Information Retrieval - CBIR Assignment - 03/2024

Football Kit CBIR

Universidad Politécnica de Madrid

Federico Paschetta, Cecilia Peccolo, Nicola Maria D'Angelo

federico.paschetta@alumnos.upm.es
cecilia.peccolo@alumnos.upm.es
nicolamaria.dangelo@alumnos.upm.es

Contents

Chapter 1

Motivation

In the dynamic world of sports, particularly football, fans share a deep connection with their favorite teams through various mediums, one of which is proudly wearing their team's kit. The fervor and passion exhibited by fans extend beyond the game itself, often manifesting in their choice of apparel. However, selecting the perfect football kit can be a daunting task. This is where Content-Based Image Retrieval (CBIR) technology steps in, offering a novel solution to simplify and enhance the kit selection process.

The proposed application will use the power of CBIR within a real-world domain, specifically tailored to assist football enthusiasts in finding kit similar to their desired choice based on color and texture. The application identifies jerseys that closely match the color palette and texture of the provided image from an extensive database that collects all football kit,even the historical ones. At the end we have implemented a QBE (Query by image) that can find similarities using the features of an image and we are talking about "One to Many" matching (Features are image properties that are present or absent). Image features can typically be divided in three different low-level features: Color, Texture, Shape.

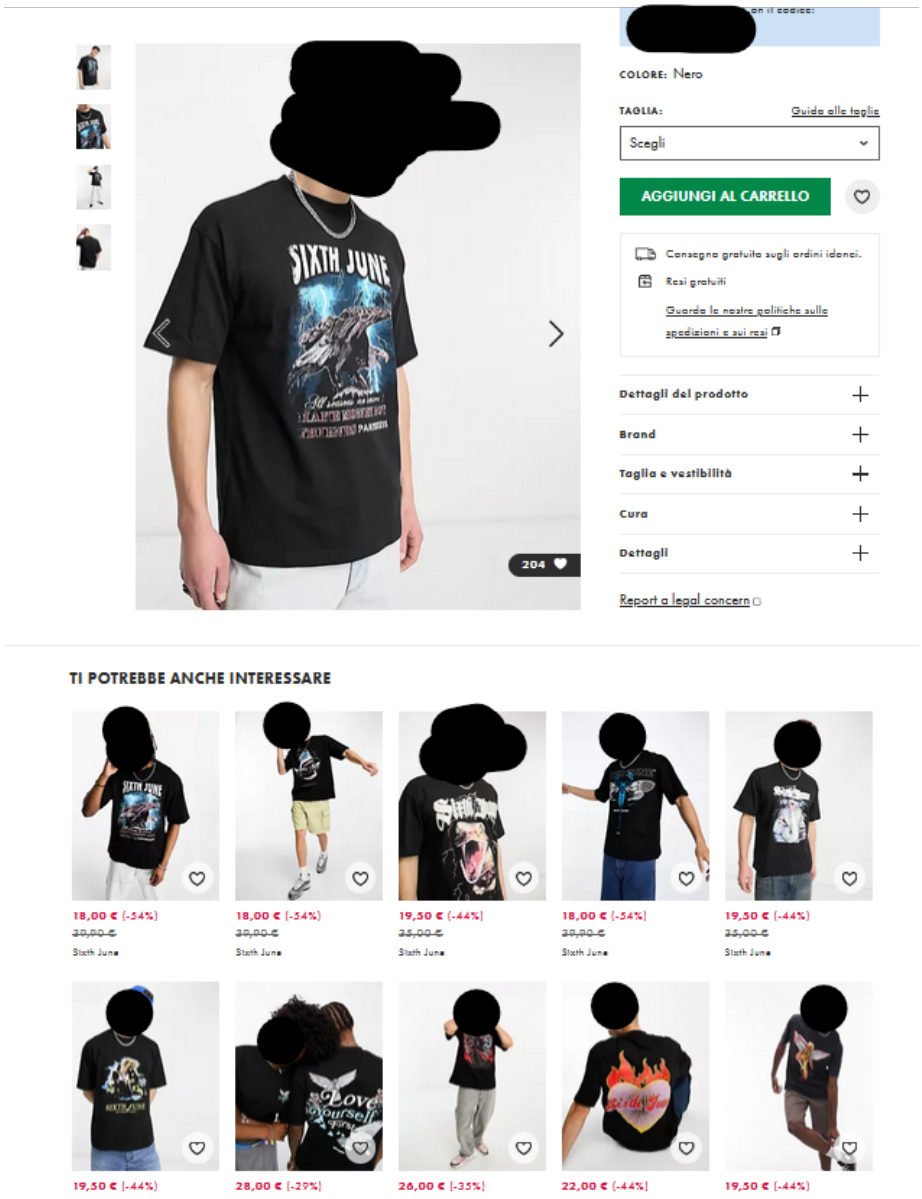

Figure 1.1: Online Shop Recommendation System Example: We took inspiration from online clothing shops, which once you choose a piece of clothing, offer you similar products using colour and texture. (An example on the following page)

