UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de un modelo y metodología para registrar patrones de diseños de moda a partir de imágenes y automatizar su replicación en nuevos productos

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Martinez Perez, Alvaro

Tutor/a: Morillas Gómez, Samuel

Cotutor/a: Jordan Lluch, Cristina

CURSO ACADÉMICO: 2021/2022

Resumen

Hoy en día, el sector textil se encuentra en revolución debido a que el modelo de fabricación está cambiando. En lugar de llevar a cabo el convencional proceso de producción de fabricación y venta, la industria actual está tan avanzada que permite producir ropa prácticamente “a la carta”. Mediante la compra online, los usuarios pueden solicitar prendas que todavía no han sido fabricadas, lo cual evitará que gran cantidad de unidades queden en los almacenes y sean desperdiciadas.
Por otro lado, se puede acceder a multitud de imágenes en diferentes redes sociales y webs donde se muestran productos de distintos diseñadores que siguen diferentes patrones de diseño.

Los dos objetivos principales que se han desarrollado en este proyecto consisten, por una parte, en diseñar un modelo, basado en la teoría de grafos, que permite representar la paleta de colores y la distribución espacial de los mismos de una determinada imagen, y por otra, un algoritmo para replicar el patrón de diseño extraído en una nueva imagen.

La solución propuesta se ha programado con Python, de forma que se han implementado técnicas de *machine learning*, algoritmos de detección de objetos y personas, de segmentación de imágenes y de etiquetado de regiones. Del análisis de un pequeño subconjunto aleatorio de soluciones, obtenemos la solución más acorde al patrón de diseño extraído.

Palabras clave: Algoritmo *K-Means*, paleta de colores, patrón de diseño, procesamiento de imágenes, segmentación, etiquetado de regiones.

Abstract

Today, the textile sector is undergoing a revolution because the manufacturing model is changing. Instead of carrying out the conventional production process of manufacturing and selling, today's industry is so advanced that it allows clothes to be produced practically "on demand". Through online shopping, users can order garments that have not yet been manufactured, which will prevent large numbers of units from sitting in warehouses and going to waste.
On the other hand, a multitude of images can be accessed on different social networks and websites showing products from different designers that follow different design patterns.

The two main objectives that have been developed in this project are, on the one hand, to design a model, based on graph theory, that allows to represent the color palette and the spatial distribution of the colors of a given image, and on the other hand, an algorithm to replicate the extracted design pattern in a new image.

The proposed solution has been programmed with Python, so that machine learning techniques, object and person detection algorithms, image segmentation and region labeling have been implemented. From the analysis of a small random subset of solutions, we obtain the solution most in line with the extracted design pattern.

Keywords: K-Means Algorithm, colour palette, design pattern, image processing, segmentation, region labeling.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1 Motivación

Una de las preguntas más frecuentes en la industria textil y en la moda es cómo hacer patrones de ropa. Por patrón de ropa entendemos un molde en papel o una versión digital que permitirá confeccionar cualquier prenda de vestir que queramos, y únicamente dependerá de la forma en la que se haya diseñado. Una ventaja de la versión digital frente a los realizados en papel es que los patrones de moda digitales son muy completos y actualmente es posible obtenerlos gratuitamente mediante PDF.

Uno de los aspectos del día a día en el que se ven afectados dichos patrones de moda surge cuando una marca famosa lanza una nueva temporada de prendas y accesorios, y nos llama la atención cierta combinación de colores. Por nuestra mente pasa el deseo de conseguir dicha prenda que nos ha gustado con tan solo verla, o bien de poder tener alguna similar, con esa combinación de colores en la que nos hemos fijado.

En la actualidad, hay muchas webs que ofrecen la posibilidad de diseñar ciertos patrones de moda y descargarlos. Pero si hablamos de extraer un patrón de diseño y trasladarlo a una prenda, queda cierto camino que recorrer. Por ello, una de las justificaciones de la elaboración de este proyecto es sacar adelante una solución basada en ciencia de datos que ayude a realizar lo anterior.

Por otro lado, las técnicas de ciencia de datos también permiten elaborar soluciones para el problema medioambiental que acarrea la industria textil, por ejemplo, como es nuestro objetivo, reduciendo la cantidad de ropa que queda en stock en los grandes almacenes y es desperdiciada.

Este trabajo final de grado resulta de mi interés debido a que tiene cierta parte de teoría de grafos, la cual me parece muy interesante, además de que consta en su mayoría sobre procesamiento de imágenes, que es algo que me llama mucho la atención pero que desgraciadamente apenas hemos estudiado en el grado.

1.2 Objetivos

Una de las ideas principales de la ciencia de datos es aplicar técnicas de *machine learning* a cantidades masivas de datos. El objetivo general consiste en extraer conocimiento e información de dichos datos, y para lograrlo se han establecido los siguientes objetivos personales:

- Mejorar y ampliar los conocimientos en el análisis y tratamiento de imágenes.
- Ser capaz de implementar un programa útil para procesar diversidad de imágenes.
- Emplear una metodología eficiente, justificada y basada en los conocimientos adquiridos durante el grado.

El objetivo principal de este proyecto consiste en aplicar conocimientos adquiridos sobre *data science* a imágenes de tendencias de moda para extraer las combinaciones de colores utilizadas, y ser capaz de replicar dicha tendencia en otras prendas. Por ello, se han definido los siguientes objetivos específicos:

- Aplicar el algoritmo *K-Means* a fin de simplificar los datos y poder trabajar cómodamente con ellos.
- Implementar algoritmos de detección de personas para que este proyecto pueda ser aplicable no sólo a imágenes de prendas, sino también en casos donde aparecen modelos.
- Extraer la paleta de color de cada imagen.
- Representar las imágenes procesadas mediante una matriz de pesos y reflejarlas mediante un grafo (extraer y representar un patrón de diseño),
- Replicar el resultado anterior en una imagen segmentada en regiones, es decir, replicar un patrón de diseño.
- Evaluar, seleccionar y justificar la mejor solución dada una imagen original, es decir, la mejor forma de colorear una prenda segmentada tras haber obtenido diferentes posibles combinaciones.

1.3 Impacto esperado

En primer lugar, decir que este TFG es concertado, de tal forma que la idea original que tenía en mente era desarrollarlo sobre temas relacionados con grafos, puesto que dicha teoría de grafos ha sido algo que me ha resultado muy interesante de conocer y estudiar. Junto a otras tareas recomendadas por mis tutores y en relación a un tópico muy común y llamativo como son las tendencias de moda, supe que podría llegar a elaborar un TFG entretenido, útil para mi aprendizaje y adecuado para mostrar los conocimientos adquiridos, no sólo como científico de datos, sino como estudiante (casi) graduado de la universidad.

Por tanto, como estudiante del grado Ciencia de Datos, espero mostrar la capacidad de solucionar problemas que aparecen en la vida cotidiana, además de buscar una forma eficiente de hacerlo. Para ello, es necesaria una buena organización y estructuración de todo lo que se va a realizar desde el primer momento, pues de ello dependerá la calidad de este proyecto. En este apartado se espera reflejar los conocimientos adquiridos en procesamiento de imágenes, trabajar con inmensas matrices de datos, algoritmos de *clustering*, compresión de grafos y demás actividades que se realizarán durante este TFG.

El lenguaje de programación utilizado para la elaboración de este proyecto ha sido Python, a partir del conocimiento adquirido durante las diferentes asignaturas del grado. Además de su potencia y sencillez, ofrece muchas y muy buenas opciones para trabajar el procesamiento de imágenes. Mis objetivos al emplear este lenguaje son elaborar un buen proyecto, obteniendo resultados eficientes y progresar en el autoaprendizaje, ya que el uso que se le puede dar a un lenguaje de programación va mucho más allá de lo que se puede aprender en un grado como el de Ciencia de Datos.

Por otra parte, durante el grado se han elaborado muchos proyectos de alta complejidad y duración, lo cual nos ha dotado de experiencia para orientar y dar soluciones a los diferentes problemas y retos propuestos, por lo que también se espera mostrar la habilidad y madurez adecuada en aspectos como documentación, organización, redacción, expresión, justificación por un lado, manejo, tratamiento, compresión y análisis de datos por otro; y capacidad de hacer frente a situaciones complejas desde el punto de partida.

1.4 Metodología

En primer lugar, tras haber recibido la información y las pautas necesarias para comenzar el trabajo, se elaboró una estructura organizativa en Trello con el objetivo de planificar las tareas secuencialmente y así poder llevar un orden de trabajo sin pausa pero sin prisa. La organización del tablero es muy simple, consta de cinco columnas (Apartados redactados, Apartados por redactar, Tareas programadas, Tareas por programas y Tareas pospuestas) en las que, conforme se va avanzando y progresando el proyecto, se van añadiendo etiquetas con tareas marcadas con más o menos necesidad de ser realizadas o mejoradas.

El propio proyecto en sí se va a explicar en el apartado 5.Solución propuesta y se expondrá en el apartado 7.Despliegue, donde mediante diferentes imágenes se mostrará todo el procedimiento realizado. Para ello, es necesario haber realizado correctamente los anteriores apartados 2, 3 y 4, que constan del estado del arte, análisis del problema y tratamiento de los datos, respectivamente. Con ellos se ofrecerá al lector la justificación de la utilidad y necesidad de dar una solución a este problema, una explicación de las posibles técnicas a aplicar para desarrollar el proceso y de los conocimientos necesarios para entender los motivos de los pasos seguidos, así como información de las imágenes desde un punto de vista centrado en los datos.

1.5 Estructura

En la primera parte de este proyecto, se introducen las entidades e investigaciones relacionadas con el tema tratado, y cuál es la actividad o posible alternativa a este proyecto. A continuación, se procede a desarrollar el análisis del problema, donde se muestra documentación y conocimientos teóricos y matemáticos sobre las técnicas empleadas (algoritmo *K-Means*, coordenadas de colores, teoría de grafos, segmentación de imágenes o etiquetado de regiones, entre otras).
A continuación, se procede a explicar la solución propuesta, detallando cada una de las diferentes fases en las que se ha dividido el proyecto y comentando cada una de las funciones implementadas en Python. Se hace un breve análisis de eficiencia algorítmica, del marco legal y ético y de posibles riesgos que puedan aparecer.
Finalmente, se realiza una implantación de la solución propuesta donde se explica todo el proceso con una imagen de partida de la que se extrae el patrón de diseño, y otra sobre la cual se va a replicar.

2. Estado del arte

La *e-commerce StyleSage* [1] se dedica al análisis de tendencias de moda (ver qué combinaciones de colores son las más preferidas). Permite crear aquellos productos que los clientes realmente quieren a través de herramientas de Inteligencia Artificial. Trabajan con datos en tiempo real, lo que permite satisfacer al mercado y obtener una mayor rentabilidad. Además, proporcionan a los usuarios información sobre las tendencias de última hora y adaptan las ofertas para satisfacer sus necesidades.

Su actividad se centra en *crawler* alrededor de mil páginas web de marcas de ropa, de las cuales extraen, para cada cada producto, atributos como precio o descuento, categorizándolos por tipo de prenda, materiales que los componen, tamaño o color. La información normalizada y categorizada es introducida en su base de datos. La idea central es que el cliente seleccione varios productos, y la aplicación muestre una rueda de color con la combinación de colores generada. Por tanto, el objetivo a abordar [2] se divide en dos partes:

1) Extraer el color dominante (RGB) de cada producto: No se busca el color más repetido porque hay sombras y ciertas partes que no interesan. El primer paso es eliminar el fondo (*fulfill*), con lo que se aísla bastante el modelo. Posteriormente, se intenta detectar la piel de la persona (con *OpenCV*). Finalmente, los píxeles blancos (fondo y piel) se excluyen y se quedan con el centro (ropa).
Como técnica de *machine learning*, aplican el algoritmo *K-Means*, al igual que en este proyecto. En el caso de los colores, convierten las imágenes al espacio de colores L*A*B (que es el que más se parece a nuestra percepción de los colores), a diferencia del espacio RGB, con el que se trabaja en este proyecto.

2) Una vez obtenido el color en coordenadas RGB de los productos seleccionados, el siguiente paso consiste en extraer la rueda de color, aplicando el algoritmo *K-Means* con más *clusters* (unos 18 colores, de forma que más o menos se cubre el espectro). Esta parte permite comprobar a *StyleSage* que en sus patrones de colores aparezcan gamas de todo tipo, para cubrir todas las preferencias posibles. Esta tarea sobrepasa los objetivos de este proyecto, pero podría ser interesante realizarla en un futuro.

A nivel de estudios e investigaciones científicas, Stephen Westland, profesor de Ciencia del color y tecnología de la Universidad de Leeds, ha llevado a cabo un estudio en el que aplica *machine learning* para extraer paletas de color de imágenes de desfiles de moda [3]. El experimento consiste en analizar 48 imágenes en RGB, a partir de una selección propuesta por los participantes (22 en total). Cada participante tenía que seleccionar, desde su punto de vista, 3 colores con los que aproximadamente se represente la totalidad de la imagen. Por tanto, para cada imagen se obtiene una paleta de 66 colores (3 colores * 22 participantes), con la que se evaluará la calidad de los modelos de *machine learning* implementados. Para cada imagen, se aplican 2 modelos:
1) El primero consiste en aplicar el algoritmo *K-Means* con 4 centroides a las imágenes originales, sin procesar los fondos de imagen. En otras palabras, es aplicar *K-Means* a toda la imagen.

2) El segundo modelo trata sobre procesar cada imagen de forma que se aplica un algoritmo de detección de personas. A continuación, se usa un algoritmo de detección de *foreground*, que diferencia los píxeles del primer plano y los del fondo. Finalmente, se implementa el algoritmo *K-Means* con 3 centroides a los píxeles del primer plano, es decir, de la figura humana. Este modelo representa una idea aproximada de la que se pretende seguir en este proyecto.
Los resultados de este experimento muestran que el segundo modelo predice los colores con mayor precisión, de forma que la paleta obtenida mediante este modelo representa mejor la imagen.

