Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Machine learning algorithms for color image segmentation

Author: Paulo Eduardo Sampaio

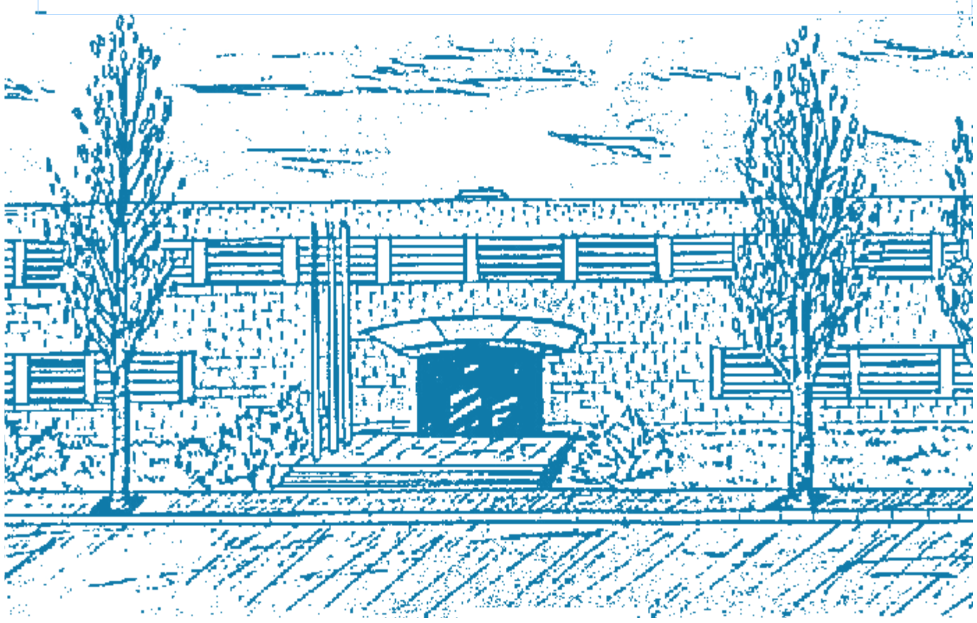
Advisor: Esteban Vegas Lozano

Co-advisor: Ferran Reverter Comes

Department: FME

University: UPC

Academic year: 2015



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

Machine learning algorithms for color image segmentation

Master thesis

Paulo Eduardo Sampaio

June 5, 2015

Advisor

Esteban Vegas Lozano

Co-advisor

Ferran Reverter Comes

Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya

Contents

List of Tables

List of Figures

Abstract

With the current advances in search engines, file compression, computer power and network speed, we have now a never seen before availability of pictures and images of the most different subjects one can imagine. But for these pictures become really useful and part of any decision making process, they need to be analyzed and evaluated, activity that usually still requires a huge amount of hours of skilled professionals.

Automated state of the art computer vision approaches are reaching incredible confidence in the activity of locating and identifying objects in images, generating bounding boxes around them. But after identifying and locating these objects, these methods do not include a frameworks to retrieve basic information from the objects detected, like color.

The objective of this work is to suggest a methodology to use simple machine learning algorithms to automatically identify regions of interest in a color picture and extract some information. As an example, the color of the region will be detected. For that, the pictures will be analyzed both pixel-wise (to identify pixels of a given color) and spatial-wise(to classify color regions inside the picture).

Keywords: Computer vision, machine learning, image processing, color image segmentation, artificial intelligence

MSC2010: 68T10, 68T45, 62H30, 62H35

Chapter 1

Introduction

1.1 Product pictures

We all have heard the expression a picture is worth a thousand words. From a business perspective, the image of a brand new car, a beautiful dress or even a perfectly ripped fruit is what wakens the desire for a new purchase. The visual perception can influence our idea of quality and change our willingness to pay. With that in mind, manufacturers hire professional photographers to take the best shot of their product assortment, in order to pass the correct message to the clients: buy our product.

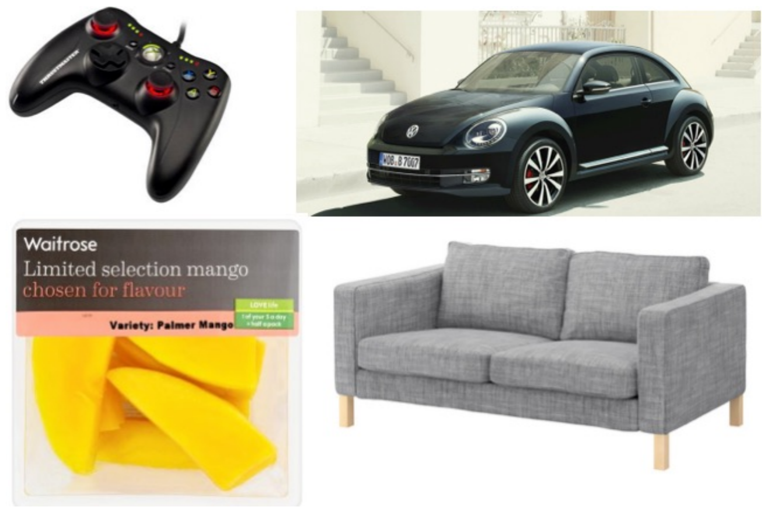
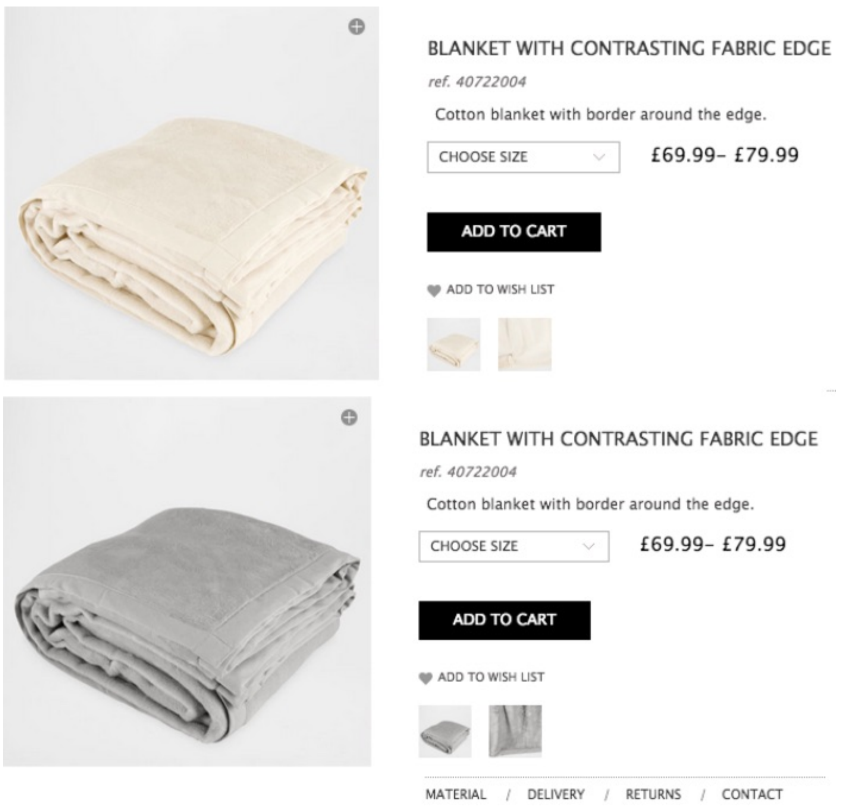


Figure 1.1: Examples of product pictures from microsoft.com, volkswagen.com, waitrose.co.uk and ikea.com

This images are distributed to different points of sale, like the company online web store, e-commerce websites that distribute their products, blogs and review webpages. Some of these websites can receive millions of product pictures every day. These products usually come with a textual description, which helps in the activity of placing the product in the right category, so the user can search and filter the results accordingly. Our problem starts to arise when the color information is not included on these textual descriptions. This can happens because:

- The product have a common description for all the color options, and only this color independent description is sent with the picture
- Different manufacturers call the same color by different names, so a website that aggregate products from different manufacturers will need to normalize the colors before using
- The textual description is inexistent or not very descriptive. Very common from products coming from retailers that are still focusing on e-commerce.


BLANKET WITH CONTRASTING FABRIC EDGE

ref. 40722004

Cotton blanket with border around the edge.

£69.99– £79.99

BLANKET WITH CONTRASTING FABRIC EDGE

ref. 40722004

Cotton blanket with border around the edge.

£69.99– £79.99

Figure 1.2: Two blankets from zarahome.com with same product name, description and even reference code, but completely different colors.

In any case, using the color information directly from the product picture is preferred because it is compatible with what the client is actually seeing in a webpage. When searching through a website filtering products of a given color, the user wants to see pictures only of that color. If products from a different color appear on the results, it is considered an error. Obtaining the color from a product picture might seem an easy task. Someone might suggest just creating a color histogram of the pixels and getting the most frequent region. This can work well for very basic product pictures, containing only the product over a white background. But even in this case, the pixel counts of the white background can surpass the pixel count of your product. So proceed by ignoring white pixels some might say. But what if you product is also white? Or if the background now is gray? The truth is: the advertising industry can be quite creative.

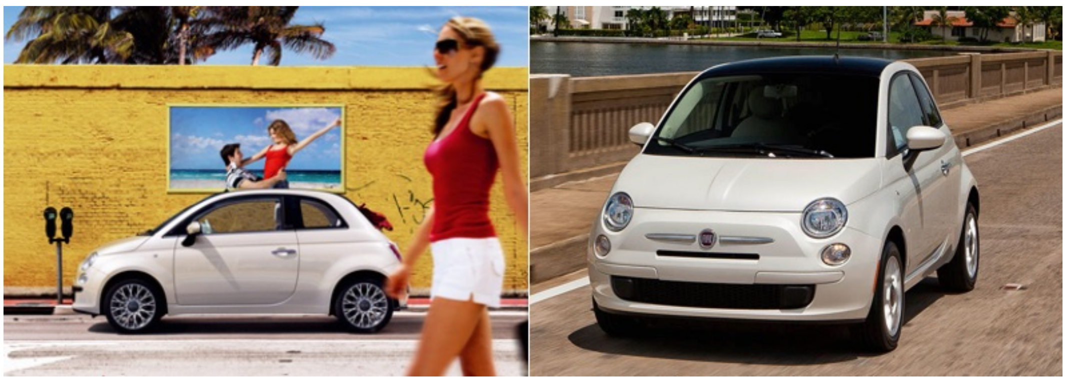


Figure 1.3: Two examples from fiat.com, product picture with complex background

In none of those counting pixels would be a good idea!

1.2 Objective

The motivation of this work is: receiving a picture from a given product, to isolate the product region and extract its colors.

1.3 Report structure

This report is divided in four main parts: literature review, methodology, results and conclusions.

On the literature review, popular approaches to the task of segmenting color images will be explored. Time will be dedicated to discuss color spaces, then extensions of grayscale image analysis and finally modern approaches.

Then on the methodology, a complete framework will be proposed to receive a color image, identify its interesting part, isolate it and read its color. Metrics to evaluate the results will be discussed.

The framework proposed on the methodology part will be put to test in a real dataset of pictures of dresses. Each step of the framework will be applied to the dataset and the results will be evaluated according to the metrics proposed on the methodology section.

Finally, on the last part the conclusions will be discussed, alongside with possibilities of future improvements.

Chapter 2

Literature review

2.1 Introduction

Literature does not have a closed solution for the color image segmentation problem, specially because a segment limit can be seen as a psychophysical perception, not susceptible to a purely analytical solution [1]. In any case, the field of computer vision has gone through incredible advances in the last couple of decade.

2.2 Color spaces

It is generally accepted that the first decision should be regarding color space selection [1, 2, 3, 4, 5, 6, 7, 8, 9]. The original color space is in which digital images are stored is RGB, a three channel (Red, Green, Blue) system that tries to mimic humans trichromatic vision in an additive way. Each channel represents the intensity of light in that basic color. Any colors can be achieved as a unique summation of the three components, going from black (the absence of light on the three components) to white (maximum of light in the three components). The problem with this representation is that, although simple and straight forward, the three channels are highly correlated and there is no separation between luminance and chrominance, which means that the perception of color is dependent on the light [1, 9].

To avoid these issues, researches apply linear and non-linear transformations to pass the pixels in RGB to other color spaces. Most usual transformations are to the non linear transformation to the perceptual color spaces (HSV family) or to the colorimetric color spaces (CIE family) [9, 10]. The perceptual color spaces (HSV) are the ones where the actual color information is isolated in the Hue and Saturation channels and the light information is on the Value channels [1]. That way, color is independent of the light. Other variations of the perceptual color spaces are HSB (Hue Saturation Brightness) and HSI (Hue Saturation Intensity). The Colorimetric color spaces comes from the Commission Internationale de l'Eclairage (CIE), and relies in some external parameters, related to the lightness and to the field of view of observer, in order to replicate the effect caused by perceiving the colors from different direction and under different lights (white light, day light, outside sun light, etc) [11]. For the color image segmentation task, the importance of this method is that it tries to achieve perceptual uniformity to a human observer, meaning that color differences can be calculated using a simple euclidean distance function [1, 9, 10, 11]. Unfortunately, it does achieve it at the cost of computationally expensive transformation [1, 10, 9, 11]. Both HSV and CIE transformations will be detailed in the methodology section. After choosing a proper color space, literature suggests three main approaches to color segmentation: histogram thresholding, edge detection and feature space clustering [1, 2, 3, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

2.3 Histogram thresholding

Histogram thresholding is a technique of image segmentation that assumes that the histogram of pixel intensities of a given image can be separates into groups of peaks (modes), and this peaks represents different regions of the image. [3] It is a technique very useful for grayscale images, since the histogram is already unidimensional (grayscale images only have one color channel). Although unidimensional, the histogram is definitely not unimodal, and the first challenges of this methods are to select which modes are relevant and the size of the relevant peak bucket. An example can be seen on figure 2.1.