Chapter 2

State of the Art

In the realm of content-based image retrieval (CBIR), significant advancements have been achieved in the techniques utilized for histogram comparison, which serve as summaries representations of image color distributions. Instead of just looking at the raw data, we're using smarter methods to understand patterns in these summaries better.Choosing the right color system is also important. Different color systems, like RGB or HSV, help us understand colors in different ways.
By using the right one, we can better capture what makes images similar or different. We've also gotten better at making descriptions of images. Instead of just listing basic features, we're now using deep learning to create more detailed descriptions that capture a lot of information about what's in the image. To measure how similar images are, we use something called distance. It's like measuring how far apart two images are in terms of their features. We've improved the ways we do this, using different formulas or methods that work well with the descriptions we've made. Overall, these improvements in comparing histograms, picking color systems, creating image descriptions, and measuring similarity have made it easier and more accurate to find the images we're looking for based on what they look like. These advances are making it possible to build smarter systems for searching images, which can be useful in many different areas. In order to solve this problem we got RGB histograms (with calcHist function from OpenCV) from images and then we computed their correlation, using NumPy corrcoef function, as OpenCV function returned NumPy arrays.

As another way to understand the T-shirts' characteristics, we looked at something called Haralick texture analysis. This method, well-documented in scientific literature, involves breaking down images into five key features: homogeneity, contrast, correlation, local homogeneity, and entropy. These features help us understand how pixel patterns vary across the T-shirts. Tommy Löfstedt, in his paper "Gray-level invariant Haralick texture features" [1], explored this method extensively. He showed how these features can tell us a lot about the texture of an image. We didn't stop at just extracting these features, though. We also interpreted them in the context of radiation physics to understand their trends better. To compare T-shirts and see how similar they are based on these features, we used a method called Euclidean distance. This helps us measure how much the features differ between T-shirts. And to make sure our comparisons are fair, we normalized the distances, taking into account differences in scale and magnitude.

Chapter 3

Implementation of CBIR

To implement our toy Content-Based Image Retrieval (CBIR) system, we opted for Python due to its robust computer vision library, OpenCV, and its ease of use. We began by acquiring a suitable dataset from Kaggle, which provided an Excel file containing over 1400 rows. Each row included a link to a high-quality photo of a kit with a white background. After importing this file into Python, we utilized pandas for manipulation. For each row in the Excel sheet, we downloaded the corresponding image and saved it in a folder named "Img".Our CBIR program prioritized clean and modular code, with a focus on refactoring processes into functions and leveraging functional programming paradigms. Consequently, our code consists of various functions, each serving a specific purpose. Functions like *read_image* retrieve an image object based on its index number, while *display_image* presents the image to the user. Additionally, we implemented functions essential for achieving our CBIR objectives. The CBIR algorithm initializes crucial variables, such as two empty dictionaries named *histogram_dict* and *texture_dict*, which store descriptor informations (about color and texture) for all images, and *end_num*, representing the index of the last image in our dataset. Next, we selected an image (in this case, image number 1373) to serve as the reference for background removal. Since the white background is consistent across images, we computed the number of white pixels in an image without white in the kit to adjust the histograms accordingly. This process, executed by the *get_white_pixels* function, ensures accurate comparison across images. Subsequently, we iterated through all images, creating an entry in both *histogram_dict* and *texture_dict* for each image and invoking functions to extract RGB histograms and Haralick texture features. These values were stored in the two dictionaries for further analysis. We then compared the selected image with all other images, calculating similarity values based on histogram correlation and Euclidean distance of texture arrays.The scores were merged using a weighted average, with a color-to-texture ratio of 90/10. We believe that kits are primarily characterized by color, and while unique textures may entail different colors, the information conveyed by the color histogram underscores the significance of color. Hence, we accorded greater importance to color in our analysis. Similarity values were stored in a dictionary and sorted in descending order.

Finally, the *get_top_values* function was called to display the x most similar kits to the user. This comprehensive approach ensures effective image retrieval within our dataset. *ordered_dict* contains kits ordered by similarity to the selected one to be compared, so it can be used to search for kits and, given the list of keys, at index i there is the i-th most similar kit.

Therefore, our steps to solve the problem and to create our CBIR have been:

- Getting color descriptor (RGB color histogram);
- Getting texture descriptor (Haralick feature array);
- Compare each image descriptor with selected image ones, using correlation for histogram and euclidean distance for Haralick array;
- Combine distances with an appropriate ratio (we chose 90/10 because of the importance of color in a kit, compared to texture);
- Sort images based on similarity.

To run the code, we utilized Google CoLab as it was the only platform where we could install the Mahotas package via pip, a necessity for computing texture values. To execute the code in this environment, it was necessary to include a *drive.mount()* call and place the dataset in the designated folder. Alternatively, one can simply specify the correct path to the dataset if running the code elsewhere. To enhance CBIR performance, we attempted histogram normalization using the *cv2.normalize* function. To test this version, simply replace calls to *load_imgs* with *load_img* in the main code. While our current results are acceptable, implementing this change should further improve performance. Unfortunately, due to limitations in Google CoLab's RAM size, we were unable to execute this experiment.

Bibliography

[1] Tommy Löfstedt. Gray-level invariant haralick texture features. *PLoS One*, 2019.