Los siguientes análisis o estudios son posibles alternativas que se podrían realizar junto a este proyecto:

- Un análisis más profundo y específico de la comparación entre paletas de colores mediante métodos de escalamiento psicofísico, como el *Magnitude Estimation* (ME) [4].

- A pesar de alejarse del tema principal de este TFG, también sería posible hacer un análisis visual de la moda en la actualidad, utilizando métodos para representar la apariencia de la ropa y medir la similitud entre las prendas. Ello requeriría la aplicación de algoritmos de detección y clasificación de las prendas, así como de sus estilos [5].

- También se podría dar un enfoque más relacionado con la predicción de los estilos y las modas más populares en un futuro. Para ello, se analizan las tendencias de los años más recientes y se aplican técnicas como métodos *naive*, autorregresión y redes neuronales [6].

- Por otro lado, al igual que en este TFG nos hemos centrado en el algoritmo *K-Means* como técnica de *machine learning* para predecir los colores, también sería posible hacer este estudio mediante redes neuronales [7]. Para ello, las imágenes se transforman en una *DenseNet* y se seleccionan las 10 primeras clases predichas de cada imagen, en función de sus probabilidades de clasificación. Se emplean redes neuronales recurrentes (RNN) y *Long-Short Term Memory* (LSTM), que son un tipo de RNN que amplían su memoria para aprender sobre sucesos del pasado. Se podría utilizar el *area under the curve* (AUC) como métrica de evaluación de los modelos.

3. Análisis del problema

3.1 Conocimientos teóricos

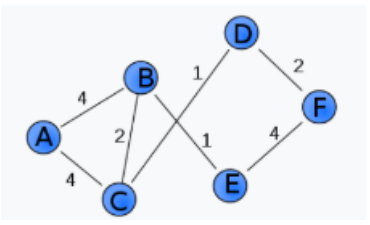
3.1.1 Teoría de grafos

Los grafos permiten representar diversas situaciones y son excepcionalmente útiles en situaciones complejas, por lo que es muy común la aplicación de análisis de grafos en estudios de ciencias exactas, ciencias sociales y en aplicaciones informáticas. Los grafos pueden ser dirigidos o no dirigidos.

Un grafo no dirigido *G* es una pareja de conjuntos (*V,E*) donde *V* es distinto de vacío, y *E* es un conjunto de pares no ordenados de elementos de *V*. A los elementos de *V* se les llama vértices, y a los de *E*, aristas. Dada la arista *e*=(*u,v*), *u* y *v* se dice que son los extremos de *e*, y que *e* es incidente en ellos. Dos vértices unidos por una arista se dice que son adyacentes, mientras que dos aristas se dicen adyacentes cuando comparten un extremo. Las aristas en las que los extremos coinciden se llaman bucles [8].

En el caso de que los pares de *E* sean ordenados, el grafo se dice que es dirigido, y los elementos de *E* se denominan arcos. A los extremos de un arco (*u,v*) se les llama extremo inicial y final, respectivamente, y la representación gráfica es una flecha de *u* hacia *v*. Si el grafo contiene pesos en sus aristas, se denomina grafo ponderado, y es un caso particular de los grafos dirigidos y no dirigidos [8].

Todo grafo de *n* vértices, dirigido o no dirigido, viene definido por una matriz de adyacencia de *C* dimensiones en la que un 1 en la posición *(i,j)* indica una arista (o arco) de *i* a *j*, y un 0 la ausencia de esta. Si el grafo es no dirigido, la matriz es simétrica. Si el grafo es ponderado, sustituyendo los unos de la matriz de adyacencia por dichos valores, que llamamos costes o pesos, obtendremos lo que denominamos matriz de pesos o de costes asociada. En la figura 1 se observa un grafo no dirigido ponderado junto a su matriz de pesos.



$$\begin{bmatrix} 0 & 4 & 4 & 0 & 0 & 0 \\ 4 & 0 & 2 & 0 & 1 & 0 \\ 4 & 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 2 & 4 & 0 \end{bmatrix}$$

Figura 1. Grafo no dirigido y ponderado junto a su matriz de pesos. Disponible en https://es.wikipedia.org/wiki/Matriz_de_adyacencia

3.1.2 Aplicaciones biyectivas

Una aplicación *f* de *A* en *B* es una correspondencia tal que a cada elemento del conjunto inicial *A* le corresponde uno y solo un elemento del conjunto *B*, es decir, cada elemento tiene una sola imagen [9].
La imagen de una función *f(x)* es el conjunto de valores que toma la variable independiente *x*. Se denomina antiimagen de un elemento *b* de *B* al conjunto de elementos de *A* cuya imagen es *b* [10].

Las aplicaciones pueden ser inyectivas, sobreyectivas o biyectivas.

- Una aplicación *f: A → B* es inyectiva cuando todos los elementos de A tienen imágenes distintas.
- Una aplicación *f: A → B* es sobreyectiva cuando cada uno de los elementos de B tiene al menos una antiimagen.
- Una aplicación *f: A → B* es biyectiva cuando es inyectiva y sobreyectiva.

En la figura 2 se muestra una aplicación biyectiva.

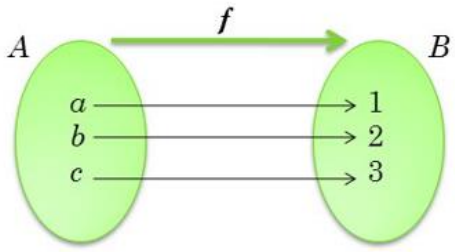


Figura 2. Aplicación biyectiva [9]

3.1.3 Distancia euclídea

La distancia euclídea entre dos puntos *x* e *y* pertenecientes a $\mathbb{R}^n$ ubicados en una recta se define como la raíz cuadrada de la suma de los cuadrados de las diferencias de sus coordenadas:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Dado que en este proyecto se van a comparar matrices, necesitamos obtener el error existente. Sean dos matrices de las mismas dimensiones, *A* y *B*, donde

$$A = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \quad y \quad B = \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

La distancia euclídea entre ambas matrices se calcula como:

$$d(A,B) = \sqrt{\sum_{i=1}^{4}(a_i - b_i)^2}$$

3.1.4 Problemas de clasificación. Algoritmo *K-Means* y su aplicación a imágenes.

En la primera parte de este proyecto nos encontramos ante un problema de clasificación [11], donde se van a agrupar los colores a partir de observaciones realizadas con datos previos. Para dicha clasificación se va a aplicar la técnica de cuantificación, que consiste en agrupar todo un rango de valores en uno solo. Al cuantificar el color de una imagen, estamos reduciendo el número de colores necesarios para representarla. Una ventaja es que el tamaño del fichero de la imagen disminuye. Se pueden considerar tres tipos de clasificación de los datos:

- **Clasificación supervisada:** disponemos de un conjunto de datos (datos de entrenamiento) donde cada uno tiene una etiqueta asociada. Con estas etiquetas se construye un modelo que dirá si la clasificación es correcta o no. Una vez construído el modelo, se puede usar para clasificar nuevos datos que no estén etiquetados.

- **Clasificación semisupervisada:** algunos datos de entrenamiento tienen etiquetas, pero no todos. Esto es típico en clasificación de imágenes, donde es habitual disponer de muchas imágenes, la mayoría de ellas no etiquetadas.

- **Clasificación no supervisada:** los datos no tienen etiquetas y se clasifican a partir de sus propiedades o características. Es el caso del algoritmo *K-Means* implementado en este proyecto para clasificar los colores de los píxeles de las imágenes.

El algoritmo *K-Means* [11], se encuentra en el top 10 de algoritmos de minería de datos. Este algoritmo de clasificación no supervisada establece una partición de un conjunto de puntos de $\mathbb{R}^n$ que consiste en *k* bloques denominados *clusters*, de forma que los puntos que pertenecen al mismo bloque tienen un alto grado de similitud, y los que pertenecen a distintos bloques difieren entre sí.

Se basa en tres pasos, en los que la distancia utilizada es la cuadrática [12]:

1. **Inicialización:** se escoge el número de grupos, *k*, y se establecen *k* valores aleatorios en el espacio de los datos, a los que llamaremos centroides.

2. **Asignación objetos a los centroides:** cada objeto de los datos es asignado a su centroide más cercano.

3. **Actualización centroides:** se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo.

El conjunto de objetos a *clusterizar* $S = \{u_1, \ldots, u_m\}$ es un subconjunto de $\mathbb{R}^n$. El algoritmo comienza con una selección aleatoria de un conjunto de *k* puntos $c_1, \ldots, c_k$ en $\mathbb{R}^n$ llamados centroides. Se computa una partición inicial del conjunto de objetos *S* asignando cada objeto u_1 a su centroide más cercano c_j [13].

Sea C_j el conjunto de puntos asignado al centroide c_j. El asignamiento de los objetos (o datos) a los centroides se expresa mediante una matriz $B = (b_{ij}) \in \mathbb{R}^{m\,x\,k}$, donde

$$b_{ij} = \begin{cases} 1 \; si \; u_i \in C_j \\ 0 \; en \; otro \; caso \end{cases}$$

Al asignar cada objeto exactamente a un solo *cluster* tenemos $\sum_{i=1}^{k} b_{ij} = 1$. Por otro lado, $\sum_{i=1}^{m} b_{ij}$ se corresponde con la cantidad de objetos asignados al centroide c_j. Tras la asignación de los datos, el centroide c_j se calcula de nuevo utilizando la siguiente expresión:

$$c_j = \frac{\sum_{i=1}^{m} b_{ij} u_i}{\sum_{i=1}^{m} b_{ij}}, \quad \text{para } 1 \leq j \leq k.$$

La matriz de centroides, $C = (c_1, \ldots, c_k) \in \mathbb{R}^{n \, x \, k}$ puede escribirse como

$$C = (u_1, \ldots, u_m) \, Bdiag \left(\frac{1}{m_1}, \ldots, \frac{1}{m_k}\right) = X'B \left(\frac{1}{m_1}, \ldots, \frac{1}{m_k}\right)$$

donde $m_j = \sum_{i=1}^{m} b_{ij}$ es el número de objetos del *cluster* C_j.

La suma de los errores cuadráticos de una partición $\pi = \{u_1, \ldots, u_m\}$ de un conjunto de objetos *S* mide la calidad de la asignación de los objetos a los centroides y se define como

$$sse() = \sum_{j=1}^{k} \sum_{u \in c_j} d^2(u, c_j)$$

donde c_j es el centroide de C_j para $1 \leq j \leq k$ [13].

El algoritmo *K-Means* resuelve un problema de optimización, donde la función a optimizar (minimizar en este caso) es la suma de las distancias cuadráticas de cada objeto al centroide de su *cluster*. En otras palabras, el objetivo de este algoritmo es seleccionar los centroides que minimizan la inercia o la suma de cuadrados intra-*cluster* [12]. Su popularidad se debe en cierta parte a su simplicidad, escalabilidad, velocidad de convergencia y poca complejidad temporal.

El algoritmo *K-Means* es sencillo y rápido, pero, como hemos dicho, depende del valor de *k* y de los centroides elegidos en la inicialización. Suele converger a un mínimo local, aunque ello dependerá de la inicialización mencionada. Como consecuencia, el cálculo suele realizarse varias veces, con diferentes inicializaciones, para así poder comparar y elegir el mejor resultado.

3.1.5 Definición de color, píxeles y propiedades

El color es una sensación creada a partir de la excitación del sistema de visión debido a la luz [14]. En otras palabras, el color es el resultado de percibir la luz en la región visible del espectro electromagnético, cuando incide sobre la retina del ojo humano.

La retina humana tiene tres tipos de células fotorreceptoras de color llamadas conos, los cuales responden a la radiación luminosa con curvas de respuesta espectral diferentes. Debido a la existencia de estos tres tipos de células, se requieren tres componentes numéricas para describir un color. Por tanto, un color puede definirse con un vector de tres componentes. El conjunto de todos los colores forma un espacio de vectores llamado espacio de color o modelo de color. De esta forma, los tres componentes de un color pueden definirse de varias formas diferentes dando lugar a varios espacios de colores [15].

Para comprender este proyecto desde el principio, lo mejor es comenzar explicando la definición de píxel. Un píxel [16] es el punto más pequeño que forma parte de una imagen. Todos los píxeles que componen la imagen trabajan de forma independiente (a pesar de que parece que estén juntos), y cada uno representa un número determinado de bits con el que se conseguirá un color diferente. Por ejemplo, en el caso del modo de color de 8 bits, se usarán 8 bits para representar cada píxel y se mostrarán $2^8 = 256$ colores diferentes. La calidad final de la imagen dependerá del número de bits que se usan para su representación, por lo que a mayor cantidad de píxeles, mejor calidad.

Un píxel *p* en las coordenadas *(x,y)* tiene cuatro vecinos horizontales y verticales y cuatro diagonales. A este conjunto de píxeles se le denomina vecinos 8-conectados de *p*, tal y como podemos ver en la figura 3, y sus coordenadas son las siguientes:

(x+1,y), (x-1,y), (x,y+1), (x,y-1), (x+1,y+1), (x+1,y-1), (x-1,y+1), (x-1,y-1)

Si algún vecino queda ubicado fuera de la imagen, una posible solución es asignar 0 al valor que le correspondería al píxel [17]. En este proyecto, debido a que uno de los objetivos es comparar la cantidad de píxeles de un color próximos a los píxeles con otro color, se va a utilizar dicha vecindad 8-conectada. En la figura 3 se representan los 8 vecinos del píxel con coordenadas *(x,y)*.

