# Finger Detection and Classification

**David Bergés, Roser Cantenys**
Image Processing and Computer Vision (UPC)

May 15th, 2020

## 1 Introduction

In this assignment we will focus on a specific aspect of gesture recognition related to sign language. The main objective is to contribute to the recognition of numbers from 0 to 5 through the hand image analysis and the finger extraction.

The algorithm we will generate will have to process images acquired in RGB and segment them generating a partition where each finger shown by the user is represented by an individual region. The partition will also include an additional region corresponding to all elements of the image that are not fingers.

This work will be composed by the following phases:

- First of all we will look for information on ways to represent colors in images and the associated color spaces that may be relevant to the detection of areas of skin, and lately of fingers, in images.
- Secondly, we will define, program and evaluate a system which allows the identification in images of areas with skin, potentially representing hands, fingers, arms, etc.
- Finally, we will use the previous result to define a system to classify the number of fingers shown by the user.

## 2 Brief summary of the analyzed color spaces

In the following section, we are going to elaborate on some of the most widely used color spaces that may be relevant for image skin detection. This field has been broadly studied in the literature (Albiol et al., 2001; Shaik et al., 2015; Subban et al., 2013), and choices have been made according to that.

In summary, there are five major color models: CIE, RGB, YUV, HSL/HSV, and CMYK. Variants of these models taken into consideration are listed below:

- **RGB**. RGB is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. This model uses additive color mixing, because it describes what kind of light needs to be emitted to produce a given color. The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems. It is a device-dependent color model: different devices detect or reproduce a given RGB value differently, since the color elements and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time.
- **YUV**. The YUV model defines a color space in terms of one luma component Y and two chrominance components, called U (blue projection) and V (red projection) respectively. One of the advantages of luma/chroma models such as YUV is that they remain compatible with black and white systems, by simply discarding U and V components. Another advantage is that some of the information can be discarded in order to reduce bandwidth. The human eye has fairly little spatial sensitivity to color: the accuracy of the brightness information of the luminance channel has far more impact on the image detail discerned than that of the other two. So one could reduce the bandwidth of the chrominance channels considerably.

- **YCbCr**. While related, YUV and YCbCr are different formats with different scale factors. U and V are bipolar signals which can be positive or negative, and are zero for greys, whereas YCbCr usually scales all channels to either the 16–235 range or the 0–255 range, which makes Cb and Cr unsigned quantities which are 128 for greys. There exist many more variants, like YCgCr that uses the Cg component instead of the Cb, or systems like CbCr which dispenses with the luminance component.
- **HSV**. HSV (hue, saturation, value), also known as HSB (hue, saturation, brightness) is often used by artists because it is often more natural to think about a color in terms of hue and saturation than in terms of additive or subtractive color components, it is to more closely align with the way human vision perceives color-making attributes. HSV is a transformation of an RGB color space, and its components and colorimetry are relative to the RGB color space from which it was derived.
- **CIELAB**. CIELAB is a color space defined by the International Commission on Illumination (CIE) in 1976 and designed so that the same amount of numerical change in these values corresponds to roughly the same amount of visually perceived change. It expresses color as three values: L* for the lightness from black to white, a* from green to red, and b* from blue to yellow. This model is device-independent, i.e. it defines colors independently of how they are created or displayed.

## 3 Skin pixels detection using histograms

The main objective of the first algorithm is to detect skin pixels given a collection of images. In order to do so, we based our algorithm in chrominance bidimensional histograms, generating a model histogram which will be used to classify skin pixels in those images.

### 3.1 Model Architecture

Using the mask associated to each training image, we have extracted, calculated and normalized the histogram of each image to be able to add them and normalize the result in order to obtain a bidimentional model histogram which represents the bidimensional probability distribution function of skin pixels chrominance. Once we have defined the model, we transform validation images to the same color space used in training, *e.g.* YCrCb, and in pursuance of classification, hence determining if a pixel should be deduced as skin or not. We compare its chroma value to the corresponding probability of the model histogram we have generated applying a threshold as a decision rule to decide whether a pixel should be classified as skin or not.

