digital photograph stored on the computer (for example using a web-cam), a series of operations need to be performed.

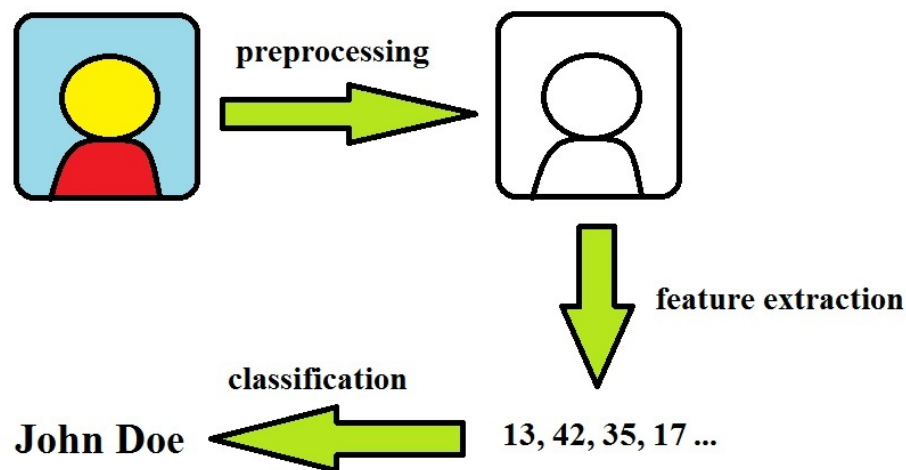


Figure 2. The pattern recognition pipeline

As we can see in figure 2, the picture is first preprocessed to better suit our needs (and be uniform in comparison to the other pictures), then [useful features are extracted](#) from the pictures (numerical values, such as the average pixel intensity, or semantic values, such as position of the person's nose inside the picture) and in the end a [classification](#) algorithm is used to determine the actual person in the photo from the features.

Many algorithms are available for each phase of pattern recognition and they greatly influence the end results.

We studied two algorithms for classification:

- [perceptron](#)
- [SVM](#) - the support vector machine,

two feature extraction methods:

- [Haar's features](#)
- [raw pixel features](#),

and several image preprocessing methods ([solarisation](#), [color spaces](#), [edge detection](#),...) in the complex Face Recognition System (FRS).

In the next few sections we will cover the methods we analysed to most depth and implemented in our FRS.

4.1.1. The perceptron algorithm

The perceptron is a [binary classifier](#) (classifies only two classes) that maps

it's input feature vector x to an output value $f(x)$, which represents a unique class. For example, in face recognition, given a feature vector for a certain photograph, it would output the person it belongs to.

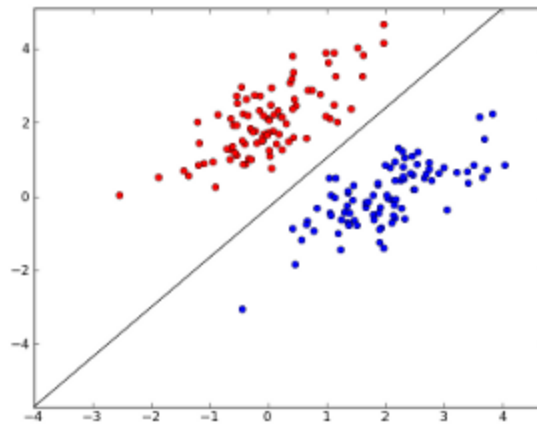


Figure 3. An illustration of a binary classifier in 2-dimensional space

Here is the perceptron algorithm in words:

For each node in output layer:

- Calculate the error, which can only take the values -1,0 and 1
- If the errors is 0, the goal has been achieved. Otherwise we adjust the weights
- Do not alter the weights from inactivated input nodes-
- Decrease the weight if the error was 1, increase it if the error was -1

For example, here is the pseudocode of the perceptron algorithm:

```

Do
{
  For All Paterns p
  {
    For All Output Nodes j
    {
      CalculateActivation (j)
      Error_j= TargetValue_j_for_Patern_p - Activation_j
      For All Input Nodes i to Output Node j
      {
        DeltaWeight=LearningConstant*Error_j*Acitvation_i
        Weight=Weight + DeltaWeight
      }
    }
  }
}
Until "error is sufficiently small" Or "Time out"

```

4.1.2. SVM

The main difference between the perceptron algorithm and the SVM is the way they separate one group of features from the other. Perceptron, being a binary classifier is able to separate only two different groups of features and it uses a linear method as shown in Figure 3.