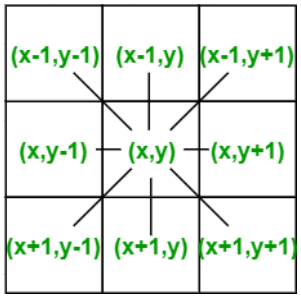


Figura 3. Representación de un píxel junto a sus 8 vecinos [15]

3.1.6 Espacios de color RGB, HSV y hexadecimal

Un espacio de color es cualquier organización de colores que permita reproducirlos de forma similar a la existente en la naturaleza. Los colores existen de forma difusa, es decir, no se puede saber dónde termina uno y comienza el siguiente [18]. En un espacio de color, esta gama de colores está acotada en un espacio discretizado generalmente en tuplas o vectores que contienen la información del color representado [19].

El modelo de color RGB es capaz de formar todos los colores a partir de combinaciones de los colores rojo, verde y azul. Estos colores son los tonos que se aprecian en los elementos que funcionan a través de emisiones luminosas como pantallas de televisores, teléfonos móviles o monitores. Representan la síntesis aditiva de color, que se refiere a la formación de colores a través de la suma de diversas longitudes de onda, es decir, luz. Al combinar estos tres colores en partes iguales, se obtiene el color blanco. En su ausencia, surge el color negro, porque el ojo humano no es capaz de reconocer ningún color sin luz. Cada uno de los colores primarios usados en el modelo RGB utilizan 8 bits, con valores que van desde 0 hasta 255 según el sistema binario. Por tanto, existen un total de $256^3 = 2563$ colores diferentes. En la figura 4 se muestran las coordenadas RGB de algunos colores.

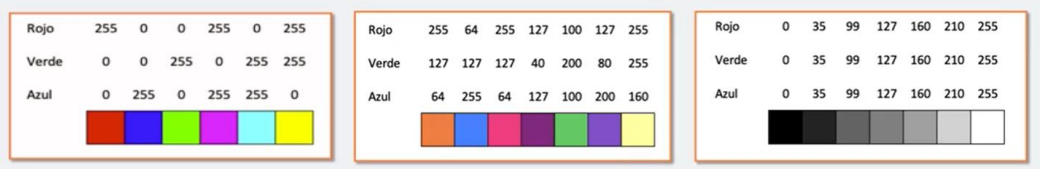


Figura 4. Colores formados según la combinación de las coordenadas RGB [20]

El espacio RGB se basa en el sistema de coordenadas cartesiano donde los colores son puntos definidos por vectores con extremo inicial en el origen [20]. El negro está en el origen [0,0,0] y el blanco está ubicado en la esquina opuesta al origen [1,1,1], tal y como se puede ver en la figura 5.

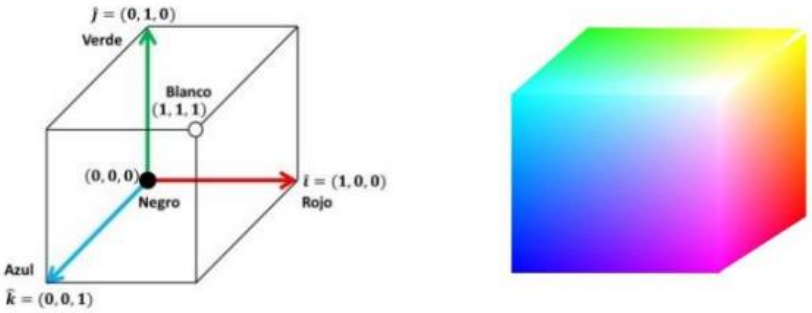


Figura 5. Espacio de color RGB. Disponible en https://carlosprivitera.blogspot.com/2017/10/cubo-de-colores-rgb.html

El color de un pixel p se escribe como una combinación lineal de los vectores base de verde, rojo y azul [20]:

$$\phi_p = r_p\vec{i} + g_p\vec{j} + b_p\vec{k}$$

Donde r_p, g_p y b_p son los componentes de rojo, verde y azul, respectivamente. La orientación y magnitud de un vector de color define la cromaticidad e intensidad del color, respectivamente [20].

El espacio de color HSV [21] es una representación tridimensional del color basado en los componentes de matiz, saturación y brillo o valor (*hue, saturation* y *value* en inglés).

- El tinte o matiz es la cualidad que se puede clasificar como verde, azul, rojo, etc. Se define entre 0º y 360º.

- La saturación indica la intensidad de un matiz específico, dado entre 0 y 1, donde 0 no representa saturación (blanco), mientras que 1 sería el matiz en toda su intensidad. A mayor saturación, el color será más vivo.

- El brillo muestra el nivel de luminiscencia entre 0 y 1. A menor brillo, el color tenderá a negro.

En este espacio, el color de un pixel p se representa por sus componentes de tono (h), saturación (s) e intensidad (v):

$$\varphi_p = [h_p, s_p, v_p]$$

Mediante la figura 6 se puede ver que el espacio de color HSV muestra la apariencia del espacio HSV. Los rangos de valores reales del tono, saturación e intensidad son [0,2π], [0,1] y [0,255], respectivamente.

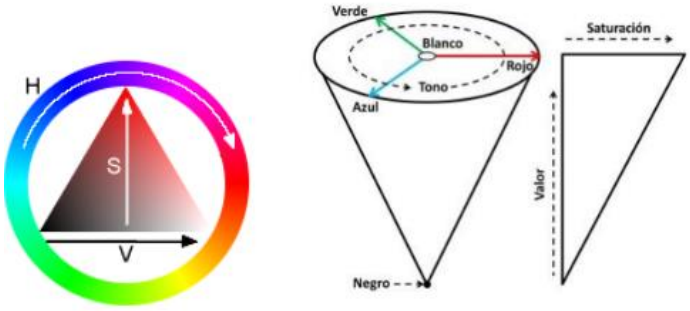


Figura 6. Espacio de color HSV. Disponible en https://es.wikipedia.org/wiki/Modelo_de_color_HSV

En el modelo RGB, las coordenadas son euclidianas, mientras que el color HSV se representa mediante coordenadas cilíndricas (forma de identificar un punto en el espacio tridimensional colocado en la superficie lateral de un cilindro cuya base está en el plano *OXY*, su centro es el origen de coordenadas y tiene un radio determinado). En otras palabras, la naturaleza de los colores no es lineal [22]. Además, es una representación más cercana a la forma en que los humanos perciben los colores y sus propiedades, pues los colores se agrupan según las tonalidades.

El sistema de color hexadecimal es un código de 6 símbolos basado en el modelo RGB. Está formado por tres elementos de 2 símbolos, donde el primero es el valor rojo, el segundo el verde y el tercero el azul. Cada uno de ellos expresa un valor de color de 0 a 255. Este sistema de colores se forma a partir del sistema de numeración en base 16, el cual permite mostrar 16^6 = 16.777.216 colores diferentes, muchos más de lo que el ojo humano es capaz de percibir. El color negro, con las coordenadas RGB = (0,0,0) se presenta como #000000 en hexadecimal; mientras que el color blanco, (255,255,255), sería #FFFFFF.
La principal diferencia entre el modelo RGB y el hexadecimal, es que el RGB es utilizado por televisores, pantallas de ordenador, cámaras digitales, teléfonos, proyectores de vídeo, etc, mientras que el hexadecimal es el lenguaje visual de la web y es utilizado en la mayoría de aplicaciones digitales.

3.1.7 Detección de personas en imágenes

Existen diferentes tipos de algoritmos de detección de personas: Algunos emplean la técnica de eliminación del fondo (*Background Subtraction*) mediante algoritmos de *machine learning* o de *deep learning,* mientras que otros se basan únicamente en *deep learning* y se centran en predecir los recortes sobre las personas en función del *dataset* sobre el que se realizará el aprendizaje. Estos últimos están basados en el aprendizaje de redes neuronales.

- Técnica de Sustracción de fondo o BGS (*Background subtraction*):

Consiste en obtener el fondo o segundo plano de una imagen (que contiene los elementos estáticos) para restarle cada fotograma y obtener el primer plano, el cual contiene los elementos dinámicos de la imagen. El resultado suele ser una imagen con el fondo representado por el color negro, pero también existe la posibilidad de indicar el color que se le quiere asignar a este, para que este no interfiera con los colores del objeto central. En este proyecto vamos a emplear esta técnica.

- Detección de personas con 'BGS' + "extracción de características HOG & LBP" + SVM:

La imagen original va a ser introducida en una máquina de vectores soporte (SVM), y como sabemos que el rendimiento de los algoritmos de *machine learning* se ve reducido cuando hay alta dimensionalidad, es necesario preprocesarla mediante los algoritmos de *Histogram Oriented Gradients* (HOG) y *Local Binary Patterns* (LBP) con el objetivo de reducir la cantidad de información.

La técnica HOG [23 y 24] consiste en descomponer la imagen en pequeñas celdas, calcular el vector gradiente de los píxeles de cada celda, crear un histograma por cada celda con los valores calculados, normalizar y obtener un descriptor para cada celda.

La técnica de *Local Binary Patters* [25] consiste en convertir la imagen a escala de grises, calcular la máscara LBP (comparativa de un píxel con los valores de sus vecinos que permite obtener texturas en una imagen), crear un histograma y normalizar.

Una vez que se han extraído las características HOG y LBP, se concatenan en un vector y se introducen en el *Support Vector Machine*, de forma que se construye un modelo SVM que será capaz de predecir si un píxel pertenece al primer o segundo plano de la imagen. Además de BGS y 'BGS' + "extracción de características HOG & LBP" + SVM, también se emplean muchas otras técnicas combinadas con redes neuronales, siendo sobre todo de gran utilidad las redes convolucionales, pero todo ello queda como posible alternativa a lo tratado en este proyecto.

3.1.8 Segmentación de imágenes y etiquetado de regiones

La segmentación de imágenes consiste en subdividir una imagen en los objetos o subregiones que la constituyen [26]. El nivel de detalle de cada subdivisión depende del problema a resolver. Los algoritmos de segmentación de imágenes se basan en dos

propiedades relacionadas con la intensidad de los píxeles: la discontinuidad y la similitud. Con la discontinuidad se separan los objetos requeridos, mientras que con la similitud se pretende separar una imagen en regiones que se asemejan según una serie de criterios establecidos previamente.

Existen diferentes problemas especializados, como segmentación por color, por texturas, segmentación semántica o superpíxel [27], así como una amplia variedad de algoritmos de segmentación de imágenes, como la umbralización, algoritmo de *Watershed*, *GrabCut*, *Livewire*, *Mask R-CNN*, etc. En este proyecto se emplea la técnica de umbralización o *thresholding*, que consiste en un tipo particular de segmentación por color con solo dos categorías, claro y oscuro [27]. Cada píxel se clasifica como claro u oscuro comparando su intensidad con un umbral.

El objetivo de la segmentación es localizar regiones con significado y se usa tanto para localizar objetos como para encontrar sus bordes dentro de una imagen. Como resultado se obtiene un conjunto de segmentos que cubren toda la imagen sin superponerse, el cual puede representarse como una imagen de etiquetas.

3.2 Software empleado

Este trabajo de fin de grado está programado en Python prácticamente en su totalidad, debido a que dicho lenguaje de programación ha sido el más utilizado a lo largo del grado de Ciencia de Datos, además de que ofrece gran variedad de librerías con algoritmos ya implementados que facilitan la resolución de tareas, siendo relativamente sencillo de utilizar en comparación con otros lenguajes. Concretamente se han empleado las siguientes librerías:

- *Numpy*: Está especializada en el cálculo numérico y el análisis de grandes volúmenes de datos. Incorpora una clase de objetos llamados *arrays* que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y dispone de funciones muy eficientes para su manipulación. Se ha usado principalmente en la lectura matricial de las imágenes y la creación de la matriz de pesos.

- *Pandas*: Está especializada en el manejo y análisis de datos, contando con tres estructuras de datos diferentes: series (con una dimensión), *dataframes* (tablas de dos dimensiones), y estructuras de tres dimensiones o cubos. Su uso se limita a la creación de la matriz de pesos en forma de *dataframe*, de manera que podemos acceder mediante los índices o nombres de las filas y columnas.

- *Matplotlib*: Se centra en crear y personalizar gráficos en dos dimensiones. Se emplea en representación visual de las imágenes, especialmente mediante la función *imshow()*.

- *Sklearn*: Es una librería especializada en aprendizaje automático, que cuenta con diferentes algoritmos de clasificación, regresión, *clustering*, reducción de dimensionalidad, selección de modelos o preprocesamiento. Se ha utilizado

exclusivamente para la programación del algoritmo *K-Means*, donde la propia función llamada *KMeans()* contiene internamente dicho algoritmo. También se emplean otras funciones como *predict()* o *reshape()* que permiten predecir la clasificación de los píxeles en cada uno de los centroides y adecuar las dimensiones de la imagen (*array*), respectivamente.

- *PIL: Python Imaging Library,* o también conocida como *PILLOW,* permite editar imágenes desde Python. Soporta gran variedad de formatos, como GIF, JPEG o PNG. Se ha utilizado para abrir imágenes de forma alternativa a las funciones de la librería *OpenCV.*

- *OpenCV*: Entre sus múltiples utilidades destaca la detección de rostros y objetos, sobre todo en ámbitos como la fotografía, el marketing o la seguridad. Se ha usado para la lectura de ciertas imágenes en las diferentes pruebas realizadas en el proyecto, pero sobre todo en la función de elaboración propia, *ProcesarImg()*, donde el algoritmo de *grabCut* [28] tiene gran importancia para detectar los píxeles del plano principal y diferenciarlos del fondo de la imagen y de aquellos píxeles pertenecientes a zonas que no interesan en el análisis (rostros o partes descubiertas del cuerpo humano).

- *Scikit-image* [29]: Es una colección de algoritmos para procesar imágenes que trabaja con *arrays* de numpy y algunas utilidades de matplotlib. Se han usado los siguientes módulos de esta librería:

    - Función *rgb2grayscale*: Computa la luminiscencia de una imagen RGB, de forma que elimina el canal de la dimensión.

    - Función *erosion*: Devuelve la erosión morfológica en escala de grises de una imagen. La erosión morfológica establece un píxel en *(i,j)* al mínimo sobre todos los píxeles en la vecindad centrada en *(i,j).* La erosión reduce las regiones brillantes y amplía las regiones oscuras.