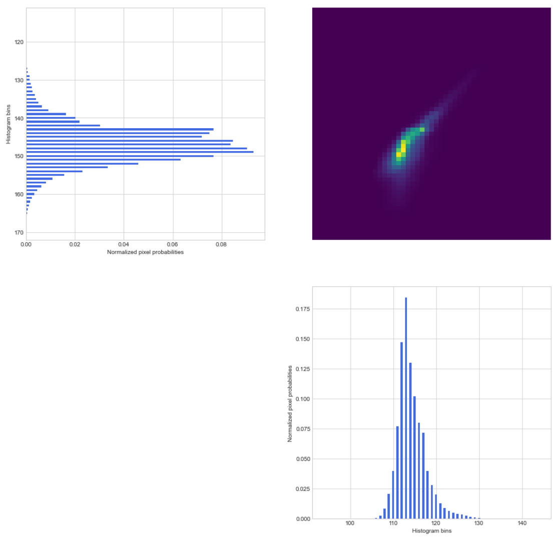

Figure 1: Bidimentional chrominance histogram.

To be able to have accurate results we optimize this model tunning the color space, the number of bins and the notion of distance or a decision rule.

### 3.2 Algorithm Parameter Optimization

The principal parameters that we want to optimize and we will study are the color space, the number of bins and the above-mentioned threshold that we use to determine the range of skin pixel values. As can be seen in Table 1, the best results are obtained using YCrCb color space, 256 bins and a threshold between 0.001 and 0.0012.

Despite YCrCb being the color space generally obtaining the better results, notice how Lab also manages to get an F1-score higher than 0.9, while contrarily the HSV color space provides poorer results all over the experiments.

On the other hand, the number of bins is closely related to the execution time and high metrics. In particular, execution times grow by a significant amount when we try to reduce the number of bins used by the algorithm, as it has to perform an extra operation for every pixel in the source image in order to determine to which bin of the model histogram that pixel belongs.

Finally, in terms of the threshold, as we observed that the maximum value is 0.0299 we decided to study the sensitivity of this parameter on the results to define its optimal value. If we choose a high value we will just capture those pixels that are in fact skin with very high probability given by the model, leaving lots of unrecognized pixels and therefore obtain low recall. On the other hand, if we choose a low value then we will get a lot of false positives so we have tried to find the balance in between these two matters. After some experiments, we concluded that the best values are located around 0.001, and more specifically 0.001 and 0.0012.

| ***color space*** | $n-bins$ | ***threshold*** | ***accuracy*** | ***precision*** | ***recall*** | $F1-score$ | ***time*** |
|---|---|---|---|---|---|---|---|
| Y | 100 | 0.0012 | 0.62 | 0.36 | 0.98 | 0.53 | 91.30 |
| Y | 150 | 0.0012 | 0.91 | 0.72 | 0.96 | 0.83 | 93.46 |
| Y | 200 | 0.001 | 0.81 | 0.54 | 0.95 | 0.69 | 81.67 |
| **Y** | **256** | **0.001** | **0.97** | **0.94** | **0.92** | **0.93** | **11.82** |
| Y | 256 | 0.0008 | 0.96 | 0.88 | 0.93 | 0.90 | 11.55 |
| **Y** | **256** | **0.0012** | **0.97** | **0.96** | **0.90** | **0.93** | **11.35** |
| H | 200 | 0.001 | 0.91 | 0.95 | 0.60 | 0.73 | 79.37 |
| H | 256 | 0.001 | 0.88 | 0.96 | 0.48 | 0.64 | 11.57 |
| H | 256 | 0.0012 | 0.87 | 0.96 | 0.42 | 0.59 | 11.09 |
| Lab | 200 | 0.001 | 0.87 | 0.63 | 0.93 | 0.75 | 81.01 |
| Lab | 256 | 0.001 | 0.95 | 0.86 | 0.89 | 0.88 | 12.75 |
| Lab | 256 | 0.0012 | 0.96 | 0.96 | 0.86 | 0.91 | 11.29 |

Table 1: Analysis on each of the parameters of our model; *color space*, *number of bins* and *threshold.* The table shows the results (*accuracy, precision, recall, F1-score and time*) of some combinations of each of the aforementioned parameters. Rows in boldface represent the best obtained combinations.