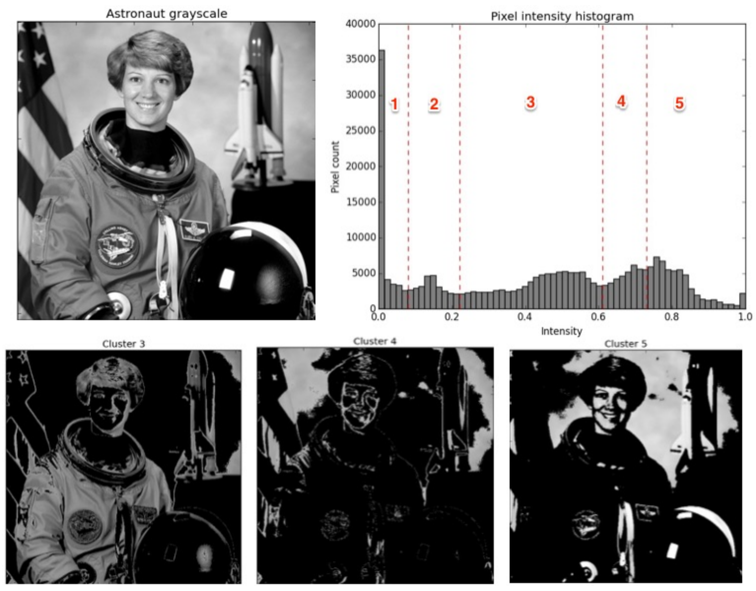


Figure 2.1: Astronaut picture from the python library skimage and respective histogram. Dashed lines mark a naive histogram segmentation. Right below, the three more significant clusters. Notice how cluster 3 seems to isolate the astronaut suit, cluster 4 details on the flag and cluster 5 the astronaut face and the wall behind her.

An usual approach is the recursive methods [18, 19, 20], where a threshold is set to identify a peak, for instance using the mean value of pixel intensity. Pixels on the histogram bins above the threshold (peaks) are separated from the pixels on the histogram bins below the threshold, generating new images. This process is repeated to all new images generated until no new significant peaks are found. A schematic is provided on figure 2.2.

The decision regarding if a peak is significant or not can be done using PSNR (Peak Signal to Noise Ratio) [20]. The main criticism of this method is that it only applies to simple images. More complex ones, like natural scenes, or heavily textured images, generate to many small meaningless regions. [12]

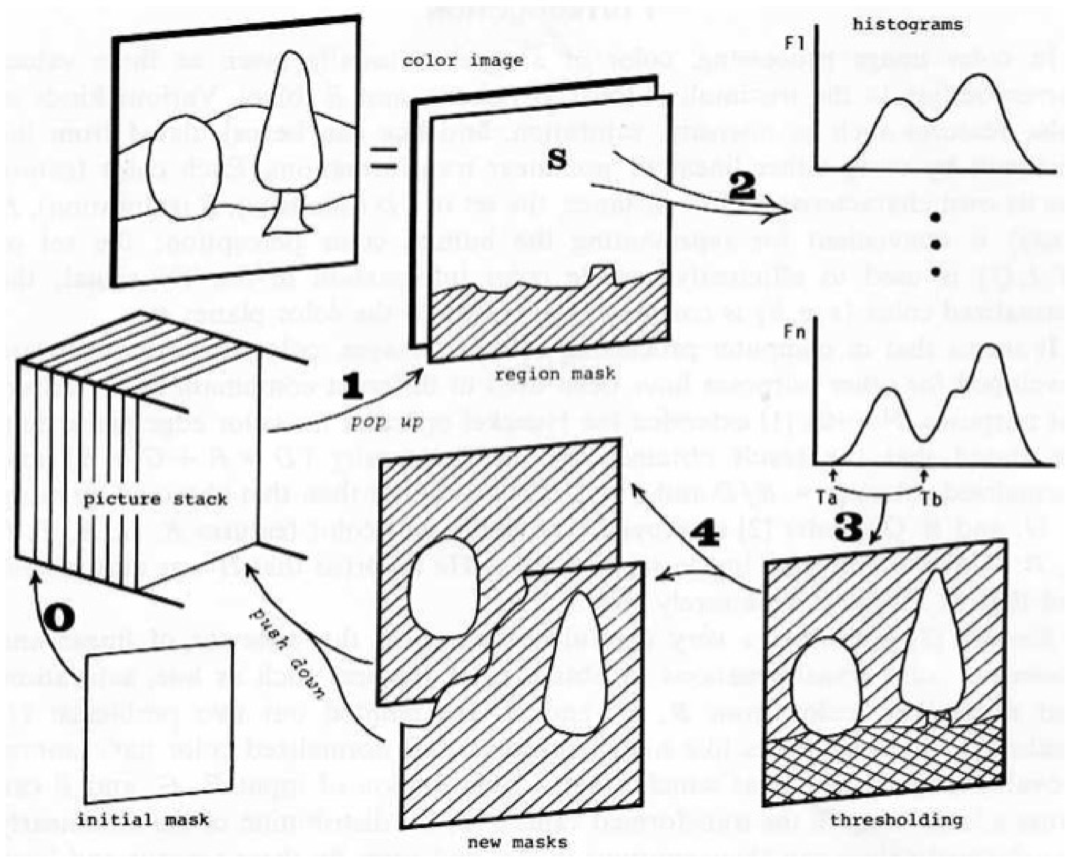


Figure 2.2: Recursive histogram threshold method from [18].

2.4 Edge detection

Edge and boundary detection are also fairly popular methods [1, 12, 13, 14, 15, 16, 21, 22, 23, 24]. The idea is to identify groups of pixels that have an abrupt change in their intensity [23]. These sharp transitions are considered edges, separating different regions of the image. There are two main approaches to this task, using differential operators and using convolutions with Gaussian operators.

On the first approach, horizontal and vertical differential operators are used to calculate the gradient of each pixel. These operators are a square pixel masks that calculates an approximation of the first derivative in both directions. These approximations are then aggregated into the gradient intensity value. A new image is usually created with the gradient intensity of each pixel. The points where the gradient achieve extreme values, represents the sharp transition in that pixel region, and therefore an edge [22, 23]. The horizontal and vertical Sobel operators are:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad and \quad G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

In G_y, the abruptly change in the vertical direction, therefore it locates and horizontal edge. In G_x, the values change rapidly in the horizontal direction and therefore there is a vertical edge.

The gradient intensity G at each point can be calculated by:

$$G = \sqrt{G_x^2 + G_y^2}$$

As an example, applying these operators in a checkerboard, where the edges are well defined produces the following outputs:

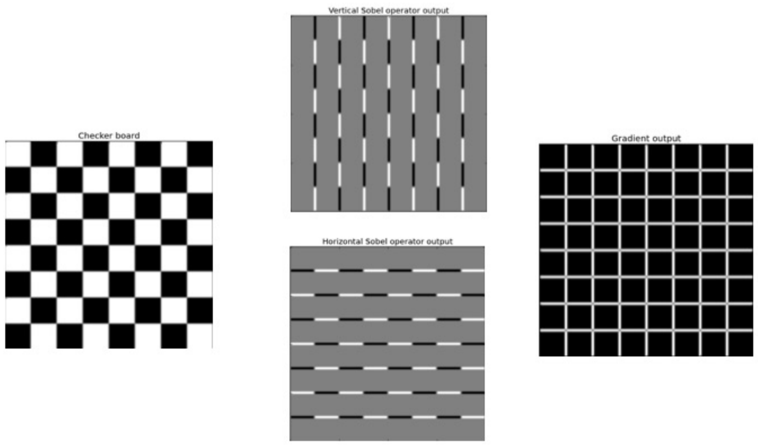

Checker board
Vertical Sobel operator output
Horizontal Sobel operator output
Gradient output

Figure 2.3: Original checker board picture from skimage python library, the outputs of the vertical and horizontal Sobel operators and the gradient intensity output.

A threshold value is used to remove noise and not significant transitions. [1, 12, 22, 23]. Main criticisms of the differential operator methods are due to the fact that it actually do not specifically detect edges. It detect transitions, not specific lines that separate bounded regions. Also, as the filters are usually small (3×3 or 5×5 matrices), they are susceptible to noise (identifying false edges) and at the same time insufficient to detect gradual transitions (ignoring possible edges) [1, 12, 21].

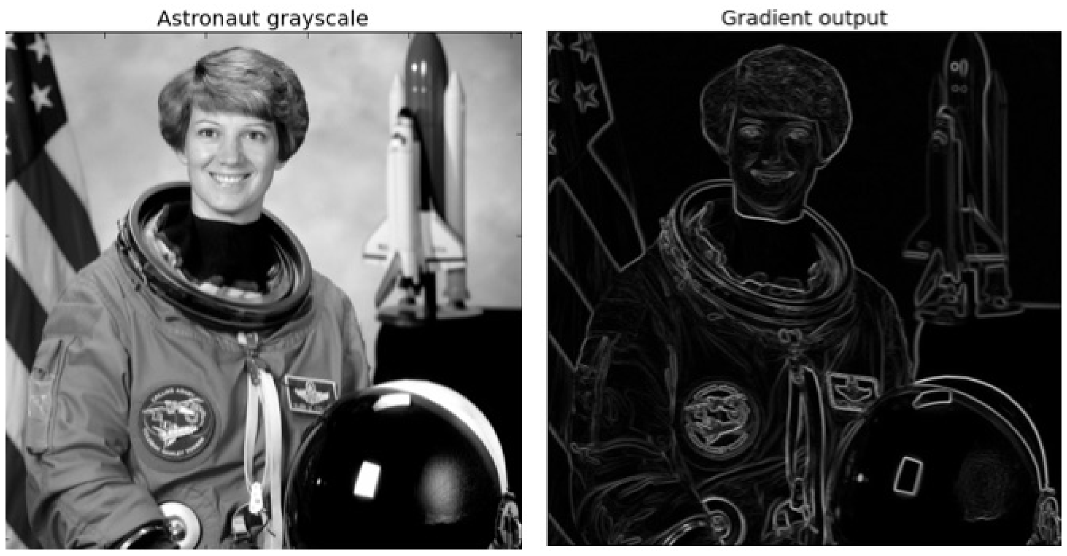


Figure 2.4: Original astronaut picture and sober filter output. Notice how most edges are well detected, but soft transitions like the one from the flag to the astronaut suit or the ones from the white background to the miniature spaceship were not detected.

To deal with this criticism, methods that use convolutions with Gaussian operators were developed to decrease the error rate on edge identification and to better locate the identified edge on the center of the real edge, as a unique line. The most popular one is the Canny filter [21], that assumes that the image can be considered as the addition of real edges and white gaussian noise. That way, it tries to find the best local filter to convolute with the image in a way that the noise is filtered out and the maximum points of this convolutions are the edge.

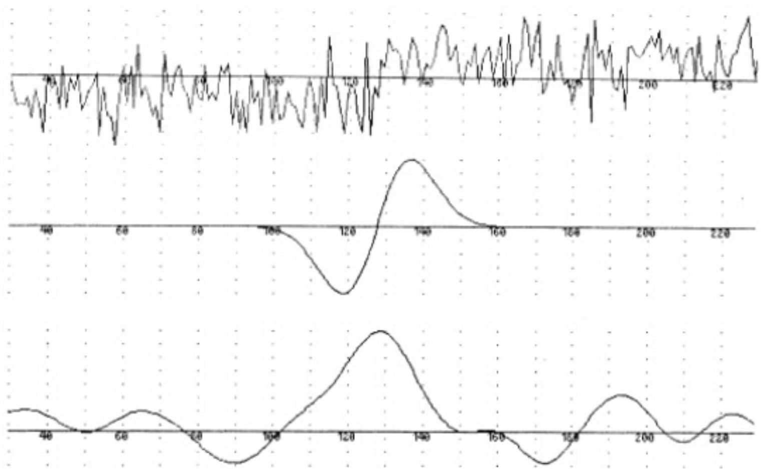


Figure 2.5: Convolution with gaussian operator example. On the first line we see an edge embedded in gaussian noise. On the second line, a simple gaussian filter an finally, on the third line, the result of the convolution. The maximum of this output is an edge.

The filter can be parametrized using the standard deviation of the gaussian distribution. Optimizing this parameters leads to a better detection without false or insignificant edges.