    - Función *dilation*: Es la operación dual de la erosión. La dilatación morfológica establece el valor de un píxel al máximo sobre todos los valores de píxeles de un vecindario. Los valores donde la huella es 1 definen esta vecindad. La dilatación agranda las regiones brillantes y encoge las regiones oscuras.

    - Función *opening*: Permite eliminar pequeños puntos brillantes y conectar pequeñas zonas oscuras.

    - Función *label*: Mediante esta función se asignan etiquetas a los píxeles de las diferentes regiones de una imagen, lo que nos permitirá identificarlas y, por tanto, acceder a cada región. Dos píxeles están conectados cuando son vecinos y tienen el mismo valor. Ver figura 7.

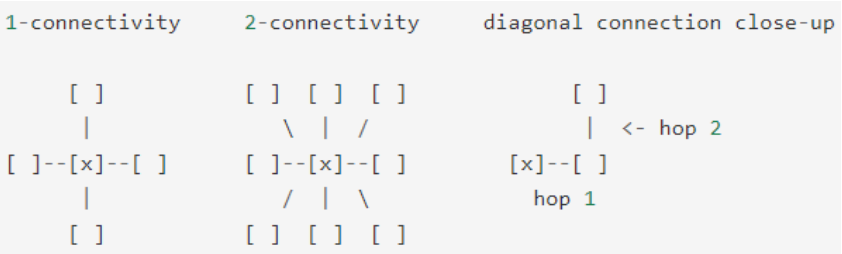


Figura 7. Conectividad de píxeles. Disponible en https://scikit-image.org/docs/dev/api/skimage.measure.html

- *Networkx*: Es una librería que permite la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas. En este proyecto se ha empleado para graficar visualmente una matriz de pesos en forma de grafo ponderado, aunque los resultados se han mejorado mediante una aplicación especializada en grafos.

La herramienta *Graph Online* [30] es una aplicación que permite crear grafos de forma muy sencilla con tan sólo introducir una matriz de pesos (entre otras opciones). El programa identifica que los valores de la matriz van a ser los pesos del grafo. Además, en nuestro caso queremos que el propio color (coordenada RGB) de cada vértice sea la etiqueta, por lo que podemos acceder a este color e introducir su respectiva coordenada.

3.3 Análisis de eficiencia algorítmica

Como se ha explicado anteriormente, el algoritmo *K-Means* converge a un mínimo local, y antes de que converja, se calcula la distancia y los centroides (es decir, todos los puntos son asignados a su centroide más cercano) mientras los bucles se ejecutan un número determinado de veces, llamémosle *i*. El valor entero y positivo *i* se corresponde con el número de iteraciones del algoritmo y depende de la inicialización establecida. Por tanto, el coste computacional del algoritmo *K-Means es* O(*n*k*i*), donde *n* es el número de datos del *dataset*, *k* es la cantidad de centroides seleccionados e *i* es el número de iteraciones, *k*<=*n*, *i*<=*n* [31].

Mediante la función *CrearMAdy()* implementada en Python, creamos la matriz de pesos de una imagen procesada previamente con el algoritmo *K-Means*. Dada la entrada de la posición *(i,j)* de la matriz original, se han buscado únicamente sus vecinos *(i,j+1),(i+1,j-1), (i+1,j)* e *(i+1,j+1)*. Esto ofrece una solución con menor coste computacional que si comparamos cada entrada con sus 8 vecinos conectados. El coste algorítmico de dicha función es de O(4*n*) donde *n* es el número de píxeles de la imagen recorrida (o elementos de la matriz).

Respecto al apartado que consiste en replicar el patrón de diseño extraído de una prenda en otra, el coste de calcular todas las posibles combinaciones de los colores es de O(k^r) donde *r* es la cantidad de regiones en las que se ha segmentado la nueva imagen y *k* es el número de colores de la paleta. Este coste exponencial hace que sea extremadamente costoso obtener todas las posibles combinaciones, especialmente en

el caso en que la cantidad de regiones segmentadas o el número de centroides sea elevado. Es por ello que en este TFG se va a calcular únicamente un pequeño subconjunto del total de posibles soluciones.

3.4 Análisis de marco legal y ético

Los datos analizados en este proyecto son los píxeles de las imágenes seleccionadas. Dichas imágenes son de libre disposición, y proceden de las webs de diferentes marcas de ropa conocidas. Su selección se basa en la "dificultad" de la propia imagen, la cual se explica en el apartado 4. Preparación y compresión de datos.

Respecto a la documentación teórica, la información de los estudios, investigaciones y webs consultadas simplemente se ha usado con fines orientativos, para saber cómo plantear una adecuada estructuración de la información que permita a los lectores comprender cada aspecto de este proyecto.

En cuanto al código programado en Python, la mayoría es de elaboración propia, a excepción de ciertas funciones de código abierto ofrecidas por las propias librerías de Python o por alguna web pública, que pertenecen al llamado *software* no privativo.

Tanto la documentación teórica como el código Python recogido de internet proceden de fuentes de uso abierto, no sujetas a propiedad intelectual ni a *copyright*.

3.5 Análisis de riesgos

En este proyecto no hay riesgos relacionados con el *software* desarrollado, pues es código de elaboración propia en su mayoría, y su uso no conlleva ningún tipo de riesgo para el TFG.

4. Preparación y compresión de datos

En este capítulo se va a hablar de los diferentes tipos de imágenes con prendas o accesorios que podemos encontrar por internet, cuáles son sus ventajas y problemas para el proyecto realizado, y en qué imágenes funciona mejor las solución propuesta.

4.1 Tipos de imágenes para llevar a cabo el proyecto

En primer lugar, es importante conocer los diferentes tipos de imágenes con los que nos podemos encontrar en internet y los problemas o dificultades que pueden aparecer a lo largo del proyecto en función de la selección que hagamos. A continuación se muestran los tipos de imágenes más comunes que podríamos utilizar en este proyecto:

- Imágenes de prendas con fondo simple y sin problemas de luminiscencia.
- Imágenes de prendas con fondo simple y con problemas causados por diferentes tonalidades de un mismo color debido a la luminiscencia.
- Imágenes de modelos con fondo simple.
- Imágenes de modelos con mucho ruido en segundo plano.

Como ya se ha comentado, el principal inconveniente que puede aparecer al seleccionar una imagen es la presencia de un segundo plano con mucho ruido, el cual dificulta la detección de la figura humana. En la figura 8 podemos ver varias imágenes de este tipo. No obstante, hemos visto que con diferentes técnicas y algoritmos de *machine learning*, como *background subtraction*, este preprocesamiento de la imagen inicial puede hacerse de forma relativamente sencilla, detectando los píxeles del fondo y estableciéndolos a un mismo color.

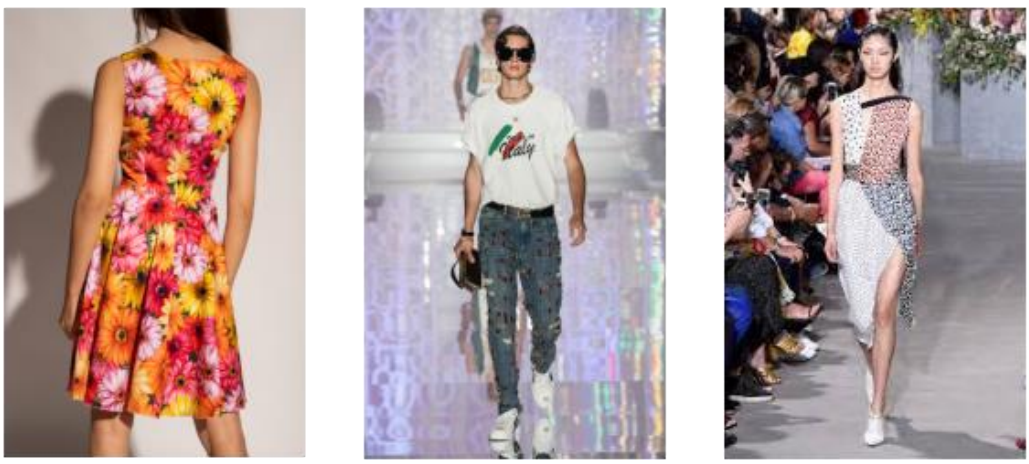


Figura 8. Imágenes con un/una modelo como figura de primer plano y con fondo ruidoso

Sin embargo, otro gran problema que puede aparecer y el cual no se ha resuelto en este proyecto, está relacionado con las diferentes tonalidades de un mismo color que pueden aparecer en una imagen debido a la luminiscencia (la imagen está capturada de forma que la mayoría de la luz se enfoca en una determinada parte). En la imagen central de la figura 9 podemos ver un bolso de Gucci, en cuyos laterales se aprecian diferentes

tonalidades de color marrón, debido a la luz, cuando realmente todo el borde está coloreado del mismo tono.

Una posible solución sería acceder al espacio de colores HSV de cada imagen, calcular el valor medio del brillo (componente *v*) y sustituirlo en dicha componente. Todo ello aparece explicado con más detalle y con imágenes en las conclusiones, en el apartado sobre los problemas surgidos a lo largo del proyecto.

Figura 9. Tipos de imágenes. En la izquierda, imagen con fondo simple y sin problema de luminiscencia. En el centro, imagen con fondo simple y con problema de luminiscencia. A la derecha, imagen con una modelo y fondo simple

4.2 Imágenes en las que mejor funciona la solución propuesta

Tal y como se ha comentado en el apartado anterior, es posible obtener buenos resultados al aplicar el algoritmo *K-Means,* excepto en las imágenes en las que hay mucha luminosidad localizada en alguna parte del objeto, siempre y cuando el algoritmo de sustracción del fondo funcione adecuadamente.
En los apartados de “Funciones aplicadas” y “Validación y despliegue” se mostrarán ejemplos del correcto funcionamiento de la solución propuesta sobre los tipos de imágenes mencionados.

5. Solución propuesta

En esta sección se va a mostrar la solución propuesta al problema planteado, de forma que en primer lugar hablaremos sobre cada una de las fases en las que se ha dividido el proyecto, y posteriormente, se explicarán las funciones implementadas en Python que permiten hacer cada paso. En la figura 10 se ilustra un pequeño organigrama que recoge un resumen del proceso a seguir. Como puede verse, se han propuesto siete fases, que van desde la selección de las imágenes sobre las que se quiere extraer y replicar el patrón de diseño, hasta la obtención de las soluciones.

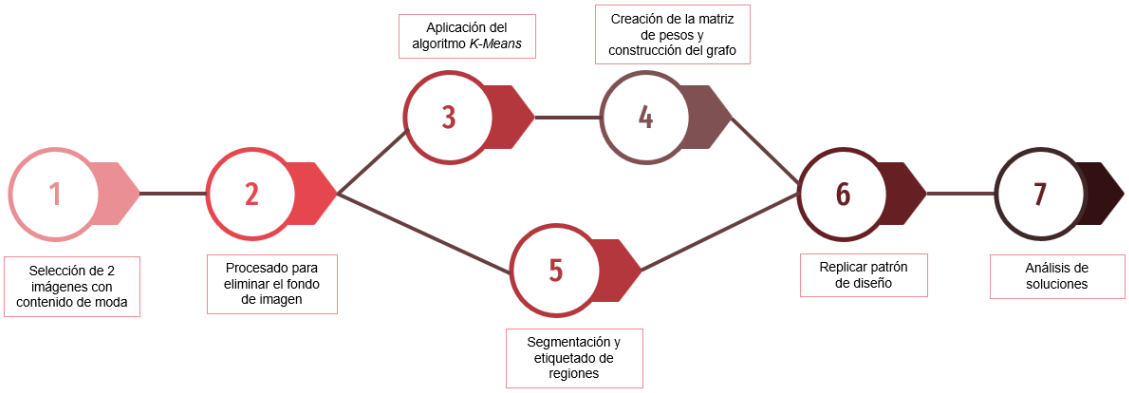


Figura 10. Organigrama de la solución propuesta. Fuente: elaboración propia

5.1 Fases del proyecto

Fase 1: Selección de imágenes con contenido de moda reciente

Para comenzar este proyecto, el primer paso es buscar y seleccionar una imagen que contenga una prenda o accesorio cuyos colores nos llamen la atención. Dicha imagen será el objeto de análisis de este estudio, el cual consiste en extraer el patrón de moda de la prenda elegida y replicarlo en otra prenda distinta.

Se pueden elegir diferentes tipos de imágenes, tal y como se ha visto en el apartado 4. En función de la dificultad de la imagen, habrá que aplicar técnicas de sustracción del fondo o de detección de figuras humanas para, posteriormente, poder acceder cómodamente a los píxeles que forman parte de la prenda o accesorio. Si la imagen concentra mucho brillo en una zona determinada, su preprocesamiento va a verse dificultado. Esto lo veremos en el apartado 8.2 que trata sobre los problemas surgidos a lo largo del proyecto.

Fase 2: Procesamiento de las imágenes

Esta fase consiste en procesar la imagen seleccionada, de forma que seamos capaces de identificar rápidamente qué píxeles pertenecen al segundo plano de la imagen (fondo y piel humana) y cuáles forman parte de la prenda. Para ello, y en función de la dificultad

de la imagen, se han implementado las siguientes funciones, explicadas detalladamente en los apartados 5.2.1 y 5.2.2**:**

- *AsignarColorFondo()* y *EliminarFondo()*
- *ProcesarImg()*

Fase 3: Aplicación del algoritmo *K-Means* para cuantificar el color de una imagen

Al cuantificar el color de una imagen, se reduce su número de colores y también el tamaño del fichero. El objetivo es detectar los *k* colores más representativos de una prenda, y con ellos, ser capaces de representar la imagen al completo. Esto se consigue aplicando el algoritmo *K-Means*, que es una técnica que selecciona *k* centroides basándose en la distancia entre los píxeles, y clasifica o predice el grupo al que pertenece cada píxel.