On the other hand SVM (which is even able to separate more classes of patterns) uses transformation of patterns to a space with more dimensions to separate linearly non-separable classes, as shown in Figure 4.

Doing so, the SVM can clearly classify the input data far more precisely than the perceptron. In addition to Perceptron, SVM separates two groups of features so that the line which separates them is the furthest from both groups.

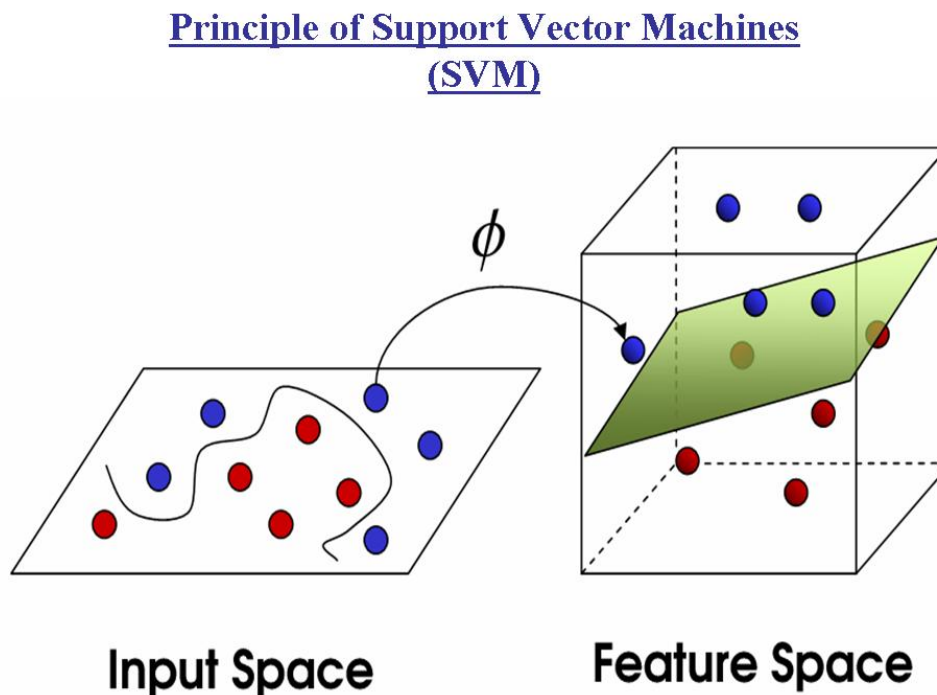


Figure 4. SVM - transforming patterns to a space with more dimensions

4.1.3. Haar's floating window

The idea here is that we don't need all the pixels in a picture, but instead

a window of some size scans the picture (we sort of take tiles of the whole picture) and calculate the difference between the left and right side of the window.

It seems that this difference (a kind of gradient information) can be semantically informative, even though the feature vector's dimensionality is much smaller (and subsequently all the following operations with the feature vector much faster).

The Python code (which is pretty understandable) that we wrote for Haar's feature extraction goes as follows:

```
def get_haar_features(self, windowSize):
    features=[]
    windowWidth, windowHeight = windowSize
    width, height = self.picture.size
    windowY=0
    while windowY<height-windowHeight:
        windowX=0
        while windowX<width-windowWidth:
            dark=0
            bright=0
            for y in range(windowY, windowY+windowHeight):
                for x in range(windowX, windowX+windowWidth):
                    luminescence=self.picture.getpixel((x,y))
                    if x < (windowX+windowWidth/2):
                        dark=dark+luminescence
                    else:
                        bright=bright+luminescence
                result=(bright-dark+float(windowWidth*windowHeight)/2*255)/
(windowWidth*windowHeight*255)
            features.append(result)
            windowX=windowX+windowWidth
        windowY=windowY+windowHeight
    self.pattern=features
    return features
```

4.2. Testing and Exploration

The scientific part of the project was calibrating parameters to get more precise data, trying out different combinations of algorithms in several iterations and looking for the best variables that will determine the actual classes. We had to determine which ones classified the pictures in the best way.