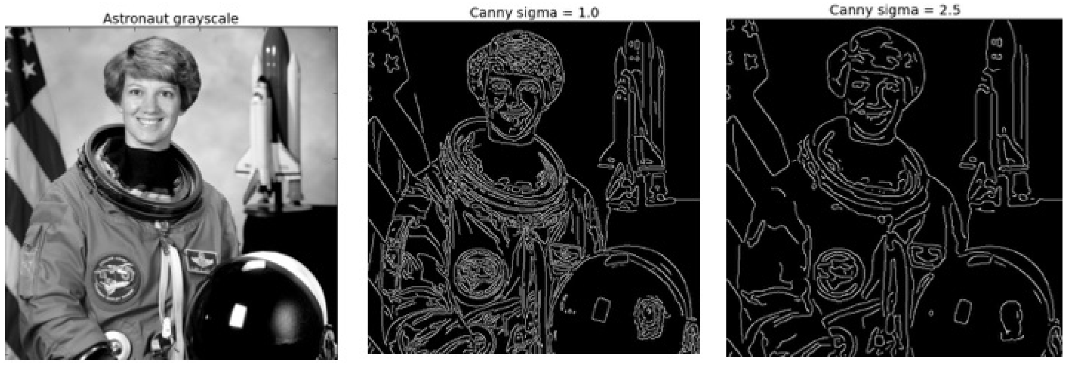


Figure 2.6: The astronaut image with a Canny filter using a gaussian filter of standard deviation of 1.0 and of 2.5.

The idea of convolute the image with a gaussian filter is one of the basic ideas of modern approaches in computer vision to object identification, like deep convolutional neural networks. In this method, from an original image, feature maps are generated by convolution with small kernels. These convolutions can happen in different layers, with different kernels, usually followed by a max-pooling subsampling operation that reduces the dimensionality by taking only the maximum value of each 2×2 pixel grid of the feature map. These are the used as input for another convolutional layer or as input for a neural network. [26]

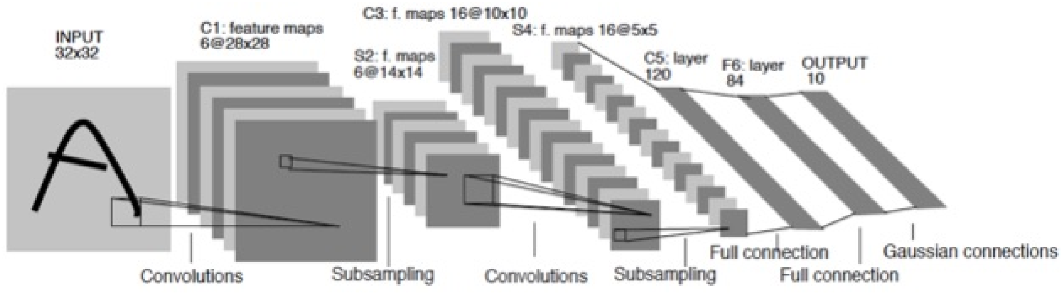


Figure 2.7: Architecture of a convolutional neural network [26].

2.5 Color clustering

Both edge detection and histogram threshold are models designed to work with one channels intensity (usually grayscale levels) that can be extended to multiple color channels. Pixel clustering approaches, on the other hand, consider a pixel as an observation in any n-dimensional space, n being the number of channels. Colors add information to the process and the computational power needed to process additional dimensions are now more accessible then ever [1]. In that sense, hard partitioning algorithms like k-means are popular choices for the task. The classic k-means implementation starts with a know number of clusters an initial guess of their centers. Then, each pixel is allocated to the cluster with the nearest center, based on euclidean distance, and a compactness function is calculated. The cluster centers are then re calculated and, since the position of the centers now changed, the pixels are once again allocated to the cluster that have the nearest center and the compactness function is calculated again. This process continues until a stop criteria is met. This stop criteria can be a number of iterations, a threshold of compactness or convergence to a stable solution.[1, 12, 17, 27, 28, 29, 30, 31, 32].

The shortcomings of the k-means methods are the need to set beforehand the number of clusters and the fact that the outcome can be a consequence of the initial configuration, optimizing to a local minimum. It is important to notice that, even though these publications were able to segment the image in different regions, they did not provide a framework to categorize the cluster into interesting or not interesting parts of the image.

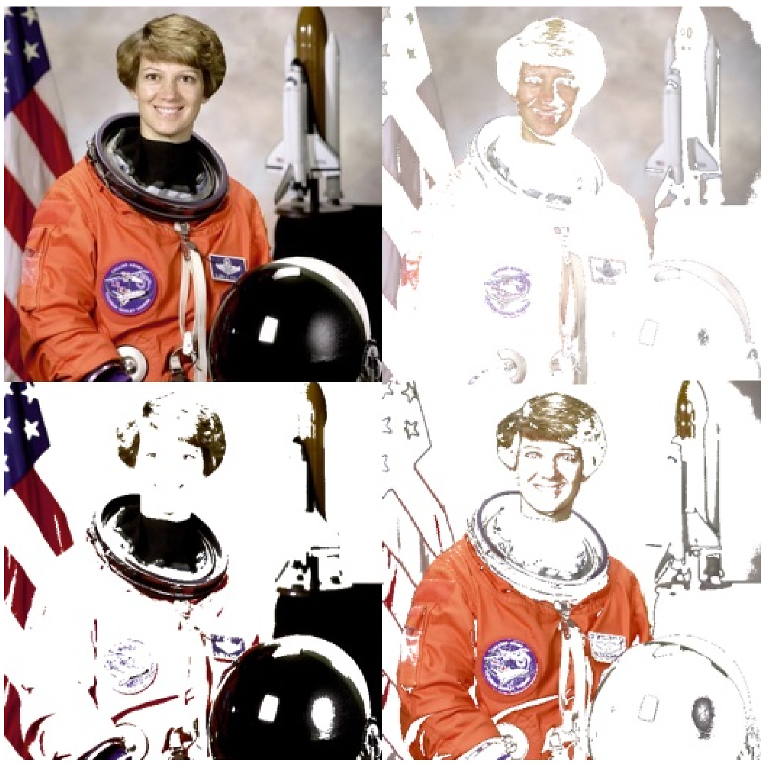


Figure 2.8: Original color astronaut decomposed in only 3 clusters. Notice how one cluster was able to aggregate all white/beige colors, other the darker colors of the flag and black objects and, finally, the last clusters was able to isolate the astronaut suit, even with just three clusters.

Chapter 3

Methodology

3.1 The framework

Following our objective to create a framework to segment and identify important parts of the picture, we propose the following flow chart:

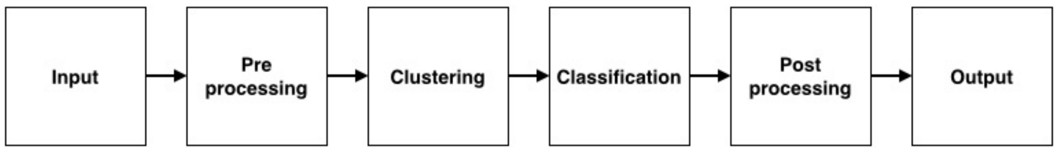


Figure 3.1: Proposed framework for region segmentation and classification

On the next sections each part of the proposed framework will be discussed in further detail, with possible approaches that will be then compared in the results chapter.

3.2 Input

The input to our frameworks will be commercial product pictures save in 24 bit RGB color, in the popular .jpg format. The color 24 bit RGB color representation means that the picture can be represented as the overlaying of three two dimensional arrays, representing the red, green and blue light. In each one of these two dimensional arrays, each cell (pixel) has an 8 bit integer numbers (from 0 to 255) that represents the light intensity in that point. These integers are transformed to float values from 0 to 1 by dividing the original integer by 255.

This means that in this model, using three 8 bit channels, the maximum number of colors that can be achieved is around 16 millions, which is know as True Color mode.

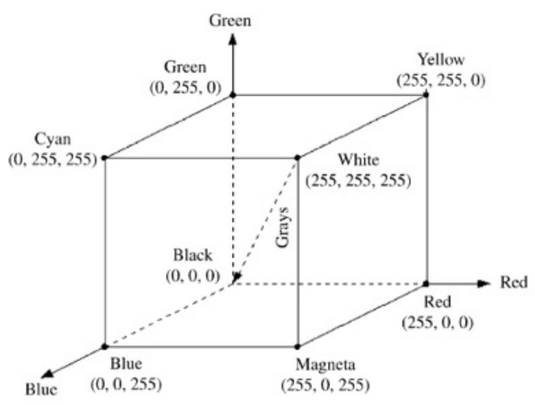
General guidelines for product pictures can be seen in major e-commerce websites, like Amazon (Amazon's product image requirements) and e-Bay (ebay's product pictures standards). In general they have a minimum size to be displayed in a high resolution display. This minimum requirements tend to change as display technology evolve.

3.3 Pre-processing

The first step in the pre-processing is to resize all the images to a fixed size. This step is done for two main reasons: for the pictures of different products to be comparable pixel by pixel and to reduce dimensionality. The decision on picture size is empirical. It needs to be large enough for the product to be clear and differentiable among different pictures and small enough for the implementation to be commercially acceptable. This can largely changes from dataset to dataset. As an initial input, currently 150×150 pixels is a reasonable recommendation both in the sense of product clarity and algorithm speed, as well se in the results. Also in the pre-processing phase is the selection of color space. As seen in the literature review, the main color spaces recommended are HSV and CIE Luv.

3.3.1 HSV color space

The HSV transformation can be seen as passing the pixels from a 3 dimensional cube (RGB space) to a cylinder, where the Hue channel goes from $0°$ to $360°$ isolating the perceived color, passing from red to yellow, green and blue. Saturation values represent the radial distance from the center of the cylinder. Ranging from 0 to 1, the closest to the center, the lighter the color. Finally, the Value is the height of the cylinder. Also ranging from 0 to 1, it represents the gray level. Lower values are closer to black, higher values are closer to white [1].



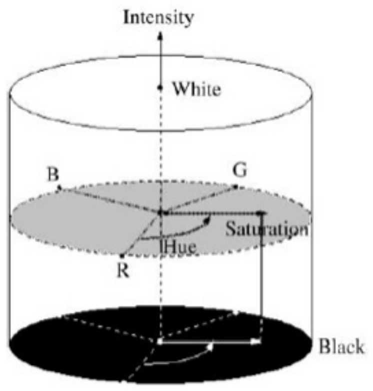


Figure 3.2: Original RGB cube and HSV cylinder (here represented as HSI) [1]

The full transformation from RGB to HSV is obtained by:

$$V = max(R, G, B), \quad S = \frac{V - min(R, G, B)}{V}$$

Then, we proceed by calculating H as [33]:

$$H = \begin{cases} \frac{G-B}{6S} & \text{, if } V = R, \\ \frac{1}{3} + \frac{B-R}{6S} & \text{, if } V = G, \\ \frac{2}{3} + \frac{R-G}{S} & \text{, if } V = B \end{cases}$$

Notice how both H and S channels are undefined for pixels where originally R, G and B are 0. On those cases, for practical reasons, H and S are considered as 0.

3.3.2 CIELuv color space

As seen in the literature review [1, 9, 10], the CIE Luv space tries to achieve a uniform perception of the colors, meaning that color differences can be calculated and compared in terms of euclidean distances. This is particularly interesting for clustering algorithms. The shortcoming of this method is that the transformation is more computationally intensive than the HSV. Also, it depends on the selection of a illuminant and and observer type, to guarantee that the colors displayed are the same for different observers positions and under different lights. As for our framework we are only interested in the uniform perception characteristic of this transformation, we will choose a standard illumination known as D65 (daylight) and also a standard observer point of view known as 2° standard observer. Both parameters are provided by CIE (Comission International de l'Eclairage)

The transformation happens in two steps. The first one is to the CIE XYZ color space, which is the base for other CIE spaces. This transformation is done by (considering D65 and 2° standard observer) [1, 34]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.955 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Then, having the XYZ calculated, when can proceed to the CIELuv calculation [1, 34]:

$$var_U = \frac{4X}{X + 15Y + 3Z}, \quad var_V = \frac{9Y}{X + 15Y + 3Z}, var_Y = \begin{cases} \frac{Y}{100}^{\frac{1}{3}}, & \text{if } \frac{Y}{100} > .008856, \\ \frac{7{,}787Y}{100} + \frac{16}{116}, & otherwise \end{cases}$$

From the tables of D65 and 2°standard observer, we have [34]:

$$ref_X = 95.047, \quad ref_Y = 100.000, \quad ref_Z = 108.883$$

And we can calculate:

$$ref_U = \frac{4ref_X}{ref_X + 15ref_Y + 3ref_Z}, \quad ref_V = \frac{9ref_Y}{ref_X + 15ref_Y + 3ref_Z}$$

And finally:

$$L = 116var_Y - 16, \quad u = 13L(var_U - ref_U), \quad v = 13L(var_V - ref_V)$$

3.3.3 Metrics

The metrics analyzed to choose between HSV or CIELuv will be correlation among channels and processing time. These values will be analyzed in the results chapter. The intuition behind the selection of these metrics are easy to see:

- the more independent the channels are, more information will be given for the cluster algorithm to be able to segment the regions.
- The transformation need to be fast enough to process a large number of images by day
- The correlation among the channels are specific to each image, so it changes from dataset to dataset.