Una vez obtenidos los centroides de la imagen (recordamos que cada centroide es una tupla de 3 elementos, ya que es una coordenada RGB), los representamos mediante una paleta de colores para obtener una resumen visual de la aplicación del algoritmo *K-Means*. Dependiendo de si el fondo se ha establecido a blanco o a negro, las coordenadas de cada centroide se ven ligeramente modificadas a nivel numérico, pero a nivel visual y estético esta variación es insignificante.

Finalmente disponemos de *k clusters*, cada uno de ellos con su centroide. La reducción del número de colores se hace asignando, para cada *cluster*, el color de su centroide correspondiente. Es decir, estamos asignando el color de los diferentes píxeles que caen dentro de un *cluster* determinado al color del centroide. De esta manera se obtiene una imagen con solamente la cantidad de colores que se pretende. Todo ello se consigue mediante la función *FlattenColors()* explicada en el apartado 5.2.3.

Fase 4: Creación de la matriz de pesos

Mediante la fase anterior, una imagen con miles de colores diferentes se ha reducido a tan sólo *k* colores. Gracias a esta significativa reducción, ahora ya se puede recorrer la imagen píxel a píxel y construir un grafo con una matriz de pesos asociada $P = p_{ij}$ donde p_{ij} es la cantidad de píxeles del color *i* que hay alrededor del color *j*. Una vez tengamos dicha matriz creada, se deberá normalizar (dividir entre la suma de los valores de la matriz) para tener la información en términos de proporcionalidad.

Finalmente, se representará visualmente el grafo, con sus nodos (cada uno de los centroides del algoritmo *K-Means* o colores), aristas (indican que el color *i* está al lado del *j*) y pesos (ponderados con la cantidad de píxeles de color *i* que hay al lado de los píxeles de color *j*). En este proyecto, todos los grafos obtenidos son no dirigidos y ponderados. Además, los elementos de la diagonal de dicha matriz de costes van a ser distintos a 0, pues es muy probable que, dado un píxel, sus vecinos sean del mismo color. El objetivo de este apartado es obtener el patrón de colores o la forma en la que estos se distribuyen en la prenda, o en otras palabras, extraer la información que explica el por qué los colores de esa prenda están de moda y gustan a los consumidores.

A continuación se explica el proceso de obtención de la matriz de pesos:

En primer lugar, hemos de comprender el proceso llevado a cabo para aplicar el algoritmo *K-Means* con el objetivo de transformar la imagen original. Para ello, recordemos los términos de *cluster* y centroide introducidos en el punto 3.1.4. Por *clusters* entendemos los *k* bloques en los que se van a clasificar los datos (píxeles), de forma que los puntos que pertenecen al mismo bloque son muy similares (es decir, los colores son parecidos entre sí). Los centroides son los puntos representantes de cada *cluster*, de forma que todos los datos de un *cluster* determinado se van a clasificar (se les va a asignar) con el color de su respectivo centroide.

Seguidamente, nos fijamos en la variable *clusters*, la cual contiene una matriz que comparte las dimensiones de la imagen original y sus valores son índices desde 0 hasta la cantidad de colores de la paleta (o centroides) menos uno (*k-1*). El objetivo es recorrer cada entrada de la matriz *A* obtenida tras haber aplicado el algoritmo *K-Means*, y compararla con sus vecinos.

Primero creamos una matriz vacía y cuadrada a la que denominamos *M*, de dimensiones igual al número de colores de la paleta, la cual iremos rellenando y será la matriz de pesos. Como sabemos que la imagen va a tener un objeto de primer plano y que el resto de píxeles van a formar parte del fondo, podemos simplificar el proceso de creación de esta matriz de pesos *M* obviando la primera y última fila, así como la primera y última columna. Esto lo podemos hacer porque dichos píxeles pertenecen al fondo de la imagen, y su exclusión no va a afectar en nada al patrón de diseño, mientras que sí nos va a proporcionar una mejor eficiencia al llevar a cabo este proceso.

Para hacer este proceso más rápido y eficiente, observaremos exclusivamente los píxeles de las posiciones derecha, abajo izquierda, abajo y abajo derecha de cada píxel. Así, si al examinar una pareja de las comentadas nos encontramos con los colores *i* y *j* de la paleta, en el caso de ser *i* distinto de *j* sumaremos un 1 en cada una de las posiciones *(i,j)* y *(j,i)* de la actual matriz *M*, y en caso de ser *i=j* añadiremos un 1 en la posición diagonal *(i,i).* Llamaremos *M* a la nueva matriz y pasaremos a estudiar la siguiente pareja de píxeles. La matriz obtenida es simétrica. Finalmente, dividimos la matriz *M* entre la suma de todas sus valores para obtener así la matriz de pesos normalizada, de forma que este patrón de diseño será aplicable a otras prendas.

Ahora vamos a explicar todo lo anterior mediante un sencillo ejemplo. Disponemos de las matrices *A* y *M*. Sea la matriz *A* de dimensiones 3x3:

$$A = \begin{pmatrix} [253,253,253] & [253,253,253] & [253,253,253] \\ [253,253,253] & [230,84,49] & [61,37,71] \\ [253,253,253] & [61,37,71] & [230,84,49] \end{pmatrix}$$

Como podemos ver, *A* sólamente cuenta con 3 valores (colores) diferentes, lo cual se debe a que esta matriz *A* es una simulación de la imagen (en formato matricial) obtenida tras haber aplicado el algoritmo *K-Means* a la imagen original. Por otro lado, debe quedar claro que la matriz de pesos *M* es cuadrada y de dimensiones igual a la cantidad

de colores de la paleta, en este caso, de 3x3. Al crearla, se inicializan todos sus valores a 0, y se va construyendo conforme se van recorriendo los elementos de *A*.

Para comprender bien el proceso, se va a explicar con una pequeña matriz de pesos de prueba.

Ejemplo de proceso de obtención de la Matriz de pesos

Este es el procedimiento implementado en la función que calcula la matriz de pesos, llamada *CrearMAdy(),* de una imagen procesada previamente con el algoritmo *K-Means*. Consiste en observar los vecinos derecho, diagonal inferior izquierdo, inferior y diagonal inferior derecho de cada píxel que no pertenece a las posiciones de los laterales. De esta forma, se optimiza el coste computacional, ya que solamente accedemos a la mitad de vecinos de cada píxel. Tras el proceso, la matriz de pesos *M* obtenida puede verse en la tabla 1.

	[253, 253, 253]	[61, 37, 71]	[230, 84, 49]
[253, 253, 253]	0	0	1
[61, 37, 71]	0	0	2
[230, 84, 49]	1	2	1

Tabla 1. Matriz de pesos

Antes de comenzar a explicar el proceso de rellenar la matriz *M*, he de aclarar que en Python, las filas de una matriz de dimensión *n* x *n* se numeran desde el 0 hasta el *n*-1. Nos centramos en el elemento de la posición [1,1] de *A*, que es (230,84,49). Observamos que sus vecinos derecho e inferior son (61,37,71), por lo que sumamos +2 en las posiciones [1,2] y [2,1] de *M*. Como su vecino diagonal inferior derecho es (230,84,49), sumamos +1 en *M*[2,2], mientras que como el vecino diagonal inferior izquierdo es (253,253,253), sumamos +1 en las posiciones [2,0] y [0,2] de *M*.

Si la imagen *A* tuviese más píxeles con 8 vecinos-conectados, el proceso anterior se repetiría para cada uno de ellos. Pero como el resto de píxeles forman parte de los laterales, no los tenemos en cuenta para calcular la matriz de pesos. Para terminar, debemos normalizar *M*, por lo que habría que dividir dicha matriz entre la suma de todos sus elementos.

Fase 5: Segmentación y etiquetado de regiones

Comenzamos esta fase eligiendo una nueva imagen, diferente a la original (la prenda que va a ser objeto de esta parte del estudio puede ser parecida a la de la imagen original u otro tipo de prenda). A esta imagen se le aplica un procesado similar al de la imagen original, con el objetivo de eliminar el fondo.

La segmentación de imágenes consiste en dividir una imagen en partes o regiones, basándose en las características de los píxeles de la imagen. En esta sección, se va a explicar el proceso de detección, etiquetado y medición automática de objetos en

imágenes mediante componentes conectados, el cual se basa en agrupar píxeles en función de su conectividad. Se va a emplear las librerías *numpy, matplotlib* y *skimage* de Python.

Nuestro objetivo será extraer características cuantificables en cada una de las partes de la nueva imagen. Para ello, se transforma la imagen a blanco y negro, a fín de computar la luminiscencia. A continuación, hay que etiquetar las regiones de la imagen de entrada según la conectividad existente entre los píxeles, de tal forma que se obtenga una matriz etiquetada donde los píxeles de cada región tengan asignado el mismo valor entero. Si los píxeles vecinos comparten el mismo valor, se etiquetarán como pertenecientes a una misma región. También puede ser útil medir algunas de las propiedades de la imagen etiquetada, como el área, perímetro o longitudes de los ejes para tener información sobre las proporciones de las diferentes regiones.

Fase 6: Replicación del patrón de diseño

Esta fase consiste en asignar los colores de la paleta a la imagen segmentada. Una vez hemos segmentado en regiones la nueva imagen, y cada región ha sido etiquetada de forma que se distingue de cada una de las otras, el siguiente paso consiste en asignar a cada etiqueta (valores de 0 hasta *r-1*, donde *r* es la cantidad de regiones en las que el algoritmo de la fase 5 ha segmentado la imagen) un color de la paleta de colores que hemos extraído de la primera imagen.

Si llamamos *r* a la cantidad de regiones segmentadas, y *k* al número de colores de la paleta, es muy probable que *r* > *k*, por lo que a varias regiones diferentes se les asignará un mismo color. En total habrá k^r combinaciones posibles de asignar los colores de la paleta a la nueva imagen, o en otras palabras, diferentes formas de replicar el patrón de diseño de la prenda original.

Este proceso va a ser computacionalmente muy costoso debido a que hay que estudiar las posibles combinaciones de asignación de los colores de la paleta a las distintas zonas de la imagen segmentada. Por ejemplo, si la paleta consta de 5 colores y la nueva imagen se ha segmentado en 10 regiones diferentes, tendremos $5^{10} = 9.765.625$ formas diferentes de colorear la imagen segmentada. Por tanto, obtener todas las posibles combinaciones requerirá mucho tiempo de ejecución, por lo que debemos buscar una solución viable.

Lo que realmente se ha hecho ha sido generar un pequeño número de imágenes donde el patrón de diseño se replica aleatoriamente en la nueva imagen, junto a la forma más lógica de replicar dicho patrón observando la matriz de pesos original. De esta manera, esperamos conseguir una solución con el mínimo error posible respecto a la imagen de partida, y una serie de muestras aleatorias con diferentes errores. Sin embargo, esta solución conlleva un problema, y es que no estamos automatizando la réplica del patrón de diseño de una prenda en otra, si no que estamos haciéndolo de una forma manual donde se requiere la interacción humana para que el algoritmo seleccione la mejor combinación de colores.

Fase 7: Análisis de soluciones

Tras haber obtenido un subconjunto reducido de todas las posibles opciones de replicar el patrón de diseño, hemos de obtener la matriz de pesos asociada a cada solución para así, poder compararlas con la matriz análoga de la imagen original. Dicha comparación entre matrices se realizará mediante el cálculo de la distancia euclídea, y de esta manera obtendremos los errores de cada solución. La mejor solución será la que devuelva un menor error. Este análisis de soluciones es mucho más sencillo y viable que comparar todas las posibles combinaciones.

5.2 Funciones aplicadas

En el Anexo II se encuentra disponible el código Python de cada una de las funciones o procesos de los que se habla a continuación.

5.2.1 *AsignarColorFondo()* y *EliminarFondo()*

En el caso en el que se trata con imágenes donde sólo hay un objeto o una prenda y el fondo tiene poco ruido, el proceso a seguir consiste en asignar el mismo color a cada píxel que forma parte del fondo y asegurarnos que ese color no forma parte de la prenda. Cuando nos encontramos con una imagen que contiene un/una modelo, cuyo fondo carece de ruido, el proceso a seguir es igual que en el caso anterior. Para estos tipos de imágenes, el fondo se puede procesar con las siguientes dos funciones implementadas en Python: *AsignarColorFondo()* y *EliminarFondo().*

La primera recibe una imagen (en formato RGB) abierta con el módulo Image de la librería *PIL* y una tupla de 3 elementos que se corresponde con el color en coordenadas RGB que se va a asignar al fondo de la imagen. Mediante la función *floodfill* del paquete *ImageDraw* de la librería *PILLOW* se consigue hacer esta asignación del color deseado a los píxeles del segundo plano de la imagen.

Por otra parte, la función *EliminarFondo(),* a la que se le pasa como parámetro una imagen abierta con la librería *OpenCV*, es algo más compleja. Esta función permite no solo detectar el fondo, sino también zonas del cuerpo humano donde la piel es visible, y considerarlas como fondo para que no se tengan en cuenta como parte de la prenda. El fondo realmente no se elimina, sino que se convierte a color negro o blanco (dependiendo de cuál interese más según su presencia o no en la prenda a analizar). Esta función convierte la imagen RGB en escala de grises, aplica un umbral para detectar los contornos de la figura humana y superpone una máscara con determinados umbrales mediante las funciones *findContours* y *drawContours* [28] de la librería *OpenCV*. Dependiendo de los ajustes del mencionado umbral y los valores de la máscara, la precisión al distinguir lo que no forma parte de la figura y lo que sí lo hace, varía considerablemente.

En la figura 11 se pueden los resultados de procesar una misma imagen mediante las dos funciones anteriores. La función *AsignarColorFondo()* recibe como parámetros la

imagen original y el color negro en formato RGB, que se corresponde con las coordenadas (0,0,0), mientras que la función *EliminarFondo()* simplemente recibe la imagen original. Como se puede apreciar, EliminarFondo() procesa la imagen casi a la perfección debido a la precisión de las funciones utilizadas de la librería *OpenCV*.