## 3.3 Post-processing and results

In order to improve the obtained results by our algorithm, we decided to post-process them using some well known morphological techniques and tooling.

After several trials, as we expected, the best results were obtained with an opening, *i.e.* an erosion followed by dilation, which is useful for removing noise, which in this case, we identify as little white objects.

This specific morphological technique requires three parameters: the shape of the structuring element, its size, and the number of iterations that we let the algorithm run. The structuring elements that we have studied are simple rectangles, ellipses and crosses. We expected that ellipses and rectangles would perform better and therefore provide better results than crosses since the main goal is to fill some little white spaces in black marked skin regions. In addition, as these white spaces that are not detected as skin are small, the structuring element size and the number of iterations should be small. In the end, the best results were obtained with a squared structuring element of size $3 \times 3$ and 3 iterations.

Taking the best combination of the model hyperparameters and applying this post-processing we achieve an F1-score of 0.94, that is, 1% higher than the best score previously obtained without post-processing. As can be seen in Figure 2 we can observe that there are some cases where we are detecting perfectly skin zones as the case of the OK hand, but there are other cases like the open hand one where the masking algorithm performs poorly. This is due to the fact that the background is close to the skin color and, in addition, nails have a different tone that is why we are not detecting them and trying to reconstruct with a morphological tool is difficult since it is a big area.

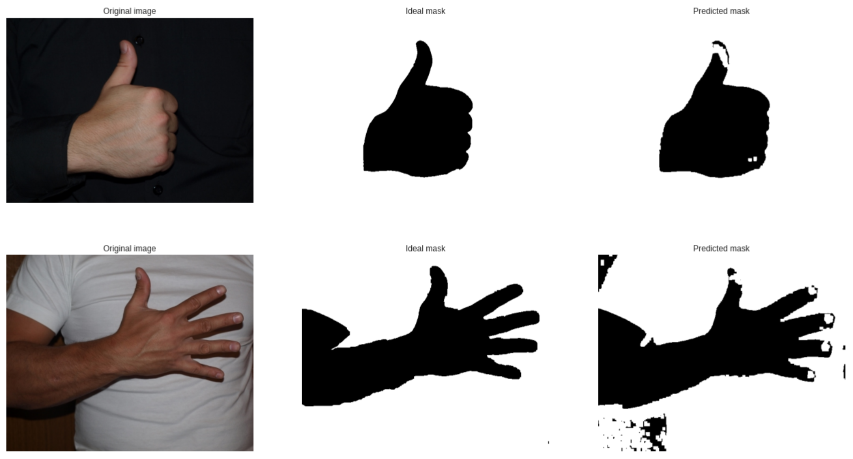

Figure 2: Representations of original image, ideal skin mask and predicted skin mask by the algorithm. In the top row, a prediction with an F1 score over 97%, and in the bottom row, a worse example with F1-score barely over 85%.

## 4 Predicting the number of exposed fingers

The main objective of the second algorithm and therefore of this section is to count the number of exposed fingers in an image collection, *i.e.* a classification problem where we want to classify each image according to the number of fingers shown by the user (from 0 to 5). In order to do so, we will take advantage of the previously generated skin masks and use them as the input for this new model.

The first idea that comes to mind when one thinks of images along with classification problems, is to use some kind of convolutional neural networks as the principal model. However, taking into consideration that we only have a total amount of 60 images for our training procedure, we expected that a simple CNN would not perform very well. Thus, we studied and implemented a completely different algorithm based in convexity hulls and compared its performance with a more classical network.