3.4 Clustering

For the clustering process, k-means will be used. The most popular implementation is the one defined by Lloyd, that can be formalized as [30, 31]:

- Let X be $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where each $\mathbf{x}_i$ is a vector in R^m space
- Let C be $C = \{c_1, c_2, \ldots, c_k\}$ a partition of X into k disjoint nonempty subsets, known as clusters, and $n(c_i)$ the number of elements of the cluster c_i
- Let $\mathbf{v}_{c_i}$ be the centroid of cluster c_i, calculated as $\mathbf{v}_{c_i} = \frac{1}{n(c_i)} * \sum_{j \in c_i} \mathbf{x}_j$
- Let WSS be the within cluster sum of squares, the "k-means cost", be calculated as [31] $WSS = \sum_{i=1}^{k} \sum_{\mathbf{x} \in c_i} (\mathbf{x} - \mathbf{v}_{c_i})^2$

The objective is to find a cluster configuration C of k clusters that minimize the WSS. This is done by a two step algorithm [30, 31]:

- Assignment step: Allocate each $\mathbf{x}_i$ to the cluster c_i with nearest centroid, based on euclidean distance
- Update step: Recalculate the centroids based on the new configuration

These steps are repeated until a stable solution is achieved or a stop criteria is met. It is important to notice that this stable solution not necessarily is the global minimum. It is an outcome of the selection of the initial centroids and most likely will sub optimal. To avoid the use bad solution, one strategy is to run this algorithm several times with different initial centroids and use the final C configuration that yields the smallest WSS. In any case, to select the initial centroids location, the k-means++ initialization will be used. The idea is [35, 36]:

- Select the first centroid $\mathbf{v}_{c_1}$ as a point of X, randomly and uniformly distributed
- For every other centroid $\mathbf{v}_{c_i}$, select a random point of X but with probability distribution of $\frac{d(\mathbf{x}_i)^2}{\sum d(\mathbf{x}_i)^2}$, where $d(\mathbf{x})$ is the shortest distance from $\mathbf{x}$ to the centroids already selected.

That way, the points near to the centroids already selected are penalized, spreading the initial configuration across the dataset. Another decision that need to be made is regarding the number of clusters k. This is completely dependent of the picture that is being analyzed. An approach to help the decision is to understand how the WSS changes with the increase of number of clusters. When the WSS reaches a threshold or the marginal gain stops to be significative, there is no reason to increase the number of clusters.

To be able to apply the k-means to a picture in HSV or CIELuv color space, we can consider that each color pixel is an observation in a tridimensional space. That way, a picture can be seen as $P = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ldots, \mathbf{p}_i\} \in R^3$, where P is the picture and $\mathbf{p}_i$ is a pixel.

The output of the clustering process is a two dimensional array with the same height and width of the original picture, and in each cell it receives the cluster number C_i to which that pixel is assigned. This array can be them transformed in k binary arrays with ones in the current cluster and zero for all the others. This will generate k binary arrays, each one with the silhouette of the cluster $c_i, i \in \{1, 2, \ldots, k\}$, and each one of these silhouettes now needs to be classified into part of a product or not.

3.5 Classification

Now that our original picture is divided into k different silhouette arrays, we need to identify which of these arrays are significant for our task and which ones are not. For that, we will test two supervised machine learning algorithms: k-nearest neighbors and

Support Vector Machines. A supervised algorithm is the one that learns from labeled data. For that, a data set with correctly classified examples is needed. This means that someone needs to go through some of the outputs of the clustering process and classify them as part of a product or not.

3.5.1 Vectorizing the dataset

The input of the machine learning algorithms will be a vectorized version of the array. This can be done simply by concatenating each row of the array one after the other. That way, a two dimensional array $m \times n$ will generate a vector of dimensionality $m * n$. Each silhouette will be an observation in a $m * n$ dimensional space. Here is where the selection of picture size, done during the pre-processing phase, starts to impact, since even a small 150×150 pixel picture transforms into a 22500 dimensional vector. This is not necessarily a problem, it is just a warning that the dimensionality grows quadratically as we increase the input picture dimension.

3.5.2 Modeling strategy

For any of the machine learning algorithms that will be explored, the modeling scheme is the same: The full dataset will be divided into two smaller sets:

- $\frac{2}{3}$ as training set, the subset used to create the predictive model
- $\frac{1}{3}$ as test set, the subset used to test the quality of the predictive model
- Cross-validation to to asses generalization
- Grid search for parameter selection

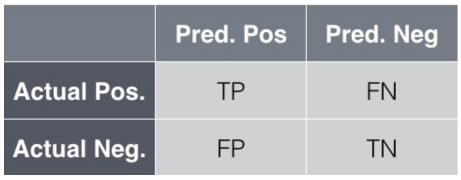
  A grid with different parameters values is created

  A model is trained and tested for each parameter value on the grid

  The parameter with which the model yields better results is used

- Confusion matrix and F1 score for measuring results [37]:

Confusion matrix is a table with actual and predicted classes, displayed in a ways that is clear and simple to visualize true positives (TP), false positives (FP), false negatives (FN) and false positives (FP) [37].



	Pred. Pos	Pred. Neg
Actual Pos.	TP	FN
Actual Neg.	FP	TN

Figure 3.3: Confusion matrix.

The values of TP, FP, FN and FP are used to calculate model evaluation metrics The metric used to select the best model will be the F1 score, calculated as the harmonic mean of the precision and the recall.

$$
\begin{aligned}
Precision &= \frac{TP}{TP+FP} \\
Recall &= \frac{TP}{TP+FN} \\
F1 \quad Score &= \frac{2*(Precision*Recall)}{Precision+Recall}
\end{aligned}
$$

The F1 score is good metric because it balances precision and recall. Precision, also known as confidence, is the proportion of predicted positives that are real positives. Recall, in the other hand, is the proportion of real positives that are predicted correctly.

3.5.3 k Nearest Neighbors

kNN is a simple and intuitive machine learning algorithm that assigns to an unknown observation the same category of the majority of the k nearest known observations according to the selected distance function. Known observations come from the training set [38].

Formally, if we call $L = \{(\mathbf{x}_1, y_i), (\mathbf{x}_2, y_2, \ldots, (\mathbf{x}_n, y_n)\}$ the learning set where $\mathbf{x}$ is an observation and y is its class. When a new observation $\mathbf{x}$ arrives, we can calculate, for $k = 1$ neighbor [38]:

$$d(\mathbf{x}, \mathbf{x}^{(1)}) = min(d(\mathbf{x}, \mathbf{x}_i))$$

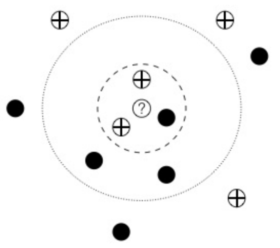


Figure 3.4: kNN example. When the new observation (marked with question mark) arrives, we use the class of the k nearest observations to assign a class to it. Notice that if $k = 3$ (inner circle), the class would be different than if $k = 5$ (outer circle).

And finally, the class of $\mathbf{x}$ will be $y = y^{(1)}$. In this case, the notation $^{(j)}$ represents the $j - th$ nearest neighbors. [38] For k neighbors (k being odd number to avoid ties) in a binary classification problem, the decision can be generalized by:

$$
\begin{aligned}
y = 0 \quad if \quad & \sum_{i=1}^{k} Y^{(i)} \leq \frac{k}{2} \\
y = 1 \quad if \quad & \sum_{i=1}^{k} Y^{(i)} > \frac{k}{2}
\end{aligned}
$$

Naturally, a clear shortcoming of this method is the fact that the prediction can be compromised in unbalanced datasets, where a class is more frequent than the other [38].

3.5.4 Support Vector Machines

SVM is also a supervised machine learning algorithm, but it tries to find the hyperplane that separate the classes with maximum margin [39, 40]. Lets suppose that we have a training set $T = \{(\mathbf{x}, y)\}$ with two classes, a positive $(+1)$ and a negative (-1), linearly separable by a hyperplane with normal vector $\mathbf{w}$ and intercept b, as in the figure below:

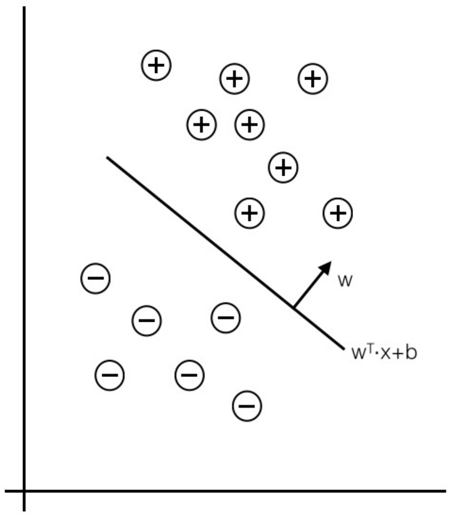


Figure 3.5: Dataset linear separable by a hyperplane of the format $\mathbf{w}^T \cdot \mathbf{x} + b$

The classification function can be written as $f(\mathbf{x}) = sign(\mathbf{w}^T \cdot \mathbf{x} + b)$, being $+1$ for the positive class and -1 for the negative classes [39, 40]. Intuitively for the positive class, the larger $\mathbf{w}^T \cdot \mathbf{x} + b$, more confidence one have in the classification function. For the negative class, the smaller $\mathbf{w}^T \cdot \mathbf{x} + b$ the better. If A is the closest point of the dataset to the hyperplane, its orthogonal projection on the hyperplane A' can be calculated by [39]:

$$A' = \mathbf{x} - (y)\frac{\mathbf{w}}{\|\mathbf{w}\|}r$$

Where y is there just to change the signal in case A is of the positive or negative class, $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is a unitary vector in the normal direction of the hyperplane and r is the smallest distance between A and the hyperplane. As the projection A' is on the hyperplane, it is correct to say that [39]:

$$\mathbf{w}^T \cdot A' + b = 0$$

and so

$$\mathbf{w}^T \cdot (\mathbf{x} - (y)\frac{\mathbf{w}}{\|\mathbf{w}\|}r) + b = 0$$

Solving it for r, we get that [39]:

$$r = y\frac{\mathbf{w}^T \cdot \mathbf{x} + b}{\|\mathbf{w}\|}$$

And r is called the margin. This can be seen in the figure below:

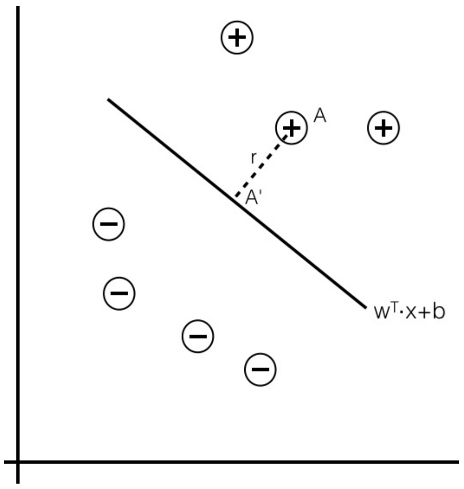


Figure 3.6: Point A, its orthogonal projection A' on the hyperplane. The distance r between A and A' is the margin.

And the hyperplane that better separate the classes are the hyperplane of maximum margin, meaning that we want to maximize $2r$ subjected to the constrain of the margin of every datapoint of the training set to be larger than r. This is the same as minimize $\frac{1}{2r}$ [39, 40]. To make the optimization problem convex and therefore converge to a global minimum, we can realize that the margin is invariable to scaling of the hyperplane through the normalization by $\|\mathbf{w}\|$. That way, r can be admitted to be 1 and our objective function becomes [40]:

$$min_{\mathbf{w},b}\frac{\|\mathbf{w}\|}{2}$$
$$s.t. \quad y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1, \quad \forall\{(\mathbf{x}_i, y_i)\} \in L$$

Solving this quadratic optimization problems will provide global optimal values for $\mathbf{w}$ and b, the parameters of the hyperplane with maximum separation of the dataset L. Notice that this case assumes perfect separation of the classes, which is very rarely the case [39, 40]. To make this algorithm more flexible, slack variable ξ for each datapoint are added to the optimization problem, with an overall cost C, in order to balance between maximizing the margin and restricting all data points to be correctly classified [39, 40]. Considering the training set $L = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$, the optimization problem with the slack function is represented by [40]:

$$min_{\mathbf{w},b} \frac{\|\mathbf{w}\|}{2} + C \sum_{i=1}^{m} \xi_i$$

$$\begin{aligned} s.t. \quad & y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1 \\ & \xi_i \geq 0 \\ & for \quad i \in \{1, 2, 3, \ldots, m\} \end{aligned}$$

Dual representation

Although the quadratic optimization problem presented in the last section is a standard problem with already known methods to be solved, more modern approaches to the SVM model suggest that the dual version of the original problem (know as primal) should be used instead [39, 40]. The original problem is known as primal, and as the dot product $\mathbf{w}^T \cdot \mathbf{w}$ is convex and all the restrictions can be written in the format:

$$g_i(\mathbf{w}) = -y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) + 1 \leq 0$$

The solution of the dual is equivalent to the solution of the primal [40]. The dual version can be constructed by [39, 40]:

$$max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

$$\begin{aligned} s.t \quad & 0 \leq \alpha_i \leq C, \quad for \quad i \in \{1, \ldots, m\} \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned}$$

In the dual form, each α_i is the Lagrange multiplier associated with restriction $g_i(\mathbf{w})$. The only α_i that are not zero are the ones from observations where the margin is 1. This points are called support vectors [39].