Figura 11. Resultado de aplicar las funciones EliminarFondo() y AsignarColorFondo() a una imagen. Fuente: elaboración propia

5.2.2 *ProcesarImg()*

En el caso donde exista mucho ruido en el segundo plano, las funciones *AsignarColorFondo()* y *EliminarColor()* no hacen un correcto procesado del fondo. Para ello, se implementa la función *ProcesarImg(),* que permite procesar imágenes con *background* muy ruidoso (y en general se puede aplicar a cualquier imagen para establecer el segundo plano a color negro) mediante un algoritmo de detección de *background* y f*oreground*. Sólo es necesario ingresar una imagen y seleccionar algunos píxeles que pertenecen al fondo o al primer plano, y el algoritmo de *grabCut* [28] calculará la línea divisoria entre ambos planos en toda la imagen en función de esta marca local. Para ello, debemos indicar las dimensiones de la imagen, así como una aproximación de las posiciones *x* e *y* del objeto (prenda en este caso) que se quiere analizar. La figura 12 ilustra a la izquierda una imagen original, y a la derecha, el resultado obtenido tras procesar dicha imagen.

Figura 12. Resultado de aplicar la función ProcesarImg(). Fuente: elaboración propia

5.2.3 *FlattenColors()*

Básicamente, esta función permite aplicar el algoritmo *K-Means* a una imagen. Recibe como parámetros la imagen (cuyo único objeto es una prenda y tiene fondo uniforme) y el número de centroides (cantidad de colores que constituirán la paleta) que queremos obtener. Si la imagen original contiene a un/una modelo, dicha imagen debe ser procesada mediante alguna de las funciones de los apartados 5.2.1 o 5.2.2, con el objetivo de extraer exclusivamente la ropa que va a ser objeto del análisis, para lo que debemos eliminar todo lo demás. En el caso más sencillo, se le puede pasar la imagen de una sola prenda con fondo uniforme.

Comenzamos convirtiendo la imagen en una matriz *M*, de dimensiones *nxm*, en la que cada entrada es una lista de 4 elementos que se corresponden a los valores RGBA de cada uno de los píxeles, siendo *A* el alpha, es decir, la transparencia, pero como los píxeles siempre son opacos, este va a ser 255.

El siguiente paso consiste en entrenar el algoritmo *K-Means* con esta matriz de listas *M*. Partiendo de *p* centroides y utilizando la función *KMeans()* de la librería *Scikit-learn* de Python, clasificamos los píxeles en *p* grupos, asignándole a cada uno de ellos una etiqueta, que los relaciona con su centroide correspondiente. En este punto, consideramos dos variables diferentes, *clusters* y centroides. *Clusters* es una matriz del tamaño de la imagen con entradas del 0 al número de colores de la paleta menos uno, es decir, *p*-1. Esta matriz de índices es la que se utiliza para calcular la matriz de pesos. Por otro lado, centroides es un vector de longitud *p* cuyas entradas son listas con los colores en coordenadas RGB. Va a ser útil para saber qué índice o etiqueta corresponde a cada color, y así asignar correctamente los encabezados de la matriz de pesos.

Seguidamente se reestructura la matriz *M*, reduciendo la profundidad de sus entradas de 4 a 3. Para ello, se crea una matriz vacía de dimensiones *nxm*, y profundidad 3. Una vez se tienen los colores, se inicializa esta a ceros y se recorren las etiquetas, obteniendo los índices de los píxeles con esa etiqueta. Estas posiciones en esta matriz se convierten al valor de su centroide (su color), obteniéndose una imagen con sólo *p* colores.

Respecto a la paleta de colores, se crea una matriz la cual se divide uniformemente por el número de centroides, y cada "segmento" se rellena con el color (tuplas de 3 elementos, RGB) de cada centroide. El color del fondo se excluye, ya que no pertenece a la prenda en sí.

FlattenColors() devuelve la paleta que muestra visualmente los colores, un *array* Python con las coordenadas RGB de cada color o centroide, una comparativa entre la imagen original y la procesada con el algoritmo *K-Means*, así como los *clusters* y los centroides obtenidos durante el proceso.

5.2.4 CrearMAdy()

Esta función recibe como parámetro la matriz de índices llamada *clusters* que se genera en la función *FlattenColors*(). El objetivo es recorrer cada elemento de la matriz obtenida tras haber aplicado el algoritmo *K-Means*, y compararlo con sus 8 vecinos (ir analizando

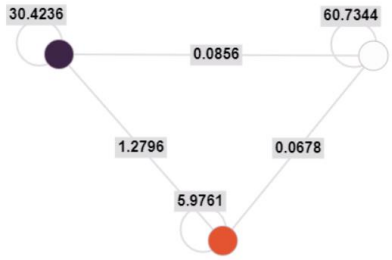
la matriz mediante ventanas 3x3 de píxeles). En primer lugar, creamos una matriz vacía cuadrada *M* de dimensiones igual al número de colores de la paleta, la cual iremos rellenando y será la matriz de pesos. Dicha matriz *M* se construirá tal y como se ha explicado en la fase 4 del apartado 5.1 Fases del proyecto.

5.2.5 DrawGraph()

Esta función recibe como parámetro la matriz de pesos creada anteriormente y transformada al formato de la librería *networkx*. Se le indica que los nodos son los índices de la matriz (colorea cada nodo utilizando las coordenadas hexadecimales de cada color, para ello se ha debido de transformar las coordenadas RGB a hexadecimal). Las entradas de la matriz son los pesos de cada una de las aristas o relaciones.

Sin embargo, el resultado obtenido mediante dicha función resulta difícil de entender y visualizar cuando la cantidad de nodos es elevada, debido al solapamiento de las aristas y sus respectivos pesos. Por motivos de estética visual, empleo una web llamada *Graph Online* [30], a la cual se le pasa una matriz de pesos y las coordenadas RGB del color de cada nodo, y permite visualizar claramente el grafo.

En la figura 13 se observa un grafo no dirigido ponderado creado mediante la aplicación *Graph Online*, junto a su matriz de pesos. El color de cada nodo representa un centroide de la paleta de colores, mientras que los valores de las aristas (pesos) representan la proporción de píxeles del color *i* que están al lado del color *j*, como se comentó en la fase 4.



	(61, 37, 71)	(253, 253, 253)	(229, 84, 49)
(61, 37, 71)	30.4236	0.0856	1.2796
(253, 253, 253)	0.0856	60.7344	0.0678
(229, 84, 49)	1.2796	0.0678	5.9761

Figura 13. Grafo junto a su matriz de pesos. Fuente: elaboración propia

5.2.6 Proceso de segmentación de imágenes y etiquetado de regiones

En esta parte, se va explicar el proceso de detección, etiquetado y medición automática de objetos en imágenes mediante componentes conectados [32]. Este se basa en agrupar píxeles en función de su conectividad. Se emplean ciertas funciones de la librería *skimage*, ya comentadas en el apartado *3.2 Software empleado.*

Se escoge una nueva imagen, se procesa mediante alguna de las funciones de los apartados 5.2.1 o 5.2.2 y, una vez tenemos únicamente el objeto, ya podemos segmentarlo en regiones. En primer lugar, se binariza la imagen mediante la función *rgb2gray().* A continuación, la imagen en blanco y negro se procesa con las funciones *erosion()* y *dilation()*, que segmentan automáticamente la imagen en regiones.

Finalmente, se asignan etiquetas a cada una de las regiones segmentadas según la conectividad existente entre los píxeles, lo cual se hace mediante la función *label().* Siempre que los píxeles vecinos compartan el mismo valor, se etiquetarán como una sola región. Esta función devuelve una matriz etiquetada donde a todas las regiones conectadas se les asigna el mismo valor entero.

En caso en el que nos interese recuperar información sobre cada región, podemos emplear la función *regionprops()*, que recibe como parámetro la matriz etiquetada, y nos permite medir algunas de las propiedades de la imagen etiquetada, como el área, perímetro o longitudes de los ejes, entre otras.

5.2.7 Replicar el patrón obtenido en una imagen ya segmentada

En esta fase no se emplea ninguna función propia ni de ningún paquete de Python, ya que simplemente se realiza una asignación aleatoria de los colores de la paleta a las *r* regiones en las que se ha segmentado la imagen escogida. Este proceso de aleatoriedad se hace automáticamente tantas veces como soluciones queremos obtener.
Por otra parte, considerando los pesos del grafo del patrón de diseño de la imagen original, realizamos manualmente una asignación de los colores a las diferentes regiones, intentando obtener una posible mejor solución.

5.2.8 Cálculo de la distancia euclídea entre dos matrices

Una vez tenemos la imagen original y la nueva imagen coloreada para calcular el error existente entre sus matrices de pesos, aplicamos las funciones *FlattenColors()* y *CrearMAdy()* a cada una de las nuevas imágenes.

Sin embargo, cuando se aplica internamente el algoritmo *K-Means* en la función *FlattenColors(),* la asignación de los centroides no se realiza siempre de la misma forma, lo cual da lugar a que, cada vez que a dicha función se le pasa una de las nuevas imágenes obtenidas, el orden de las filas y columnas varía. Todo ello es un problema a la hora de calcular la distancia euclídea entre matrices, pues si los colores no siguen exactamente un mismo orden, la distancia se calculará erróneamente.
Para solucionar este contratiempo, se ha implementado la función *OrdenarDataframe()*, la cual recibe la matriz de pesos de una imagen en formato *dataframe,* y se ordena de menor a mayor según la coordenada RGB de cada color. Si la componente *red* coincide para dos colores, se compara la componente *green*, y sino, la *blue.*

Una vez están ordenadas todas las matrices, la distancia euclídea entre la matriz original y cualquiera de las nuevas matrices se obtiene mediante la función *linalg.norm()* de la librería *Numpy*, a la cual se le pasa la diferencia entre la matriz original y la de la nueva imagen. Otra forma de calcular la distancia euclídea es aplicar directamente su fórmula de cálculo.

6. Conocimiento extraído y evaluación de modelos

Como los datos empleados en este TFG son imágenes descargadas directamente desde internet, el conocimiento extraído en este proyecto simplemente está relacionado con la información que se puede obtener a través del patrón de diseño de una imagen. En cuanto a la evaluación del modelo realizado, se va a realizar un análisis de las fortalezas y debilidades que permita identificar los puntos a favor y en contra del modelo implementado.

Un punto fuerte de este modelo es que la representación del diseño en forma de grafo permite comparar diseños entre sí de forma más sencilla y visual. Además, la información contenida en el grafo permite saber cuál es la replicación del patrón de diseño más precisa en otra prenda distinta. El algoritmo *K-Means,* en el cual se basa el modelo, permite la interacción humana debido a que es el usuario el que selecciona el número de colores (centroides) que se desea que tenga la imagen procesada y la paleta de colores. Por otro lado, dicha interacción humana tiene la desventaja de que existe la posibilidad en la que el usuario no tenga claro qué valor de *k* escoger o bien, que sea necesario elegir un valor muy alto de *k* para que la imagen procesada tenga sentido. Otra debilidad de este modelo es la dificultad de "colorear" un diseño con una paleta debido a la elevada cantidad de posibles combinaciones existentes.

7. Validación y despliegue

En este apartado vamos a realizar una detallada demostración del proceso de extracción del patrón de diseño de una prenda y su replicación en otra diferente. Para ello, comenzamos seleccionando dos imágenes diferentes en las que, a ser posible, no exista el problema de luminiscencia del que hemos hablado anteriormente (Ver figura 14). De la chaqueta de colores de Gucci de la izquierda se va a extraer el patrón de diseño, el cual se va a replicar en la camiseta de Run de la derecha.

Figura 14. Imágenes sobre las que se va a extraer y replicar el patrón de diseño. Fuente: elaboración propia

Para comenzar con el proceso, vamos a analizar en primer lugar la imagen de la chaqueta de Gucci. Nos encontramos ante una imagen sencilla, donde el fondo es uniforme y todos los píxeles del objeto del primer plano parecen recibir una equitativa cantidad de luz. Por tanto, no es necesario aplicar ningún proceso de sustracción de fondo a la imagen, y podemos explicar directamente la extracción de la paleta de colores.

Para obtener los colores más representativos de la chaqueta de Gucci, vamos a aplicar el algoritmo *K-Means* a la imagen original. Hemos de decidir el número de colores o centroides, *k*, que queremos que tenga la nueva imagen transformada, y en consecuencia, la paleta de colores. Observando la chaqueta, podemos llegar a distinguir claramente 7 colores diferentes: amarillo, dos tonos de naranja, dos tonos de azul, verde y el gris del fondo. Por lo tanto, a la función *FlattenColors()* le pasaremos como parámetros la imagen del vestido y *k*=7. En las figuras 15 y 16 vemos el resultado de aplicar el algoritmo *K-Means* a la imagen original con 7 centroides, y también cómo sería el resultado si *k* fuese 4 o 5.

Figura 15. Resultado de aplicar el algoritmo K-Means con 7, 4 y 5 centroides. Fuente: elaboración propia

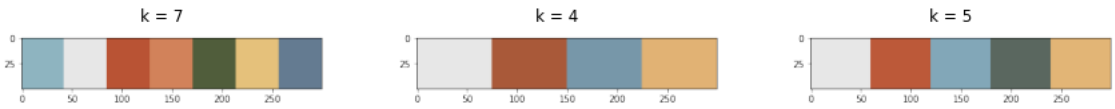


Figura 16. Paletas de colores obtenidas al aplicar 7, 4 y 5 centroides. Fuente: elaboración propia

Al seleccionar k=7, los centroides (colores en formato RGB) son los siguientes: ([142, 180, 192], [231, 231, 231], [185, 83, 52], [210, 130, 90], [80, 93, 59], [229, 193, 123], [100, 123, 145]]). Sin embargo, el color gris del fondo cuyas coordenadas RGB son [231,231,231] va a ser eliminado de la paleta de colores para que no interfiera en la obtención del patrón de diseño de la chaqueta, puesto que no pertence a la prenda.