### 4.1 Convolutional Neural Network

#### 4.1.1 Model architecture

The CNN architecture we use to tackle this problem is very easy since, as we only have few images, we will overfit our model very easily.

Therefore, the architecture we propose is composed by a single convolutional block, with three input channels, 4 output channels, kernel size = 3 and paddig = 1; a batch normalization, a Rectified Linear Unit, a 2D maxpool and a final projection, a linear layer with cross entropy loss which is used as the final classifier.

#### 4.1.2 Image augmentation

To combat the problem with small training data, we propose an image augmentation approach, where we transform existing images in order to create new samples. In order to do so, we randomly select original images at a time, serving as references to apply a random number of the available transformations: *i.e* right rotation, left rotation, vertical flipping, horizontal flipping and adding noise with a Gaussian filter.

#### 4.1.3 Results and further ideas

The obtained results, as we expected, are very poor, and the network overfits on the training set very rapidly. By generating this augmented image set we obtain slightly higher accuracy on the validation set, however, as generated images are very similar to the original ones, it does not improve results as much as one could originally think, plus we do not fix the problem with overfitting at all.

In order to improve the obtained results using a neural network and using this little amount of available data, we would need to study some rather more advanced modelling techniques such as Generative Adversarial Training or transfer learning with some network trained with a similar dataset and use our data as a final fine tuning.

## 4.2 Convexity Hull Defects Algorithm

As it was expected, the previous neural network did not perform very well in terms of validation accuracy, *i.e.* it was not capable of generalizing to not seen images during training. Consequently, we did some research and implemented an algorithm that did not rely on the training observations at all.

### 4.2.1 Main algorithm definition

The main goal of our algorithm is to efficiently find, calculate and count the defects that are formed between the convex hull wrapping the hand figure, and the inter-gaps that can be seen between exposed fingers (fingers that are forming an angle).

Using the predicted skin masks generated by our first algorithm, one can define the contours of these masks and interpret them as a finite ordered set of points on a bounded subset of the Cartesian plane. In this case, the convex hull will be the smallest convex set containing it, and it may be visualized as the shape enclosed by a rubber band stretched around this subset.

From this point, the algorithm aims to find cavities inside this object, *i.e.* areas that do not belong to the object but are in fact located inside of its outer boundary. We can visualize an example in the right, where the defects would be the areas between the green figure and the blue polygon. Next, the algorithm uses some specific notion or guidance in order to use this defects to count and predict the total number of fingers shown in the image.

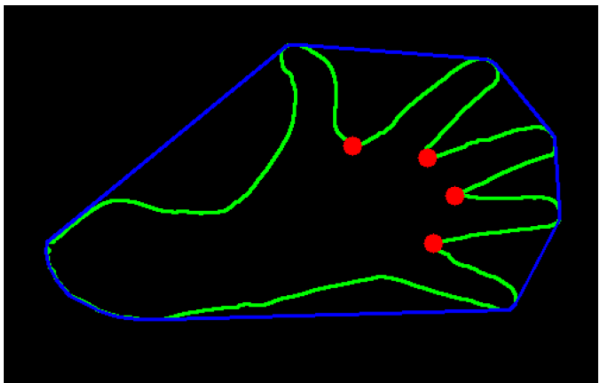
Figure 3: Representation of the ideal behaviour of the algorithm. In green, the contour drawn by the hand figure; in blue, the wrapping convex hull; in red, the cavities or defects that the algorithm considers valid.

### 4.2.2 Fine-tuning the algorithm

Of course one of the most difficult part is to define this specific notion in order to use the convexity defects such that it fits our problem. There is some literature in the Internet, and for example some people use the angles between the triangles that form the defects vertices in order to select them as valid candidates. In our case, this approach did not achieve great results, as there were many cases where for example, the angle between the thumb and the index finger was not acute. Thus, we needed to modify the standard and come up with some new conditions that fitted our dataset.