The classification function is:

$$f(\mathbf{x}) = sign(\sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b)$$

Notice how the classification function only relies on the support vectors ($\mathbf{x}_i$ where α_i is not zero). This is a key point in this representation and allows the application of the kernel transformation, described on the next section [39, 40].

Kernel transformation

Let $\phi(\mathbf{x})$ be a function that maps the observations from their original dimensional space R^n to a feature space of higher dimensionality R^m, where the separation of the classes is more clear. Then, the kernel function is the dot product $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ [39, 40]. As the classification function relies on the inner product, it can be seamless replaced by the kernel function:

$$f(\mathbf{x}) = sign(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b)$$

The kernel can also be represented as a square symmetric matrix $m \times m$, where each cell (i, j) is the inner product $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. For a Kernel to be valid, this matrix need to be symmetric positive semi definite. [39, 40]

Most common kernel operators are the polynomial and the rbf (radial basis function) [40]. The polynomial maps the observations to a higher dimensionality feature space by applying a transformation of the following format [40]:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^d$$

In the case of $d = 1$, the kernel is known as the linear kernel.

The rbf kernel uses a Gaussian distribution to do the mapping [40]:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \frac{- \|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}$$

This kernel maps the observations to a infinite dimensional space, known as Hilbert space. [39, 40]

3.6 Post-processing

The output of the classification process is a list of clusters with their respective label “product” of “not-product”. Now, it is time to put the product clusters together to create the output picture. To perform this task, as the clusters are basically binary matrices, it is just a matter of sum the clusters classified as products. This will generate a new matrix where all the pixels with value 1 are product pixels, and all the pixels with value 0 are not product pixels. A new image is generated from the original RGB color picture, but only with the product pixels.

3.7 Output

From this new image generated, we can extract the color as being the RGB values most common on the output picture, since it now is segmented only with the regions that contain a product. The output is a list of RGB values and its percentage on the new image.

The RGB code is a starting point, in reality the output depends on how this information is going to be used. From the RGB code, a color name can be searched on RGB color list, a HEX color code (used in web design) can be calculated, a TPX code (used in art design) can be searched also on standard lists, etc.

Chapter 4

Results

4.1 The dataset

To put the framework to test, a dataset of 211 pictures of dresses will be used. The pictures have significant variation of backgrounds, colors, sizes etc. The only restriction is that all products in the dataset are dresses.

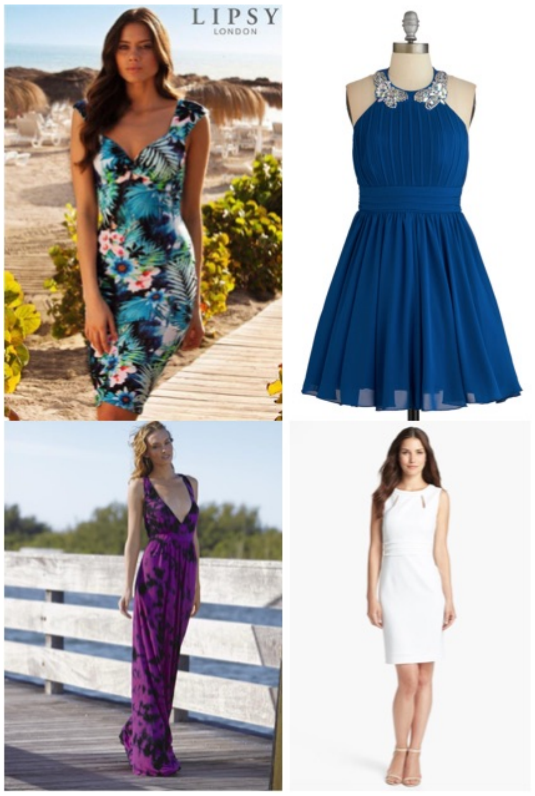


Figure 4.1: Examples from the dataset. Notice how some pictures have complex backgrounds, other have white on white contrast problems, others are rather simple.

The objective of the framework is to isolate the dresses from all other elements of the picture and be able to guess its main colors.

4.2 Pre-processing

Following the framework proposed, the first step is to resize all the pictures to the same size. Empirically, for this dataset 150×150 is small enough for reduce the dimensionality of the problem but still clear enough to understand all elements of the picture. Then comes the first results, the color space transformation. As seen in earlier sections, the objective of the color space transformation is to reduce the correlation among the channels of the original RGB channels. On the next figure, details on the channel of the bottom left picture of 4.1:

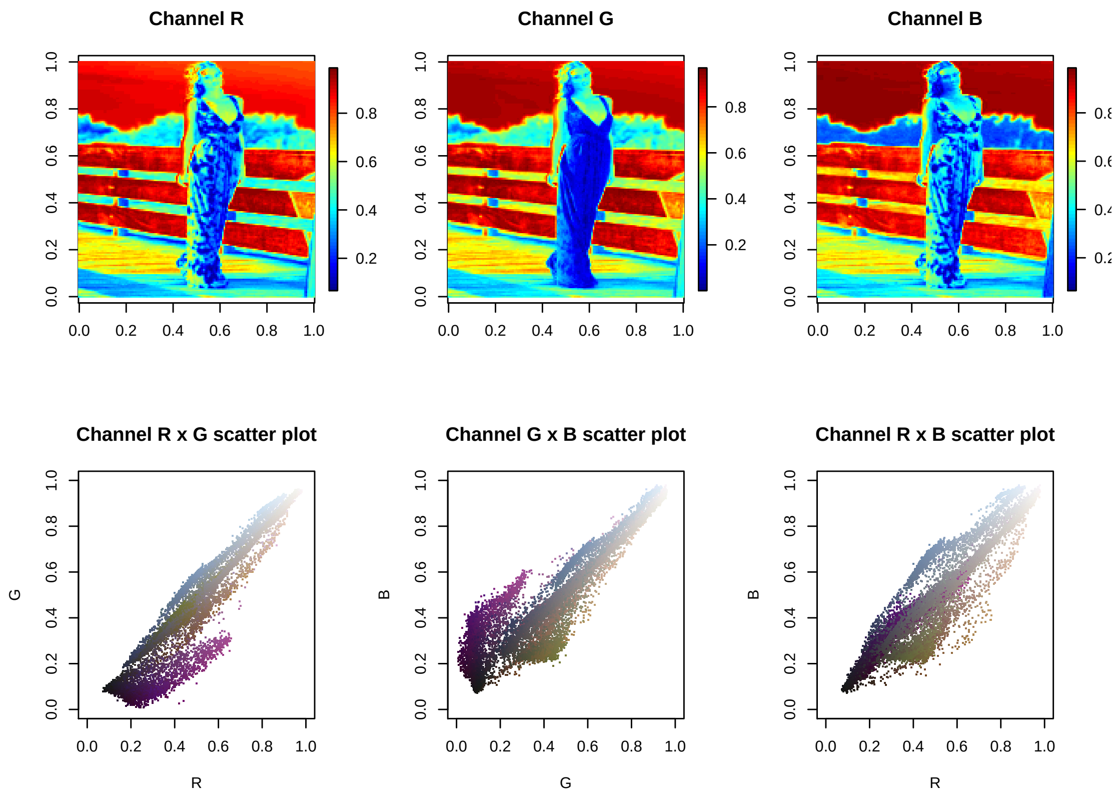


Figure 4.2: Heat map and scatterplot of RGB channels. Notice how the three channels have similar heat maps. On the scatterplots, notice how the points are distributes along the main diagonal.

On the last figure, the similarity between the heat map of the three channels and the scatterplot of the pixels already indicates visually a high correlation among the channels. Calculating the actual correlation matrix for that picture, we have:

	R	G	B
R	1.00	0.97	0.94
G	0.97	1.00	0.95
B	0.94	0.95	1.00

Table 4.1: Correlation matrix of RGB channels for sample image

As expected, the channels are highly correlated. Applying the HSV transformation on the same picture, yield the following results:

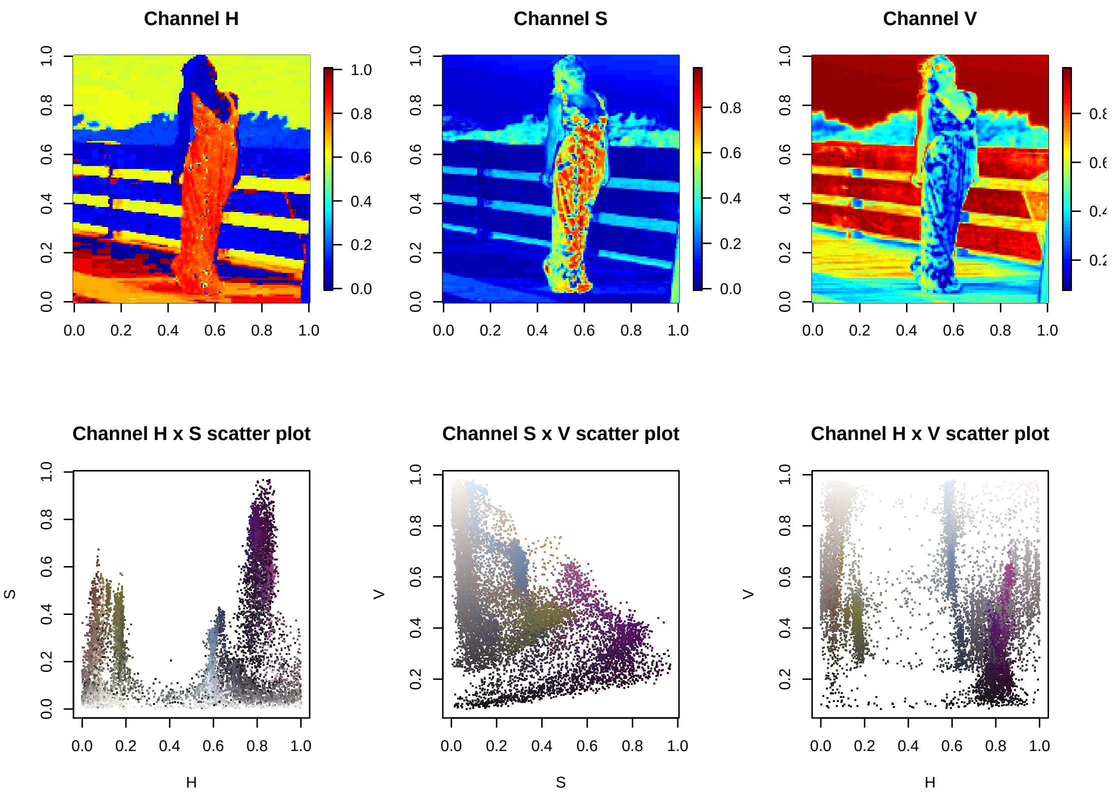


Figure 4.3: Heat map and scatterplot of HSV channels. Notice how the heat maps are different and how the scatterplot is more scattered on all dimensions.