Como salida (ver figura 17), dicha función devuelve la imagen original, la nueva imagen con únicamente 7 colores, la paleta de colores, los centroides (matriz con los 6 colores de la chaqueta en coordenadas RGB) y los *clusters* (matriz de iguales dimensiones que la imagen original donde cada píxel lo clasifica en uno de los *k clusters*, en este caso, de 0 a *k*-1, es decir, de 0 a 6). En la paleta no se refleja el color del fondo.

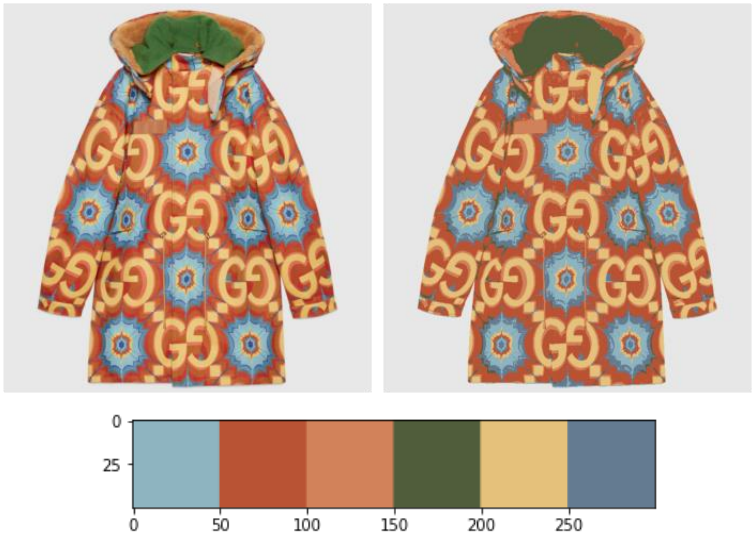


Figura 17. Salida de la función FlattenColors(). A la izquierda tenemos la imagen original, con 181551 colores diferentes. A la derecha, la imagen procesada con el algoritmo K-Means con únicamente 7 centroides. Abajo, la representación de los centroides en la paleta de colores. Fuente: elaboración propia

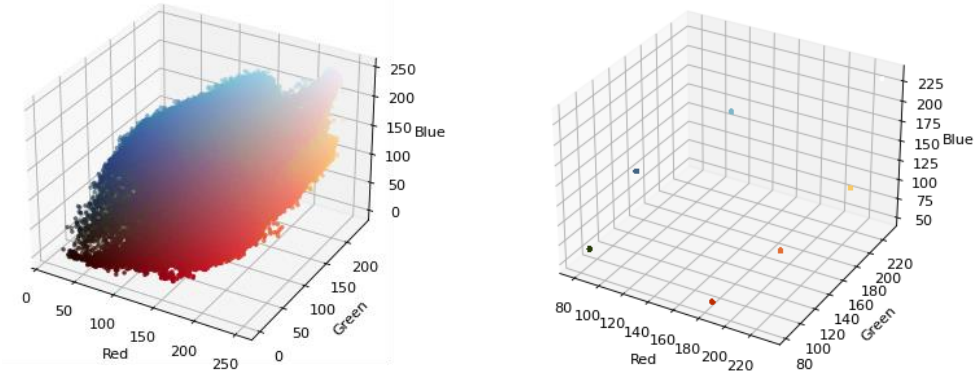


Figura 18. Figura 18. Representación en coordenadas RGB de la imagen del vestido de colores (izquierda) y representación de sus 7 centroides (derecha). Fuente: elaboración propia

El siguiente paso consiste en transformar la nueva imagen procesada con el algoritmo *K-Means* en una matriz donde se refleje la proximidad de los colores entre sí. Para ello, se van a recorrer todas las entradas de la imagen procesada (la hemos representado como una matriz) y se van a analizar los vecinos de cada una de ellas.

Este proceso se hace con la función *CrearMAdy(),* la cual recibe una imagen procesada con el algoritmo *K-Means* con únicamente *k* colores o centroides, y devuelve un *array*, el cual se transforma a un objeto *dataframe*. Este *array* es una matriz de dimensiones *(k-1)*(k-1)* (6*6 en nuestro caso) que contiene el valor normalizado de la cantidad de píxeles del color *i* que se encuentran al lado de los píxeles del color *j*, tal y como se ilustra en la tabla 2. La suma de todos sus valores da el 100%, y apreciamos que los elementos de la diagonal principal contienen valores más elevados, lo cual se debe a que la probabilidad de que dos píxeles vecinos sean del mismo color es mayor que si son de diferentes colores.

	(142, 180, 192)	(185, 83, 52)	(210, 130, 90)	(80, 93, 59)	(229, 193, 123)	(100, 123, 145)
(142, 180, 192)	9.9609	0.0221	0.0744	0.0083	0.2790	1.3147
(185, 83, 52)	0.0221	29.9061	2.1546	0.5462	0.1301	0.6875
(210, 130, 90)	0.0744	2.1546	9.6093	0.0585	1.5979	0.2553
(80, 93, 59)	0.0083	0.5462	0.0585	6.5735	0.0053	0.4255
(229, 193, 123)	0.2790	0.1301	1.5979	0.0053	19.2668	0.0401
(100, 123, 145)	1.3147	0.6875	0.2553	0.4255	0.0401	9.4844

Tabla 2. Matriz de pesos en formato dataframe de Python. Fuente: elaboración propia

Para evitar problemas a la hora de calcular la distancia euclídea en los próximos pasos, aplicamos la función *OrdenarDataframe()*, la cual ordena el *dataframe* de menor a mayor según los colores (se guía por la primera coordenada RGB, y en caso de empate, mira la segunda, o la tercera). Véase el resultado en la tabla 3.

	(80, 93, 59)	(100, 123, 145)	(142, 180, 192)	(185, 83, 52)	(210, 130, 90)	(229, 193, 123)
(80, 93, 59)	6.5735	0.4255	0.0083	0.5462	0.0585	0.0053
(100, 123, 145)	0.4255	9.4844	1.3147	0.6875	0.2553	0.0401
(142, 180, 192)	0.0083	1.3147	9.9609	0.0221	0.0744	0.2790
(185, 83, 52)	0.5462	0.6875	0.0221	29.9061	2.1546	0.1301
(210, 130, 90)	0.0585	0.2553	0.0744	2.1546	9.6093	1.5979
(229, 193, 123)	0.0053	0.0401	0.2790	0.1301	1.5979	19.2668

Tabla 3. Matriz de pesos ordenada por colores. Fuente: elaboración propia

Para comprender dicha matriz de pesos, construimos un grafo, dibujado en la figura 19, donde cada nodo está coloreado con uno de los colores de la matriz, y el peso de la arista *(i,j)* es el valor de la entrada *(i,j)* de la matriz. De esta forma se representa toda la información del patrón de diseño en términos gráficos y visuales.

En la figura 19 vemos que la probabilidad de que un píxel naranja oscuro esté al lado de otro píxel de este mismo color es de un 29.9%, mientras que la probabilidad de que dos píxeles amarillos sean vecinos es del 19.26%. Estos elevados valores nos indican

que los colores naranja fuerte y amarillo son los que más presencia tienen en la chaqueta, y tienen un peso mayor. También podemos decir que el color con menos píxeles vecinos entre sí es el verde, lo cual tiene sentido ya que este color solo tiene presencia en la zona interna de la capucha de la chaqueta, que ocupa una pequeña región de la imagen. En cuanto a las relaciones entre diferentes colores, apreciamos que los dos tonos de azul están muy próximos entre sí, así como los dos tonos de naranja y el amarillo con el naranja claro.

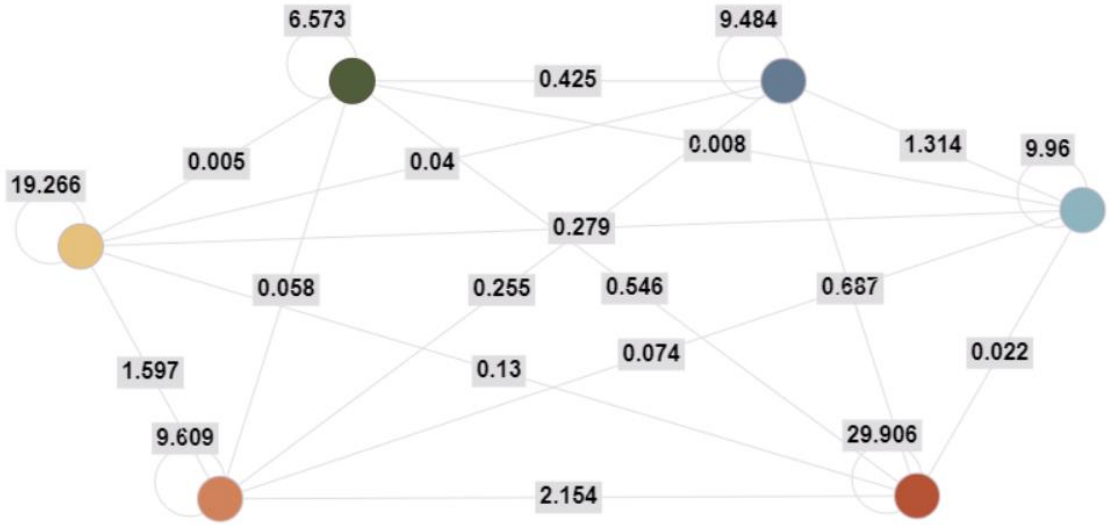


Figura 19. Grafo que representa la matriz de pesos. Fuente: elaboración propia

Una vez hemos extraído el patrón de diseño de la imagen de la chaqueta de Gucci, podemos olvidarnos de dicha imagen y centrarnos en la camiseta de Run (imagen derecha de la figura 14), en la cual vamos a replicar dicho patrón de diseño.

Comenzamos con un breve análisis de la imagen, donde a primera vista observamos que el fondo contiene diferentes tonos de color, y aparecen partes del cuerpo de la persona que viste la camiseta. Por lo tanto, hemos de procesar dicha imagen de forma que los píxeles del segundo plano y los pertenecientes a los rasgos físicos de la persona se diferencien de los píxeles de la camiseta, que es el objeto central de la imagen y el cual nos interesa.

Para ello, aplicamos la función *ProcesarImg(),* a la que le pasamos como parámetro la imagen que queremos procesar, indicando las dimensiones de la imagen, así como los valores *x* e *y* a partir de los cuales, los píxeles dejan de pertenecer al *background* y forman parte del objeto central. Como resultado obtenemos la imagen b) de la figura 20 de abajo, donde podemos observar que todos los píxeles no pertenecientes al primer plano se han coloreado de negro ((0,0,0) en coordenadas RGB).

Una vez tenemos la imagen con únicamente los píxeles de la prenda, hemos de segmentar la camiseta en regiones. En este caso, esto consiste en etiquetar numéricamente cada una de las regiones que se pueden observar en la imagen b) de la figura 20. A simple vista podemos decir que cada uno de los triángulos de colores va a ser una región diferente, así como cada una de las letras del centro de la camiseta.

Figura 20. Procesamiento de la imagen. En la figura a) se observa la imagen original, en la b) se ha aplicado la técnica de Background Subtraction. La imagen c) contiene la camiseta únicamente en blanco y negro, mientras que la imagen d) muestra la imagen ya segmentada en regiones y etiquetada. Fuente: elaboración propia

Tras aplicar los algoritmos de segmentación y etiquetado de regiones de la librería *scikit learn* de Python (estos son *rgb2gray()*, *dilation()*, *erosion()* y *label()*), obtenemos la imagen transformada a escala de grises y la imagen segmentada (Ver imágenes c) y d) de la figura 19, respectivamente). Dichas funciones establecen una segmentación en 44 regiones, donde son etiquetadas automáticamente desde la región 0 hasta la 43. Destacan las regiones con etiquetas 0 y 1, que se corresponden con los píxeles del color blanco de la camiseta y con los del fondo, respectivamente. También hay que decir que los algoritmos de segmentación tienen cierto margen de error al predecir los píxeles que van a pertenecer a una nueva región, lo cual justifica que, de las 44 regiones, unas 15 aproximadamente ocupan un área muy pequeña, y se les podría asignar automáticamente la etiqueta 0. Esto lo sabemos porque es posible obtener un breve análisis de las propiedades de las regiones, tal y como puede apreciarse en la tabla 4. Se observa que las regiones 5, 7, 8 y 9 ocupan un área insignificante respecto al resto de regiones, por lo que dichas zonas podrían haberse asignado a la etiqueta 0.

	centroid-0	centroid-1	orientation	area	convex_area	bbox_area	extent	solidity	eccentricity
0	348.644801	284.031085	-1.538548	188548	389244	389244	0.484395	0.484395	0.235801
1	80.838968	241.228648	0.806900	1124	1175	2205	0.509751	0.956596	0.648700
2	118.513749	195.781355	0.165163	1491	1572	2915	0.511492	0.948473	0.473150
3	170.762619	154.417462	0.341029	2199	2375	4293	0.512229	0.925895	0.833340
4	148.230352	274.731707	-1.062412	369	396	728	0.506868	0.931818	0.600998
5	146.578947	509.526316	-0.543105	19	23	48	0.395833	0.826087	0.954794
6	180.462687	516.303483	-0.676792	201	346	2091	0.096126	0.580925	0.995820
7	161.555556	502.611111	-0.528395	18	22	48	0.375000	0.818182	0.960965
8	160.000000	497.000000	0.785398	5	5	9	0.555556	1.000000	0.000000
9	164.000000	494.500000	-0.463648	10	12	20	0.500000	0.833333	0.870388

Tabla 4. Información geométrica de las primeras diez regiones segmentadas. Fuente: elaboración propia

El siguiente paso consiste en asignar los colores de la paleta (ver imagen inferior de la figura 16) a las diferentes regiones en las que se ha segmentado la camiseta. Si calculamos todas las posibles combinaciones, estaríamos ante k^r diferentes formas de colorear la camiseta, donde *r* y *k* son la cantidad de regiones en las que se ha segmentado la imagen y el número de colores de la paleta, respectivamente. Es decir, estaríamos ante $6^{44} = 1{,}732 * 10^{34}$ combinaciones. Obtener esta elevada cantidad de

imágenes es imposible en términos de coste computacional. Por lo tanto, la solución que se va a ofrecer se basa en conseguir una serie de muestras aleatorias de posibles combinaciones, así como la mejor y peor solución tras observar el patrón de diseño de la figura 19.