We also needed to test different features and see the outcome. The main goal

of the project was to determine which features can give the best data and make a program capable of using them to recognize a face.

After making the program itself we had to teach it to recognize a face by training it. We gave the program input data and the expected result. With a good combination of variables and iterations we taught the program to give us the right data in a big part of cases. After the training the program could repeat the same thing, but without knowing the expected results beforehand.

To sum up, here is how it works in general:

- 1. We split all the data with the determined class in 2 groups - a group for learning and a group for testing*
- 2. For every sample from the learning class:*
 - 2.1. We give the classifier samples and the corresponding classes*
 - 2.2. The classifier checks if he can classify correctly at the moment*
 - 2.3. If necessary, the classifier calibrates his parameters to correctly classify the sample*
- 3. For every sample from the testing group of samples:*
 - 3.1. We give the classifier samples, but without the corresponding class*
 - 3.2. We check the classifier results and see how many times it classified the sample correctly*
 - 3.3. We present that result with percentages*

5. RESULTS

After teaching the classifier, we tested it and calculated the statistics.

While preprocessing, we figured that finding edges is the best method, and when we used solarization the results were not very successful.

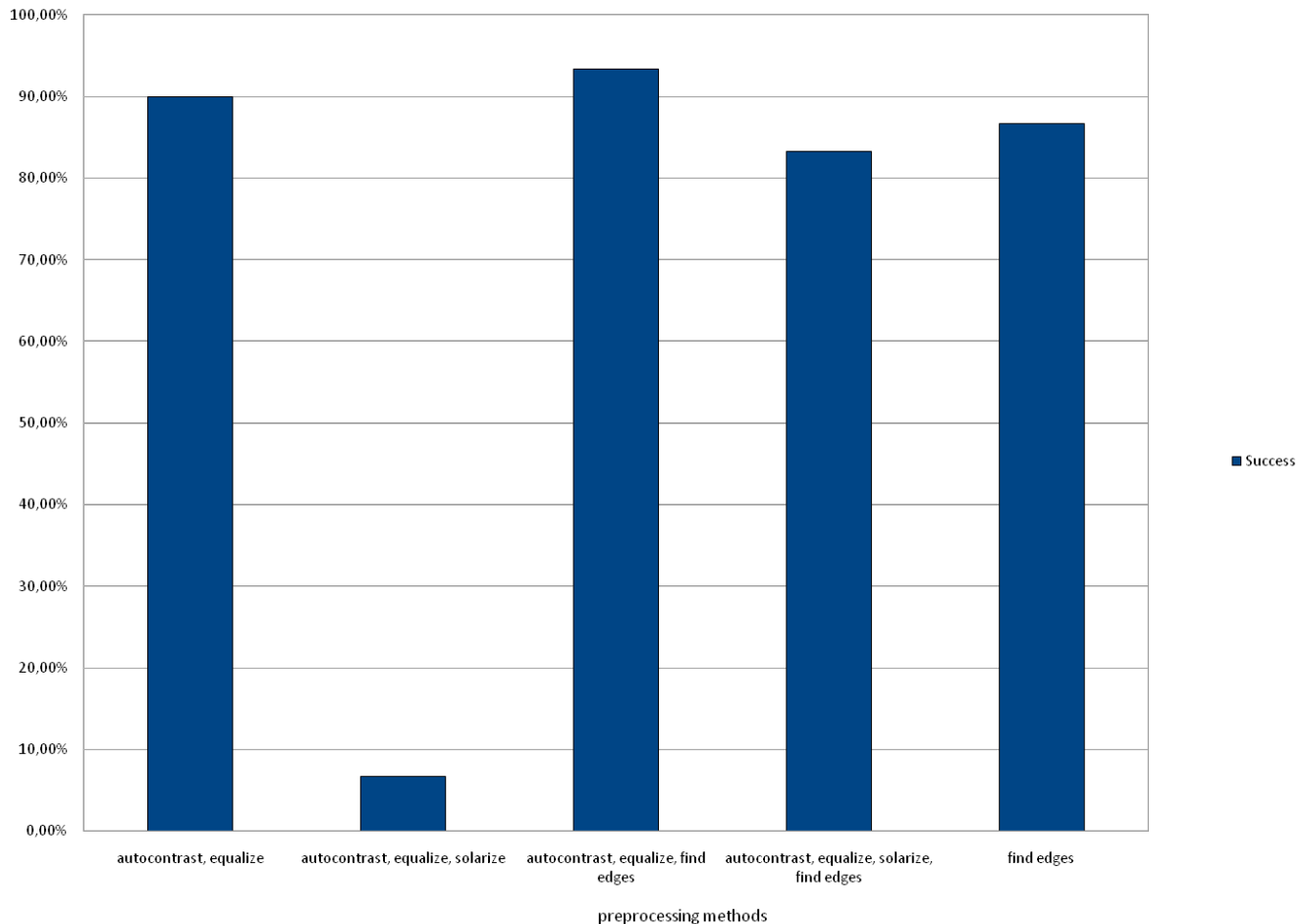


Figure 5. Comparison of different preprocessing techniques

We tested two techniques of feature extraction: raw features and Haar's feature extraction. We noticed that raw feature extraction was more successful than Haar's on bigger image resolutions, and the opposite was true on smaller resolutions.

Then we combined each of the algorithms with every feature extraction. The SVM was more successful in both cases.