The HSV transformation seems to have reduced the correlation. Furthermore, check also on 4.3 how on the $H \times S$ scatterplot, on the H axis the colors seems to smoothly transition from beige to purple. Also, on the other scatterplots, how on the V axis the colors go from black to white. It follows what was discussed on the HSV transformation. Checking the correlation matrix we have now:

	H	S	V
H	1.00	0.31	-0.34
S	0.31	1.00	-0.62
V	-0.34	-0.62	1.00

Table 4.2: Correlation matrix of HSV channels for sample image

Doing the same experiment for the CIE Luv transformation, we have the results below:

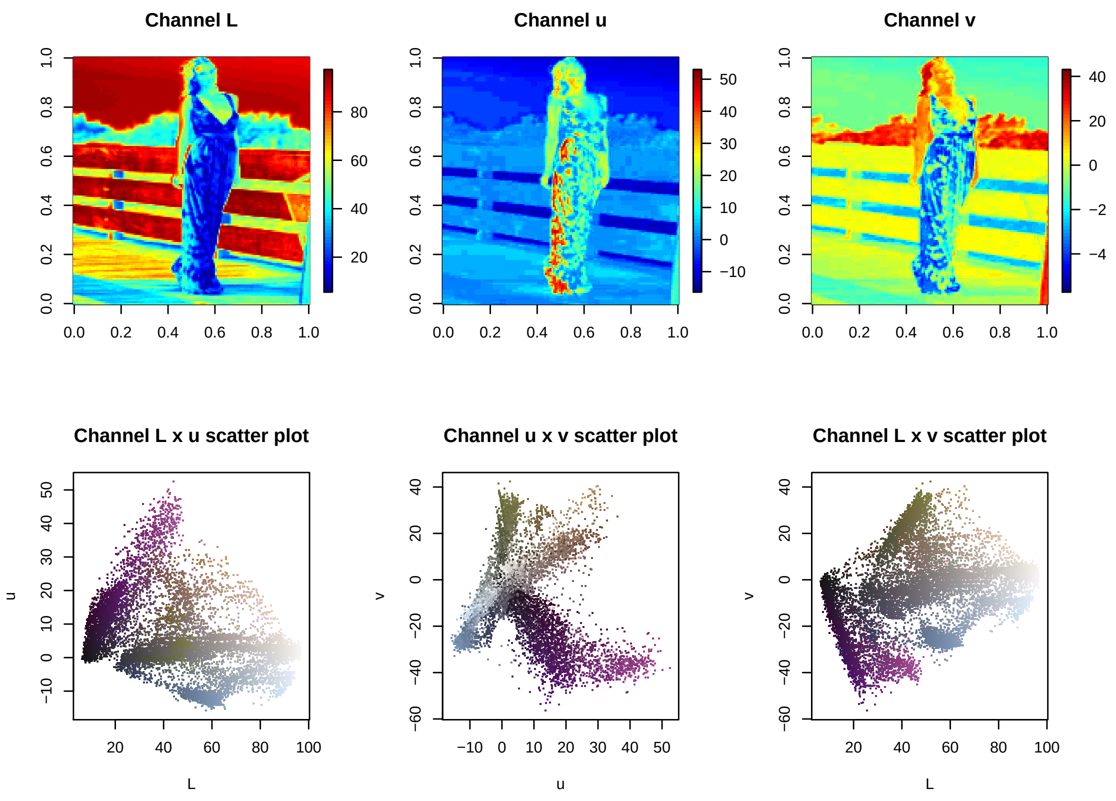


Figure 4.4: Heat map and scatterplot of Luv channels. Again, heat maps and scatter-plots very distinct among each other.

The differences on the heat maps are remarkable. In fact, from the correlation matrix below, it is possible to see that it reduced the correlation a step further when compared to the HSV:

	L	u	v
L	1.00	-0.45	0.20
u	-0.45	1.00	0.01
v	0.20	0.01	1.00

Table 4.3: Correlation matrix of Luv channels for sample image.

Analyzing now the whole dataset of 211 pictures, we have the following correlation matrix for the original RGB color space:

	R	G	B
R	1.00	0.88	0.86
G	0.88	1.00	0.97
B	0.86	0.97	1.00

Table 4.4: Correlation matrix of RGB channels on the full dataset

Applying the same transformation, now for the whole dataset, we can se the following reduction for the HSV channels:

	H	S	V
H	1.00	0.43	-0.40
S	0.43	1.00	-0.42
V	-0.40	-0.42	1.00

Table 4.5: Correlation matrix of HSV channels on the full dataset

And finally for the Luv, transforming the full dataset:

	L	u	v
L	1.00	-0.28	-0.04
u	-0.28	1.00	0.46
v	-0.04	0.46	1.00

Table 4.6: Correlation matrix of Luv channels of the full dataset

Again, clearly both transformations were able to reduce the correlation, but the Luv color space is more beneficial. The second metric to be assessed is the processing time of the transformation, since on the literature review it was always stated that the Luv transformation is more computationally expensive:

	RGB	HSV	LUV
Elapsed time	345.97	346.95	621.38
% increase over RGB		0.28	79.61

Table 4.7: Processing time and increase over processing time using only RGB color space

From the table above we can see that the processing time for transforming from RGB to HSV is almost non existent. In the other hand, transforming from RGB to Luv increases processing time in almost 80%. As both fill the requirement of reducing correlation, we will opt for using HSV color space due to the requirement of fast processing time.

4.3 Clustering

The metric used to evaluate the clustering process is the reduction of the original WSS. In that sense, for each picture we can evaluate how the WSS reduces as we increase the number of clusters. Using the same test image used for the color space discussion, we have figure 4.5. There, we can see how WSS reduces as we increase the number of clusters. If we set a threshold of 10% of the original WSS, 7 is the number of clusters needed to cross this threshold.

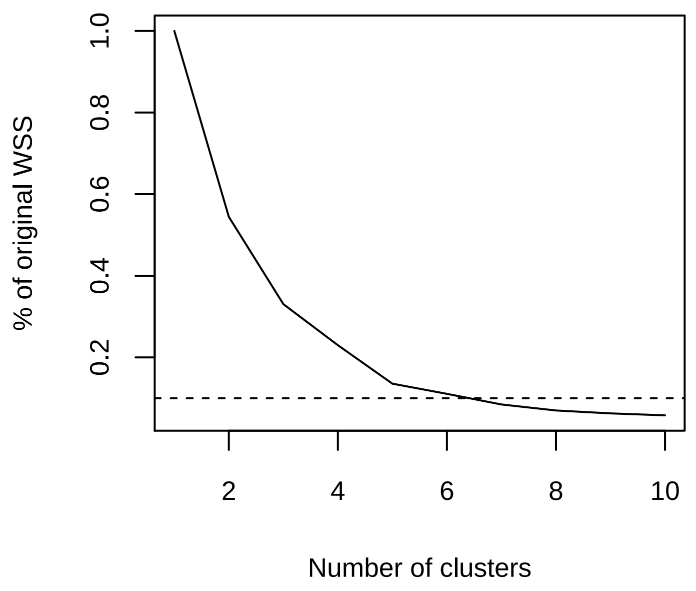


Figure 4.5: Reduction of original WSS with the increase of number of clusters. Dashed line indicates reduction to 10% of the original value

On the figure 4.6, it is clear that using 7 clusters was good enough to isolate most of the product (dress) from the other elements. The results on the other clusters can be easily interpreted too:

- Cluster 1 and 5 isolated, respectively, the wooden boardwalk and the fence behind the model
- Cluster 2 and 6 isolated, respectively, the water and the sky on the background
- Cluster 4 grouped the model skin with the vegetation in the background. Possibly because her skin is shadowed, appearing in a darker tone.
- Cluster 3 got some of the black details of the dress, grouped with the shadowed area of the boardwalk. It can be argued that this could be considered part of the product, but it is not as representative as the information on cluster 7.

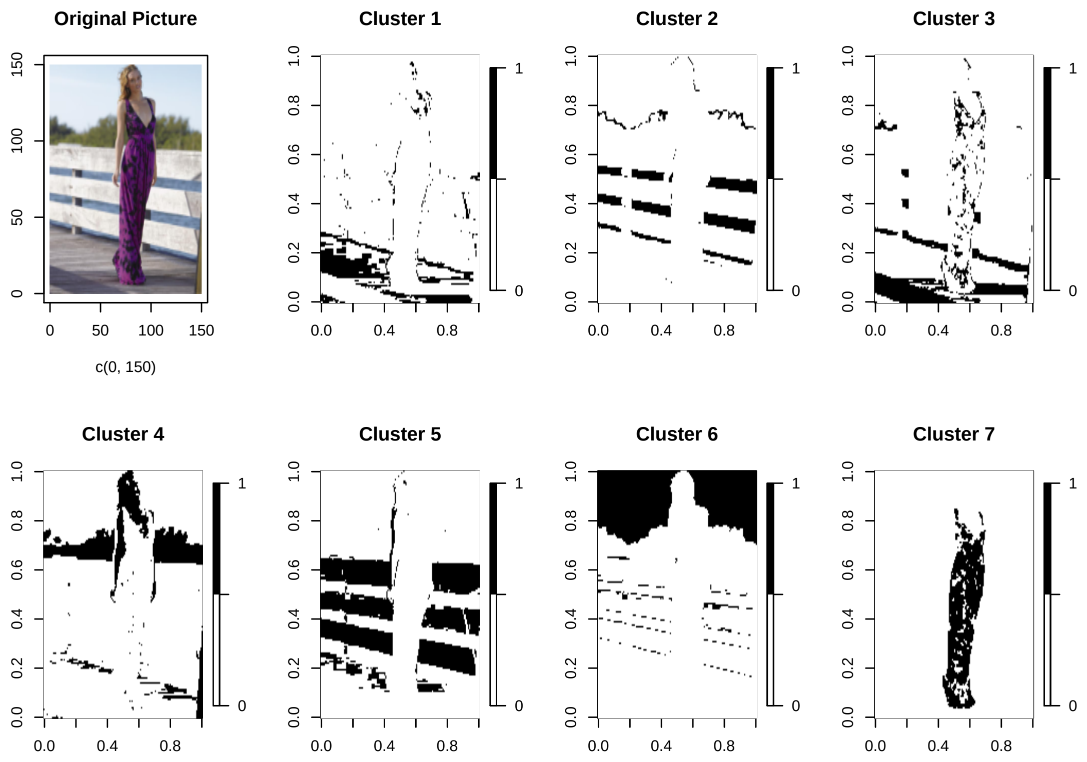


Figure 4.6: k-means output with 7 clusters. Notice how cluster 7 isolated the dress.

Applying the same analysis of WSS reduction for all pictures in the dataset, we have the figure 4.7, where on average, after reaching 10% of the original WSS, the marginal gain is not as representative as on the previous clusters. So for the each picture processed by our framework, the number of clusters to be used will be the first number in which the WSS crosses the threshold of 10% of the its original value, or a maximum value of 10 cluster, to avoid creating too small and uninformative regions.

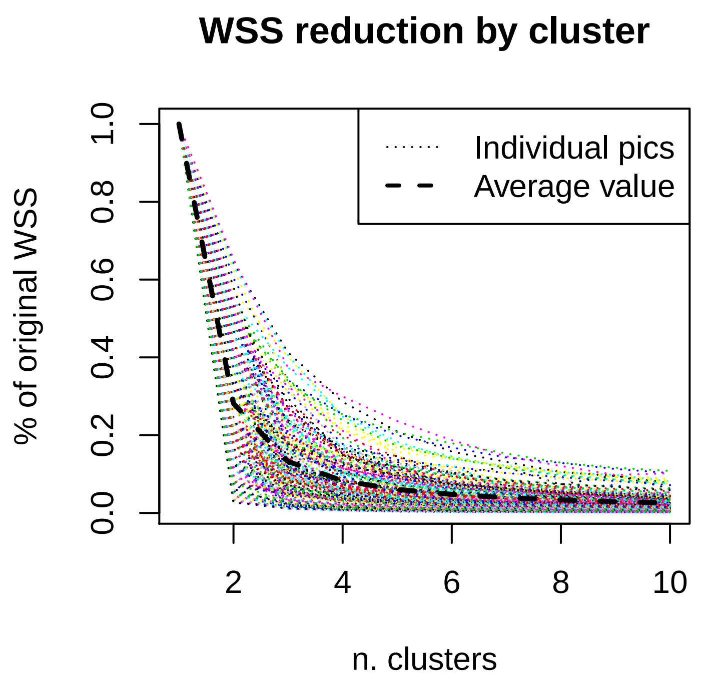


Figure 4.7: Analysis of WSS reduction by number of clusters used. Notice how on average, 4 clusters are enough to reduce WSS to 10% of the original.

4.4 Classification

4.4.1 The training set

Applying the clustering algorithm to the original dataset of 211 pictures, generated a total of 897 cluster silhouettes, that were manually labeled as "product"or "not product". The "product" class represents 37.9% of the dataset. This dataset is then divided into a training set and a test set as described on the methodology section, with the training set being $\frac{2}{3}$ of the original 897 observations. Two algorithms will be tested:

- kNN with 1, 3 and 5 neighbors and
- SVM with kernels: linear, polynomial of degrees 2 to 5 and rbf.

Accordingly to what was discussed on the methodology section, the F1 score will be the metric used to compare the results.

4.4.2 Comparing results

Applying the kNN to the dataset yielded the following results:

k	F1.Score
1	0.73
3	0.74
5	0.73

Table 4.8: F1 Score for the kNN models

The confusion matrix for the model using 3 neighbors is:

	Pred. Pos.	Pred. Neg.
Actual Pos.	68	48
Actual Neg.	0	182

Table 4.9: Confusion matrix for the best kNN model

From the confusion matrix, we can see that the model have a great precision (all the positive prediction are correct), but have a very low recall (only around 58% of the actual positive categories were predicted correctly). In practical terms, this means that on production every cluster that this model predicted as product, most likely will be a product, but on the other hand, the model will not be able to identify almost half of the clusters that are actually products.

The SVM results are more promising:

Model	F1.Score
Linear	0.91
Poly-2	0.88
Poly-3	0.91
Poly-4	0.82
Poly-5	0.85
RBF	0.93

Table 4.10: F1 Score for the SVM models

The confusion matrix for the model with rbf kernel:

	Pred. Pos.	Pred. Neg.
Actual Pos.	105	11
Actual Neg.	4	178

Table 4.11: Confusion matrix for the best SVM model

The F1 Scores are considerably higher than the ones seen in the kNN model. Even more significant is to understand that the better F1 score comes from a better balance between precision and recall, as discussed on the methodology section. The SVM model using *rbf* kernel was able to achieve a precision of 96% and a recall of 91%.