Full details of our implementation can be seen in the notebook attached in the delivery folder, but the main technique and special cases that we had to address are listed below:

- Of course, the silhouette of the skin mask is not perfect, thus the convex hull wrapping the figure resulted in many undesired small convexity defects. In order to cope with this, we filtered out this list of defects candidates and stick with the biggest ones.
- We also had to take into account that our masks did not only include hand figures, but a small part of the forearm as well. This, as can be seen in figure 3 and figures 4 and 5, can be a problem as the algorithm could detect those cavities as valid candidates. To filter these out, we noticed that all of the observations represented right-handed figures, thus we could filter defects with starting and/or endpoints located at the very left of the image.

- There were cases where the expected length or area of the defects was big enough to not be filtered by the first step, but they were very elongated with very little height. Thus we need to filter out candidates with very little height w.r.t. the biggest ones.
- Finally, after having our definitive list of defect candidates, we applied an extra post-processing in order to determine the exact number of fingers shown in every image.

#### 4.2.3 Results and further ideas

Using the previous algorithm output to predict the total number of fingers this approach managed to get an overall accuracy over 84% on the validation set. We further studied this results to observe where our algorithm was failing in its predictions, and not by surprise as can be seen in the example figures bellow, it failed because of masking procedure imprecisions. As a proof of concept, we tested our algorithm with the ideal masks available in the validation dataset, and achieved the perfect score 100% of classification accuracy.

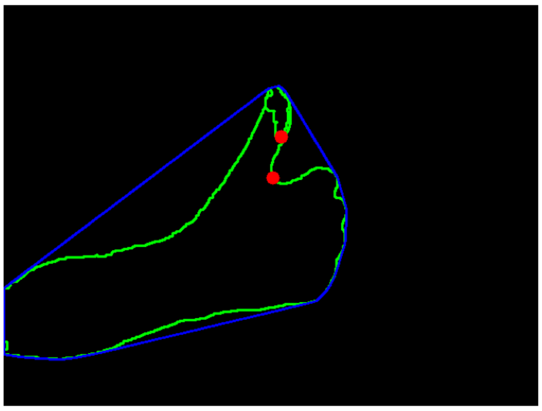
Figure 4: Incorrect prediction using the predicted masks from previous algorithm.

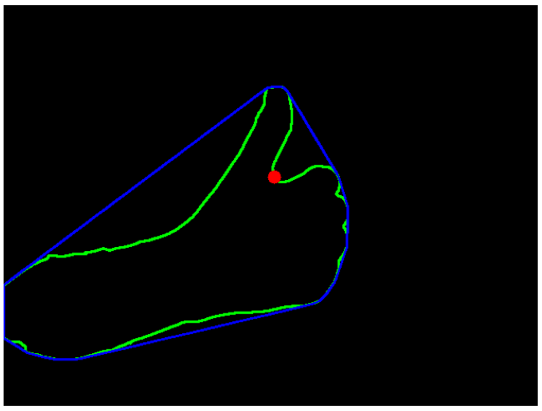
Figure 5: Corresponding correct prediction using the ideal mask.

Of course, this fine-tuning approach can be further tuned in order to achieve higher generalization for images coming from other datasets, that can be much different from these ones (left-handed for example). Plus, as we previously mentioned, we state that with higher quality masks this algorithm can achieve surprisingly good results despite its simplicity.

## 5 Conclusions

Taking into consideration the work done in the previous assignment and the rather complex models we used for semantic segmentation, we have found it very surprising that such simple histogram-based systems can perform as well in some specific use cases. More specifically, in this scenario, we discovered how skin pixels have a very well determined range of values in the chroma channels in some color spaces such as YCbCr. It is true that the available dataset had a very limited range of skin tones and light conditions. And we expect that kind of algorithms to not work so well with datasets with higher variance in those regards.

Finally, we confirmed our initial hypothesis that neural networks could not learn generalized from such small training data. In these cases, the network will try so hard to learn from the training samples which will always result in overfitting. Therefore, for these specific scenarios it could be worthwhile trying out some simpler yet more robust variants.