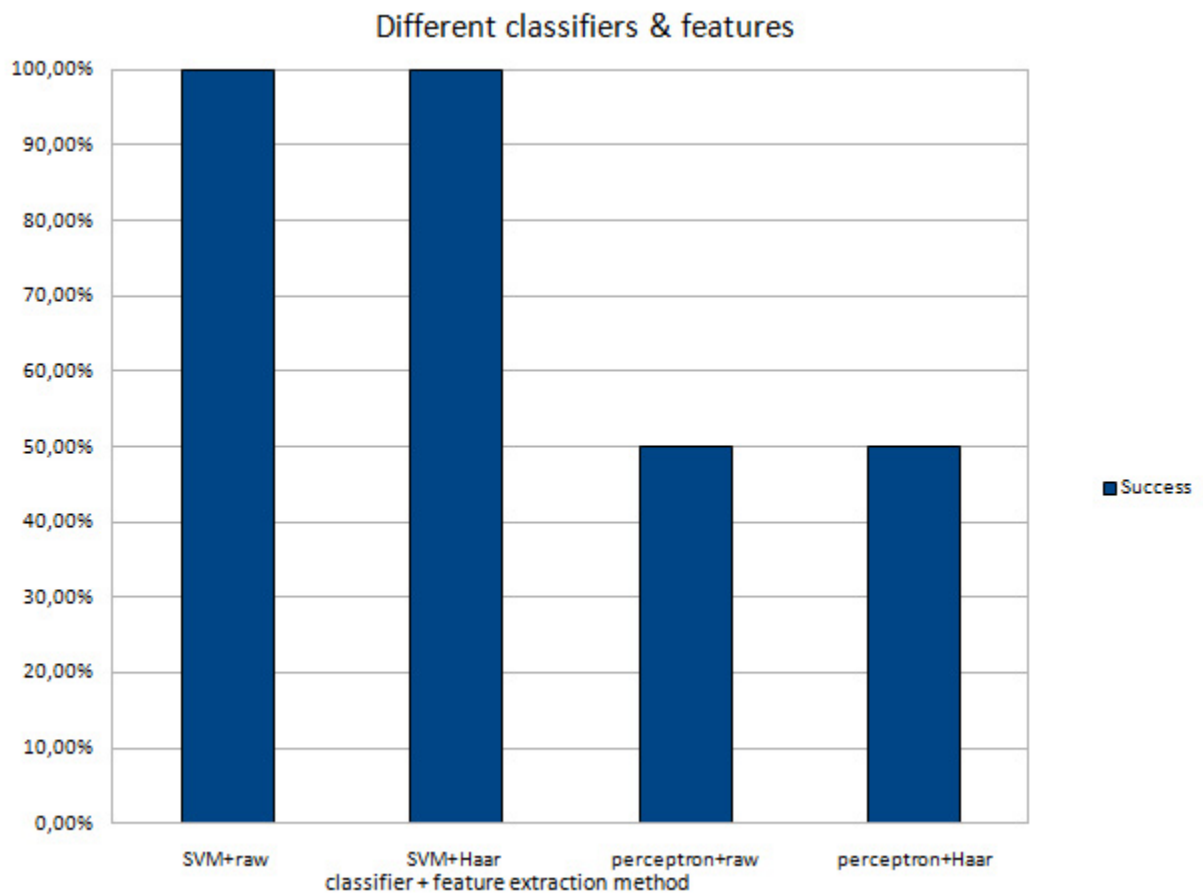


Figure 6. Comparison of different classifier and features

We tested different parameters for the SVM algorithm to get the highest quality classifier.