To guarantee a good generalization of this model, a 5-fold cross-validation was done. In this process, the training set is randomly divided in 5 subgroups that are, one at a time, removed from the training set and used to validate the model trained. There were not significant differences in the results of the 5 folds, meaning that the models seems to have a good generalization.

The model can still be optimized. In the methodology section, we discussed that the SVM have a cost parameter C that can be used to penalize misclassification. To choose the best value for this parameter, a grid search approach is used. For that, we train the model with different values of C and verify its F1 score. The results of this process in the table below:

C	F1.Score
1	0.9333
2	0.9422
4	0.9381
8	0.9292
10	0.9292
15	0.9292
20	0.9292
25	0.9292
50	0.9292
100	0.9292

Table 4.12: Grid search results to specify the best value of C

Using a cost C of 2 generated the best results. Updating the model results for this new cost, we have and F1 Score of 0.9422 and the following confusion matrix:

	Pred. Pos.	Pred. Neg.
Actual Pos.	106	10
Actual Neg.	3	179

Table 4.13: Confusion matrix for the best SVM model

Comparing to the previous one, we can see that using a better cost C generated one more TP and one more TN. This is because, on the optimization of the hyperplane, we are increasing the weight of minimizing the misclassification, as discussed on the methodology.

Now that the the model is completely defined, it is possible to test it on the clusters generated on the clustering section. For that, each one of the seven cluster is used as input of the classification model, if the output is 0, our negative class, the cluster is not a product. If the output is a 1, the cluster is classified as a product. The output can be seen in the figure below:

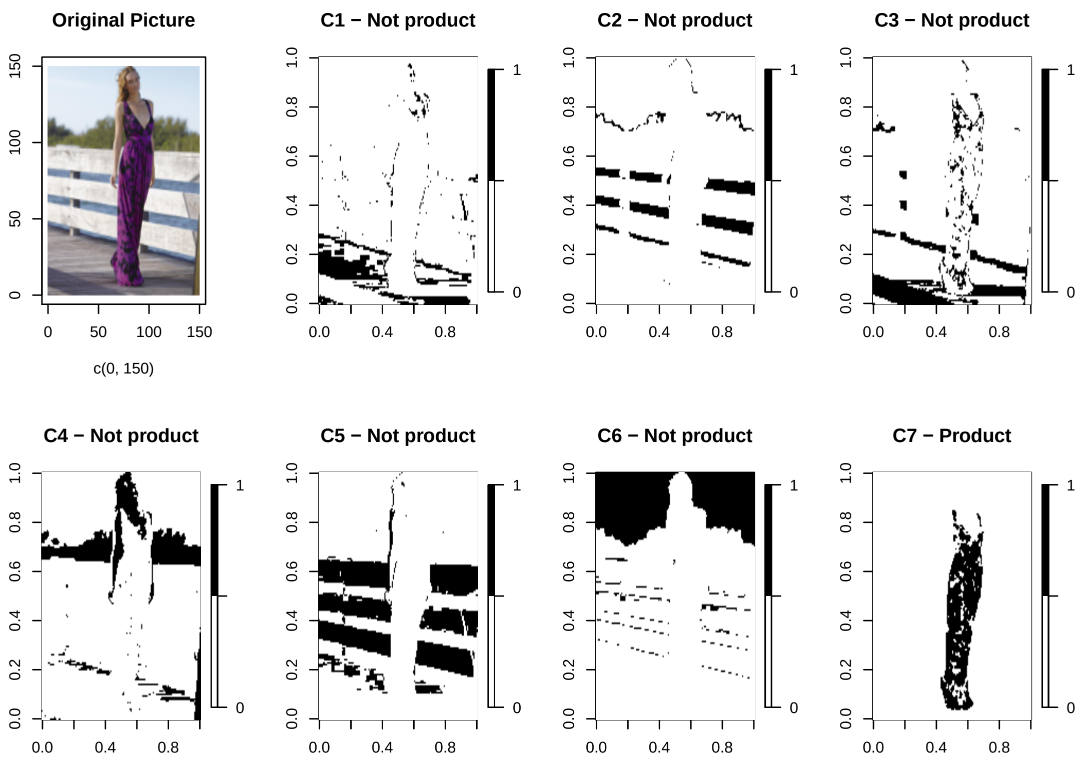


Figure 4.8: Model output.

As expected, the model only classified cluster 7 as product.

4.5 Post-processing

Now that the clusters are classified, following the framework proposed, we can sum the product clusters into a binary matrix and use this matrix to generate a new picture, only with the pixels with value one on the matrix. For the picture of the last examples, we have:

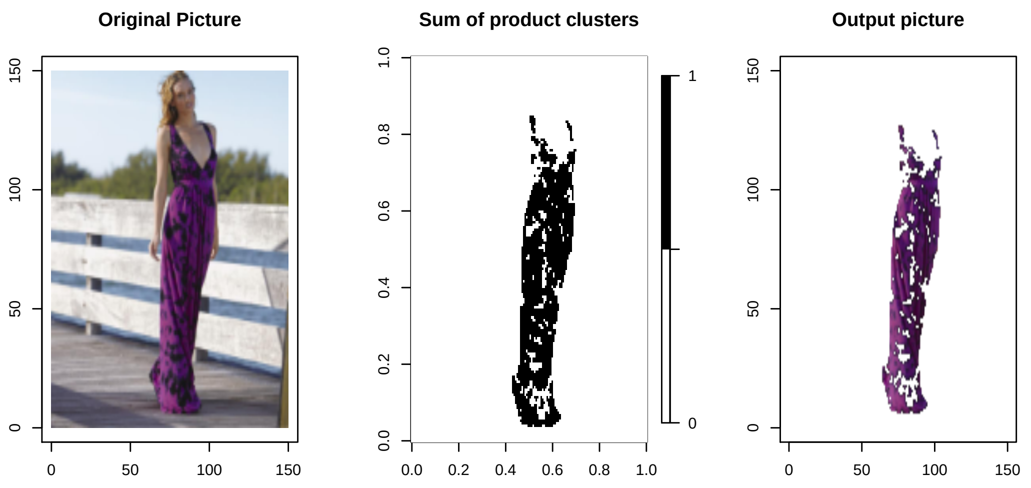


Figure 4.9: Output picture

4.6 Output

Now, simply calculating the pixel distribution, we can generate a color list.

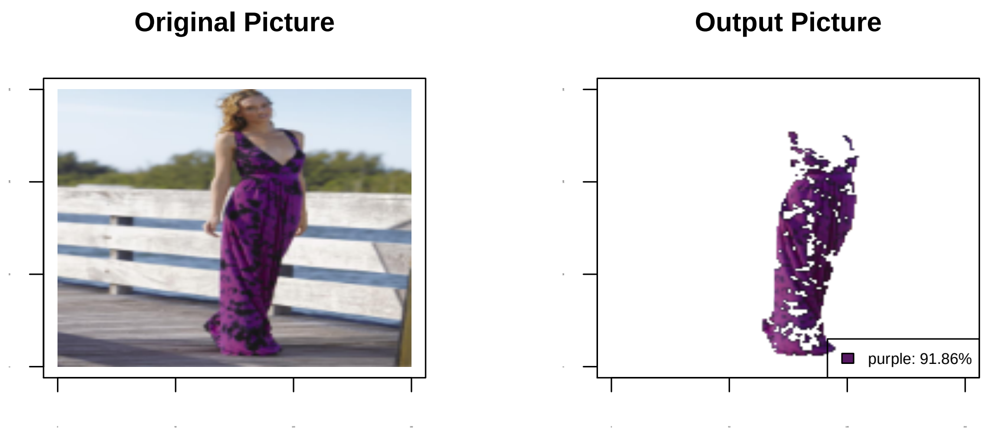


Figure 4.10: Original and output picture with color list.

Notice how for 4.10, the color purple covers almost 92% of the product. The rest are below a threshold that can be defined by the user. For this example, we are using 3%.

Chapter 5

Conclusion

We provided a complete framework to remove non interesting elements in a color product picture to improve the relevant color detection. The framework uses simple machine learning techniques to achieve a color segmentation and classify this segments based on labeled examples.

We verified the impact of using different color spaces, not only regarding the reduction of the correlation among RGB channels, but also on the impact of process time of the CIELuv transformation, as cited on most of the literature.

Regarding classification, it is clear the difference and the flexibility that the different kernels add to the linear SVM algorithm. Also, performing grid search to better select the cost parameter improved results even more. The F1 score proved to be a good metric to balance precision and recall of the models.

One important observation is about the cost of acquiring the labeled data. Manually classifying almost a thousand clusters was time consuming, but the results are worthy.

The model works on simple pictures, like products with a main color over an uniform background:

To harder examples, like pictures with low contrast with the background:

To real complex pictures, with natural background:

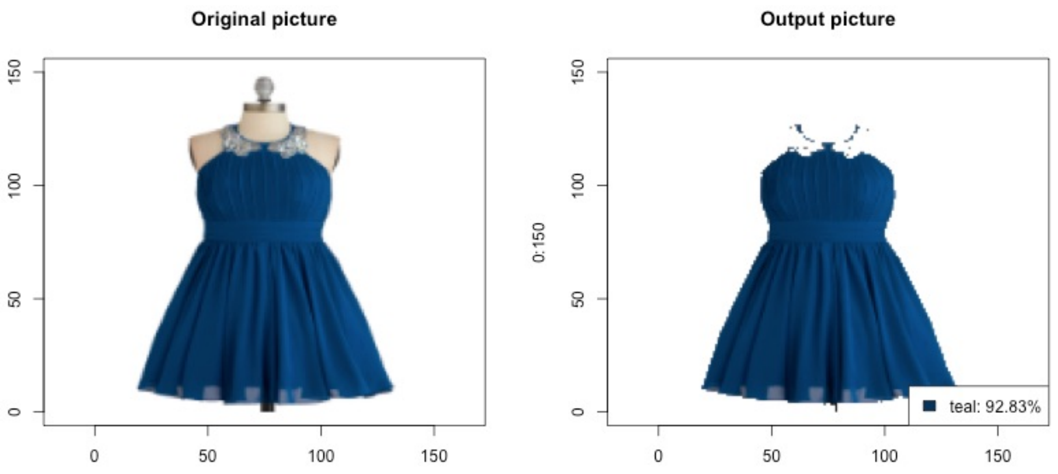


Figure 5.1: Simple picture. On this example, only two cluster were needed to achieve product isolation. Notice how the support on which the dress was laying was also removed.

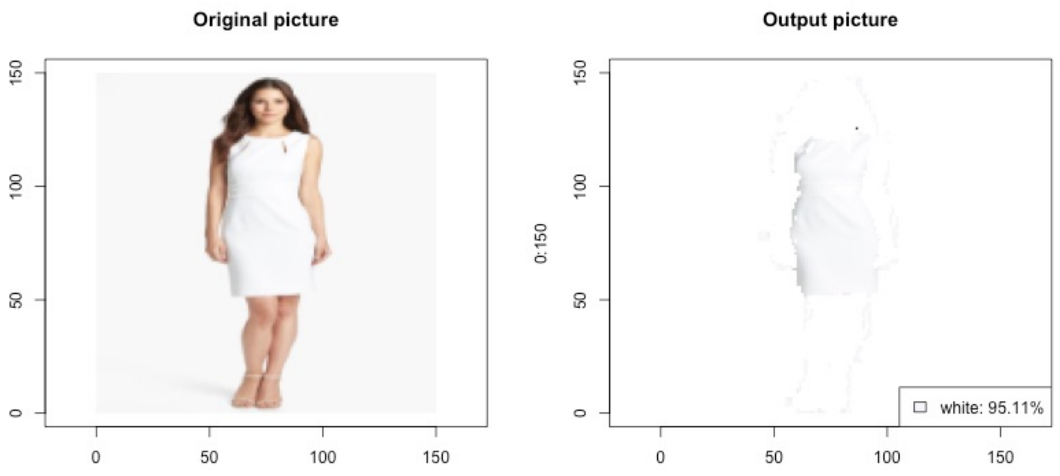


Figure 5.2: White product on white background. Here, 4 clusters were used to detect the white background, the model body, model hair and finally the white dress.

5.1 Future research

There are improvements that can be made to this framework. One option is to experiment with different cluster algorithms, like PAM (Partition Around Medoids). In this method, instead of using the center of gravity of the observations, one of the observations is used as centroid. Still on clustering, another possible research is on improving the selection of optimal number of clusters.

Another possibility is using the kNN with different distance metrics. In this framework we just explored different number of neighbors, but it is possible that other distance

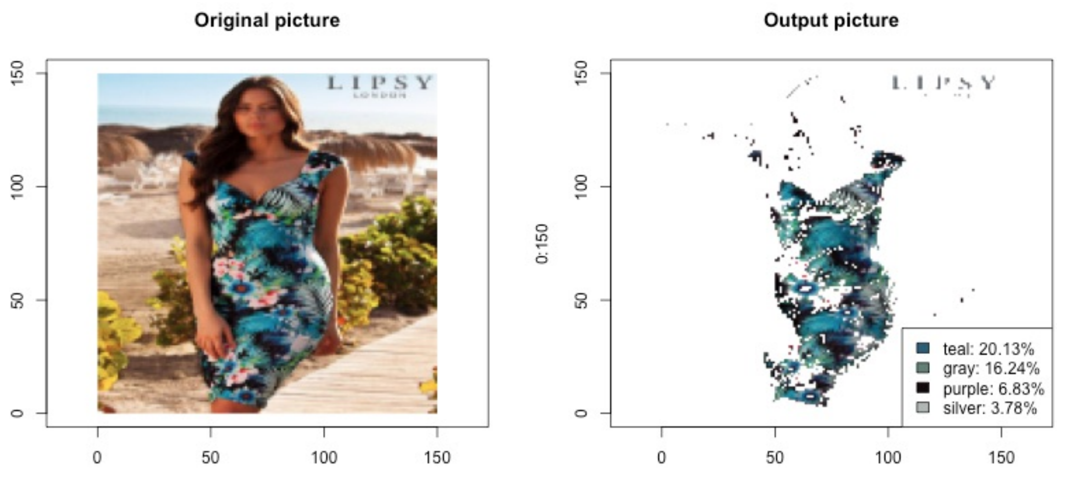


Figure 5.3: Floral dress over complex natural background. Here, the number of clusters was selected by the max allowed of 10, not by the WSS reduction, which caused the loss of some of the details. But in any case, the model was still able to identify the main colors of the dress.

metrics, like cityblock or Hamming could have given better results.

Also, the post-processing can be improved by adding filters to avoid empty areas inside a closed region. For instance, the product picture on figure 5.4:

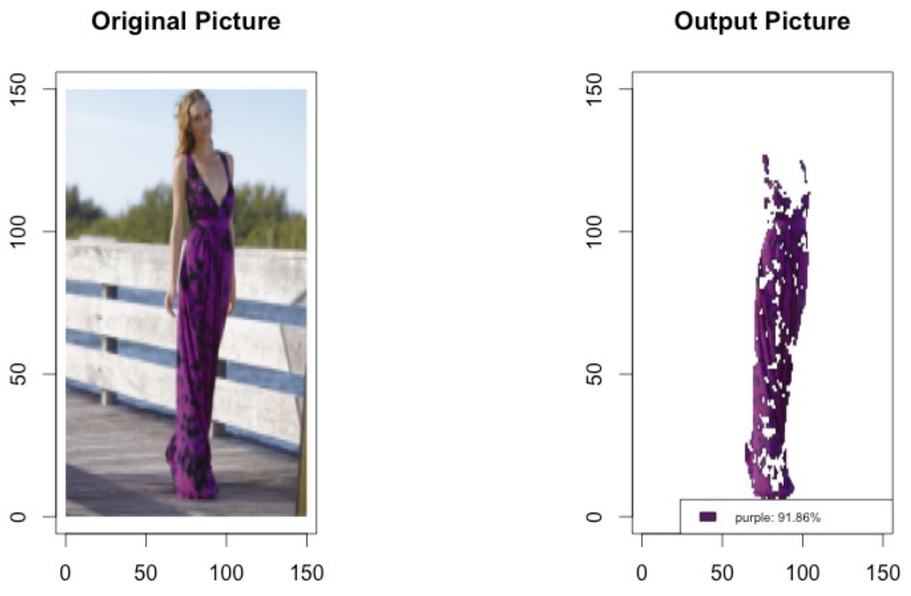


Figure 5.4: Empty spaces inside product region

Some areas inside the product were removed with another cluster. A square filter can be used to detect these areas inside product regions, improving even more the color identification on product details.

Bibliography

[1] H.D. Cheng, X.H. Jiang, Y. Sun, and Jingli Wang. “Color image segmentation: advances and prospects”. In: *Pattern Recognition* 34.12 (2001), pp. 2259–2281. DOI: 10.1016/s0031-3203(00)00149-7.

[2] Mehmet Celenk. “A color clustering technique for image segmentation”. In: *Computer Vision, Graphics, and Image Processing* 51.3 (1990), p. 370. DOI: 10.1016/0734-189x(90)90009-k.

[3] Jorge S Marques, Nicolas Perez de la Blanca, and Pedro Pina. *Pattern Recognition and Image Analysis.* Springer-Verlag Berlin Heidelberg, 2005, pp. 239–246.

[4] P. Wayne Power and Roger S. Clist. “Comparison of supervised learning techniques applied to color segmentation of fruit images”. In: *Intelligent Robots and Computer Vision XV: Algorithms, Techniques,Active Vision, and Materials Handling* (1996). DOI: 10.1117/12.256294.

[5] T. Uchiyama and M.A. Arbib. “Color image segmentation using competitive learning”. In: *IEEE Trans. Pattern Anal. Machine Intell.* 16.12 (1994), pp. 1197–1206. DOI: 10.1109/34.387488.

[6] M.R. Tabassum, A.Ul. Gias, M.M. Kamal, S. Islam, H.M. Muctadir, M. Ibrahim, A.K. Shakir, A. Imran, S. Islam, and M.G. et al. Rabbani. “Comparative Study of Statistical Skin Detection Algorithms for Sub-Continental Human Images”. In: *Information Technology J.* 9.4 (2010), pp. 811–817. DOI: 10.3923/itj.2010.811.817.

[7] J. Brand and J.S. Mason. “A comparative assessment of three approaches to pixel-level human skin-detection”. In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000* (2000). DOI: 10.1109/icpr.2000.905653.

[8] Alvise Lastra, Alberto Pretto, Stefano Tonello, and Emanuele Menegatti. “Robust Color-Based Skin Detection for an Interactive Robot”. In: *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing* (2007), pp. 507–518. DOI: 10.1007/978-3-540-74782-6_44.

[9] “Skin Detection”. In: *Encyclopedia of Biometrics* (2009), pp. 1218–1224. DOI: 10.1007/978-0-387-73003-5_89.

[10] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. “A survey of skin-color modeling and detection methods”. In: *Pattern Recognition* 40.3 (2007), pp. 1106–1122. DOI: 10.1016/j.patcog.2006.06.010.

[11] Adrian Ford and Alan Roberts. *Colour space conversions.* Technical report. 2015.

[12] Linda G Shapiro and George C Stockman. *Computer vision.* Prentice Hall, 2001, pp. 305–351.

[13] D.R. Martin, C.C. Fowlkes, and J. Malik. “Learning to detect natural image boundaries using local brightness, color, and texture cues”. In: *IEEE Trans. Pattern Anal. Machine Intell.* 26.5 (2004), pp. 530–549. DOI: 10.1109/tpami.2004.1273918.

[14] A.D. Jepson and D.J. Fleet. *Edge Detection.* 1st ed. Department of computer science - University of Toronto, 2011. URL: http://www.cs.toronto.edu/~fleet/courses/2503/fall11/Handouts/edgeDetection.pdf.

[15] Yan Li, Siwei Luo, and Qi Zou. “Learning to Detect Boundaries in Natural Image Using Texture Cues and EM”. In: *2008 Fourth International Conference on Natural Computation* (2008). DOI: 10.1109/icnc.2008.233.

[16] H.D. Cheng, X.H. Jiang, and Jingli Wang. “Color image segmentation based on homogram thresholding and region merging”. In: *Pattern Recognition* 35.2 (2002), pp. 373–393. DOI: 10.1016/s0031-3203(01)00054-1.

[17] Thanh N. Tran, Ron Wehrens, and Lutgarde M.C. Buydens. "Clustering multispectral images: a tutorial". In: *Chemometrics and Intelligent Laboratory Systems* 77.1-2 (2005), pp. 3–17. DOI: 10.1016/j.chemolab.2004.07.011.

[18] Yu-Ichi Ohta, Takeo Kanade, and Toshiyuki Sakai. "Color information for region segmentation". In: *Computer Graphics and Image Processing* 13.3 (1980), pp. 222–241. DOI: 10.1016/0146-664x(80)90047-7.

[19] K.S. Sim, C.P. Tso, and Y.Y. Tan. "Recursive sub-image histogram equalization applied to gray scale images". In: *Pattern Recognition Letters* 28.10 (2007), pp. 1209–1221. DOI: 10.1016/j.patrec.2007.02.003.

[20] S. Arora, J. Acharya, A. Verma, and Prasanta K. Panigrahi. "Multilevel thresholding for image segmentation through a fast statistical recursive algorithm". In: *Pattern Recognition Letters* 29.2 (2008), pp. 119–125. DOI: 10.1016/j.patrec.2007.09.005.

[21] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/tpami.1986.4767851.

[22] Manoj K. Vairalkar and S.U. Nimbhorkar. "Edge Detection of Images Using Sobel Operator". In: *International Journal of Emerging Technology and Advanced Engineering* 2.1 (2012), pp. 291–293.

[23] Wenshuo Gao, Xiaoguang Zhang, Lei Yang, and Huizhong Liu. "An improved Sobel edge detection". In: *2010 3rd International Conference on Computer Science and Information Technology* (2010). DOI: 10.1109/iccsit.2010.5563693.

[24] S. Lakshmi and Dr.V. Sankaranarayanan. "A study of Edge Detection Techniques for Segmentation Computing Approaches". In: *International Journal of Computer Applications* CASCT.1 (2010), pp. 35–41. DOI: 10.5120/993-25.

[25] Dana E. Ilea and Paul F. Whelan. "Image segmentation based on the integration of colour–texture descriptors". In: *Pattern Recognition* 44.10-11 (2011), pp. 2479–2501. DOI: 10.1016/j.patcog.2011.03.005.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proc. IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[27] Bhaskar Mondal and J. Paul Choudhury. “A Comparative Study on K Means and PAM Algorithm using Physical Characters of Different Varieties of Mango in India”. In: *International Journal of Computer Applications* 78.5 (2013), pp. 21–24. DOI: 10.5120/13485-1189.

[28] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. “An efficient k-means clustering algorithm: analysis and implementation”. In: *IEEE Trans. Pattern Anal. Machine Intell.* 24.7 (2002), pp. 881–892. DOI: 10.1109/tpami.2002.1017616.

[29] Greg Hamerly and Charles Elkan. *Advances in Neural Information Processing Systems 16*. MIT Press, 2004, pp. 281–288.

[30] Francesca Rossi. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. AAAI Press, 2013, pp. 1677–1684.

[31] Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Chaitanya Swamy. “The effectiveness of lloyd-type methods for the k-means problem”. In: *JACM* 59.6 (2012), pp. 1–22. DOI: 10.1145/2395116.2395117.

[32] J. MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Fifth Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1967, pp. 281–297.

[33] Neelamma K. Patil, Ravi M. Yadahalli, and Jagadeesh Pujari. “Comparison between HSV and YCbCr Color Model Color-Texture based Classification of the Food Grains”. In: *International Journal of Computer Applications* 34.4 (2011), pp. 51–57.

[34] URL: http://www.easyrgb.com/index.php?X=MATH&H=02#text2.

[35] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. “Scalable k-means++”. In: *Proceedings of the VLDB Endowment* 5.7 (2012), pp. 622–633. DOI: 10.14778/2180912.2180915.

[36] David Arthur and Sergei Vasilvitskii. “k-means++: the advantages of careful seeding”. In: *18th annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, 2007, pp. 1027–1035.

[37] D. M. W. Powers. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation*. 2007.

[38] Klaus Hechenbichler and Klaus Schliep. “Weighted k-nearest-neighbor techniques and ordinal classification”. In: *Discussion Paper 399, SFB 386*. 1st ed. 2006.

[39] Andrew Ng. *CS 229 Lecture notes - Support Vector Machines*. 1st ed. Stanford University. URL: `http://cs229.stanford.edu/materials.html`.

[40] *Introduction to information retrieval*. Cambridge University Press, 2008, pp. 319–348.

[41] Christian Platzer, Martin Stuetz, and Martina Lindorfer. “Skin sheriff”. In: *Proceedings of the 2nd international workshop on Security and forensics in communication systems - SFCS ’14* (2014). DOI: `10.1145/2598918.2598920`.

[42] “How the initialization affects the stability of the k-means algorithm”. In: *ESAIM: PS* 16 (2012), pp. 436–452. DOI: `10.1051/ps/2012013`.

[43] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: `10.1023/a:1010933404324`.

[44] M. Borsotti, P. Campadelli, and R. Schettini. “Quantitative evaluation of color image segmentation results”. In: *Pattern Recognition Letters* 19.8 (1998), pp. 741–747. DOI: `10.1016/s0167-8655(98)00052-x`.

[45] Kanwal K. Bhatia, Anil Rao, Anthony N. Price, Robin Wolz, Joseph V. Hajnal, and Daniel Rueckert. “Hierarchical Manifold Learning for Regional Image Analysis”. In: *IEEE Transactions on Medical Imaging* 33.2 (2014), pp. 444–461. DOI: `10.1109/tmi.2013.2287121`.

[46] Geoffrey H. Ball and David J. Hall. “A clustering technique for summarizing multivariate data”. In: *Behavioral Science* 12.2 (1967), pp. 153–155. DOI: `10.1002/bs.3830120210`.