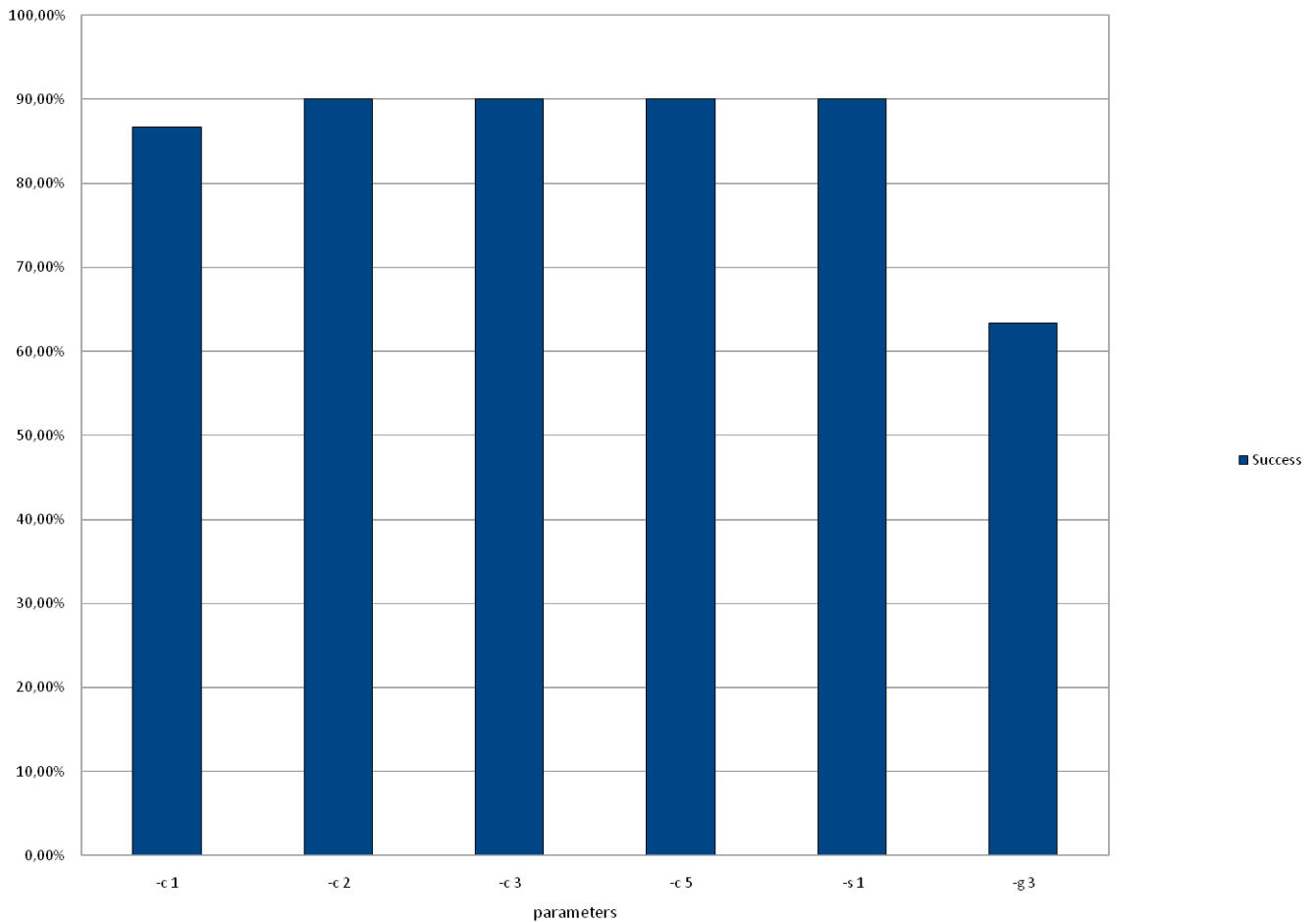


Figure 7. SVM classification parameters' influence on success

And, finally, we compared human's and computer's classification. On the bigger resolutions humans were 100% correct, while on the smaller resolutions their classification was based on guessing. On the other hand, computer made small mistakes on all resolutions but not more than 20%.

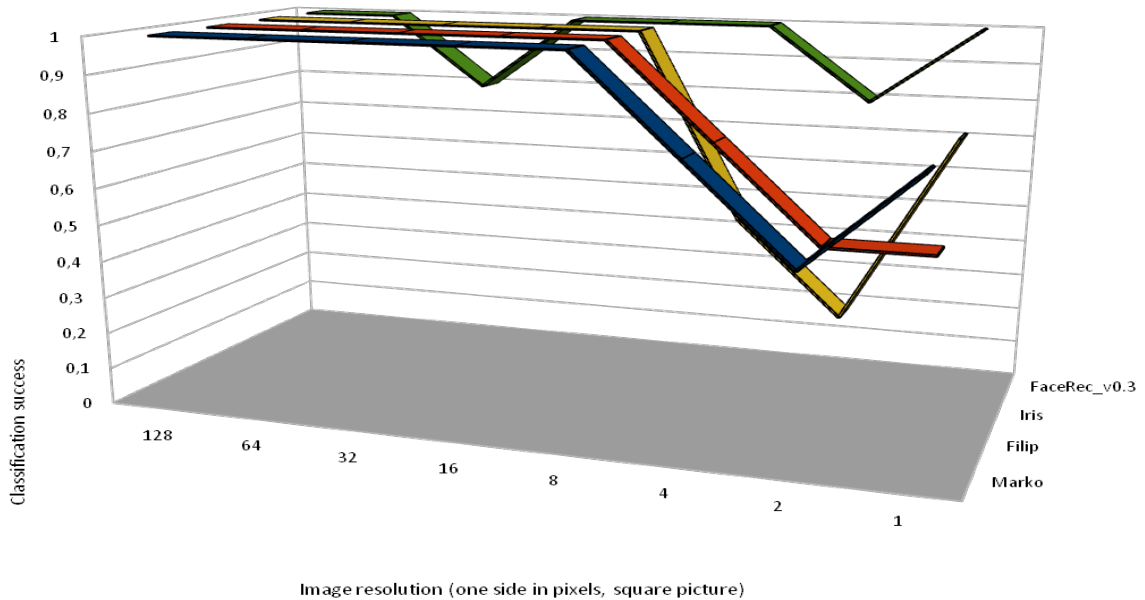


Figure 8. Human vs. computer classification

6. Conclusion

After testing our face recognition program we can conclude that:

1. Humans are better at face recognition than computers on bigger resolutions, but on the other hand computers are better at smaller resolutions.
2. Haar features extraction proved to be as much efficient as raw features extraction when comparing the results from the same classifier
3. The SVM is better than perceptron in classification of patterns.