En el caso de la búsqueda de la mejor solución, a la etiqueta 1 le asignamos el color gris, pues sabemos que es el color del fondo de la imagen. A la etiqueta 0 le asignamos el color naranja oscuro, ya que anteriormente hemos visto que era el color de la paleta con más posibilidades de que, dados dos píxeles vecinos entre sí, ambos sean de dicho color. Por tanto, parece ser el color más adecuado para la camiseta en sí.

En este punto nos quedan 5 colores por asignar (naranja claro, amarillo, dos tonos de azul y verde) y 42 regiones por colorear (de la etiqueta 2 a la 43). Para hacer una asignación lógica entre los colores y las regiones restantes, el proceso que se ha seguido consiste en particionar la lista de las 42 regiones restantes en 5 secciones, donde las etiquetas aparecen ordenadas y a cada color se le asignan más o menos etiquetas en función del patrón de diseño. Por ejemplo, el color amarillo, que es el siguiente con más probabilidad de tener 2 píxeles vecinos de ese mismo color, ha sido asignado a las regiones 2, 3, 4, 5, 6 y 7, que son regiones de la camiseta con un área considerable (correspondientes a las letras de la palabra RUN y al dibujo. Y así con los demás colores y regiones. El resultado se puede observar en la figura 21.

Figura 21. Aplicación del patrón de diseño a una imagen segmentada en regiones para obtener la mejor solución. Fuente: elaboración propia

Respecto a las soluciones generadas aleatoriamente, se han generado seis combinaciones entre las miles de millones de posibilidades. El proceso de obtención de dichas imágenes es muy sencillo. Consiste en asignar el color gris a la región con etiqueta 1 (el fondo), asignar aleatoriamente uno de los seis colores restantes a la región etiquetada como 0 (el cuerpo de la camiseta), y el resto de colores son asignados de forma equitativa y aleatoria entre las regiones sobrantes. En la figura 22 podemos apreciar seis posibles soluciones aleatorias.

Figura 22. Posibles soluciones. Representación de la replicación del patrón de diseño en la camiseta de RUN, mostrando una posible mejor solución, una posible peor solución y un subconjunto de seis soluciones aleatorias. Fuente: elaboración propia

Para finalizar este análisis, vamos a obtener, para cada una de las soluciones anteriores, una matriz de pesos análoga a la matriz de pesos de la imagen original (ver Tabla 3). Esto se debe a que queremos comparar la matriz original con cada una de las nuevas matrices de las imágenes de muestra en las que se ha replicado el patrón de diseño. Mediante esta comparación se obtiene un error, calculado a partir de la distancia euclídea entre dos matrices. He de comentar en este punto que, para obtener un correcto cálculo del error, las filas y columnas de las matrices que comparamos deben estar ordenadas de la misma forma.

La mejor solución posible, es decir, la mejor aplicación del patrón de diseño del vestido de Gucci a la camiseta de RUN, será aquella que tenga el menor error. En la tabla 5 podemos observar la salida del programa Python que calcula la distancia euclídea entre la matriz original y cada una de las matrices creadas:

```
Mejor solución: 59.048
Aleatorio 1: 59.743
Aleatorio 2: 82.741
Aleatorio 3: 82.175
Aleatorio 4: 83.687
Aleatorio 5: 85.984
Aleatorio 6: 71.758
Peor solución: 86.309
```

Tabla 5. Errores de cada solución. Fuente: elaboración propia

Analizando los errores, confirmamos que la mejor solución planteada es, en efecto, la que obtiene un menor error. Apreciamos que la imagen Aleatorio 1 es muy similar a la mejor solución, lo cual se debe a que ambas comparten el color naranja oscuro como color central de la camiseta, y dicho color es el que tiene una mayor presencia en la imagen original de la chaqueta de Gucci. La imagen Aleatorio 6 de la figura 22 tiene un error intermedio entre la mejor solución y el resto de soluciones aleatorias (sin incluir Aleatorio 1), a causa de que su color central es el amarillo, y como hemos visto en el

grafo de arriba (figura 19), dicho color es el que tiene mayor importancia en el patrón de diseño, por detrás del naranja oscuro. Las demás soluciones aleatorias comparten un error similar.

En cuanto a una posible peor solución, hemos asignado el color verde (el que aparece en menor cantidad en la imagen de la chaqueta) a la región central de la imagen, de tal forma que ahora tiene un peso muy importante. Lo que se ha hecho ha sido asignar una mayor importancia a los colores con menor presencia en la imagen de la chaqueta, y viceversa, es decir, se ha tratado de replicar la peor combinación de los colores, respecto a la imagen original. Para crear esta solución, se ha seguido el proceso inverso utilizado para obtener la mejor solución. Como resultado, se obtiene una distancia euclídea de 86.3 unidades.
Vemos que la imagen Aleatorio 5 tiene un error similar al de la peor solución, lo cual se debe a que su color central es también el verde.

8. Conclusiones

En este apartado se van a reflejar las conclusiones tras haber terminado el proyecto, su relación con los aspectos estudios durante el grado, así como los principales problemas surgidos durante su realización y cómo se han solucionado.

8.1 Conclusiones del trabajo realizado

El modelo desarrollado ofrece la posibilidad de extraer el patrón de diseño de una prenda cualquiera, y replicarlo en otra prenda o accesorio deseado. Todo ello mediante algoritmos de *machine learning*, conceptos de teoría de grafos y técnicas de segmentación de imágenes.

Una vez se le hayan aplicado futuras mejoras y dicho modelo se integre en una aplicación web, su uso puede ir dirigido a las grandes marcas de moda con el propósito de lanzar a la venta sólo aquellas prendas o accesorios con combinaciones de colores que gusten entre los clientes. De esta forma, la industria textil que hay por detrás de las grandes multinacionales de ropa podría optimizar la cantidad de prendas a fabricar y dedicarse en mayor proporción a utilizar los patrones de color que más tendencia causan. Todo ello permitiría que queden menos unidades sin vender en los grandes almacenes, las cuales suelen ser desperdiciadas, y afectaría positivamente al medio ambiente, pues se reducirían los procesos de fabricación de ropa.

Personalmente, este proyecto me ha servido para consolidar conocimientos previos adquiridos durante el grado, especialmente las técnicas, algoritmos y conceptos estudiados en las asignaturas relacionadas con el análisis de datos y con la teoría de grafos. También he afianzado mis habilidades al programar en Python, así como al emplear técnicas de gestión y planificación de proyectos, estructurar y redactar memorias, y en general, aplicar los conocimientos adquiridos para resolver problemas reales.
Por otro lado, para completar este TFG también he sido autodidacta en ciertas partes, sobre todo en los apartados de procesado de imágenes y de aplicación de técnicas de segmentación, ya que durante el grado no he tenido la ocasión de estudiar sobre ello. Me ha impresionado la inmensa cantidad de aplicaciones que tiene la ciencia de datos para el análisis de imágenes, ya que he visto cierta cantidad de proyectos similares a este TFG, y sus aplicaciones a la vida real son realmente interesantes y útiles.

Respecto a las competencias transversales, en este TFG se han puesto en práctica: el análisis y resolución de problemas, ya que desde un primer momento se ha desmenuzado el proyecto, y se ha ido analizando parte a parte; la innovación, creatividad y emprendimiento, y el conocimiento de problemas contemporáneos, debido a que este proyecto está basado en un problema de la actualidad, y para ofrecer una solución eficiente hay que ser capaz de de aplicar las técnicas de ciencia de datos a este ámbito; responsabilidad ética, medioambiental y profesional, ya que el problema que acarrea la industria textil es algo a lo que hay que darle una rápida y efectiva solución; aprendizaje permanente, puesto que las técnicas de ciencia de datos mejoran día a día, y siempre se aprenden cosas nuevas, tal y como me ha ocurrido en este

proyecto; comunicación efectiva, debido a que todo proyecto debe ser transmitido correctamente, tanto de forma oral como escrita, para que los destinatarios comprendan su utilidad; y planificación y gestión del tiempo, ya que todo proyecto debe tener una clara organización que permita adaptarse a las necesidades y los plazos establecidos.

8.2 Problemas que han surgido a lo largo del proyecto

1) Dificultad al tratar el grado de luminiscencia en las imágenes:

Supongamos que disponemos de una imagen con un objeto el cual, por el motivo que sea, tiene reflejos de luz que provocan la aparición de diferentes tonalidades de un mismo color (véanse las figuras 23 y 24 del bolso y la visera, respectivamente). Al asignar el número de centroides (*k*), el algoritmo *K-Means* desconoce los colores básicos que se quieren escoger.

Para seleccionar dichos colores, que van a ser los centroides, el algoritmo se basa en la cantidad de píxeles de un determinado color, y asignará un centroide a los colores que se encuentran en mayor proporcionalidad. Es por ello que, a simple vista, en el bolso (figura 22) distinguimos un marrón oscuro y otro más clarito. Sin embargo, el algoritmo establece 3 tonos diferentes de marrón, ya que clasifica los píxeles de la zona marrón oscura que reciben más luz como un tercer tono de marrón, diferente a los dos que se pueden ver a priori.

Como posible solución a este problema, se podría transformar la imágen de RGB a HSV. Una vez hecho esto, a la función *FlattenColors()* habría que pasarle dicha imagen en HSV y hacer un *reshape* con tan sólo los valores del matiz y la saturación (coordenadas *h* y *s*). A continuación, se debería calcular el valor medio del brillo (*v*) de la imagen, y asignar dicho valor a la coordenada *v*. De esta forma, todos los píxeles recibirán una similar cantidad de luz y así, el algoritmo no confundirá colores que ciertamente son iguales.

En resumen, lo anterior es un problema ya que una asignación no lógica de los centroides causará la incorrecta aparición de determinados colores en la paleta, por lo que la representación del patrón de colores de esa prenda difícilmente será el adecuado. Este problema no se ha solucionado porque realmente, las prendas no tienen estos problemas de luminiscencia cuando se fabrican en la industria textil, sino que este problema aparece al fotografiarlas. Sin embargo, la incorporación de dicho tipo de imágenes queda pendiente para próximas mejoras de este proyecto.

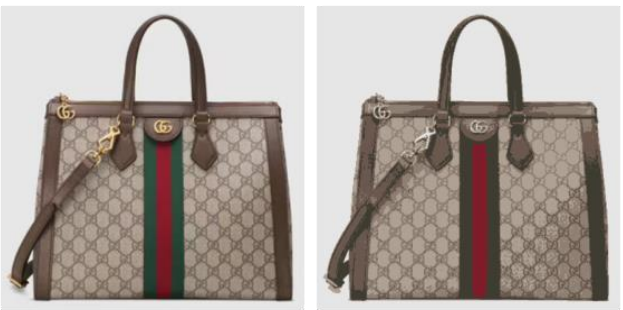


Figura 23. Figura original, y figura procesada con K-Means y 6 centroides. Fuente: elaboración propia

Figura 24. Figura original, y figura procesada con K-Means y 5 centroides. Se ve que la asignación de los colores a los píxeles no es adecuada. Con k=4 (el fondo, color paja de la visera, colores blanco y negro de la parte de atrás) el algoritmo no es preciso. Fuente: elaboración propia

2) Eliminación del *background* en imágenes con modelos y fondo muy ruidoso

Las funciones *CambiarColorFondo()* y *EliminarFondo()* son relativamente precisas en los casos en los que los modelos tienen un *background* uniforme. Sin embargo, la presencia de varios colores en el fondo hace que ambos algoritmos no sean capaces de detectarlo bien y de establecer todos sus píxeles a una misma coordenada RGB.

Como solución, se ha empleado la función *grabCut* [28] de la librería Opencv de Python, que permite hacer una segmentación óptima de la imagen, pasándole las dimensiones de la imagen y los valores *x* e *y* donde se localiza la figura. Todo ello aparece incorporado en la función *ProcesarImg()*.

3) Problema de aplicación biyectiva

Para asignar un valor único a cada tipo de píxel (es decir, trasformar cada coordenada RGB de 3 elementos a un solo número), comencé haciendo la media de cada coordenada RGB, y en función del valor, asignar 0 al píxel con menor media, 1 al siguiente, y así hasta la media mayor. Sin embargo, podría darse el caso de que dos coordenadas RGB diferentes tuvieran la misma media, por lo que a ambos colores se les asignaría el mismo valor. Es decir, la función definida no sería biyectiva, lo que supone un problema: por un lado, por la pérdida de colores que esto conlleva, y por otro, no poder obtener de forma unívoca las coordenadas RGB de los valores asignados a los píxeles.

Este problema se ha solucionado de una forma muy simple. La función en la que se implementa el algoritmo *K-Means*, *FlattenColors()*, etiqueta cada centroide con un valor distinto. Y ciertamente, estas etiquetas son los valores únicos que hay que asignar a cada tipo de píxel.

4) Inconvenientes del algoritmo *K-Means*

Los métodos que utilizan técnicas de *clustering*, como el algoritmo *K-Means*, proporcionan buenos resultados debido a que se adaptan a cada imagen. Sin embargo, requieren de la intervención humana, de forma que es necesario definir en el propio

algoritmo la cantidad de *clusters* en los que se van a agrupar los colores de la imagen. Si el resultado obtenido no es adecuado, es posible redefinir dicho número de *cluster* para procesar la imagen nuevamente hasta obtener resultados aceptables.

Como posible solución, la librería *scikit-learn* de Python ofrece una variante del algoritmo *K-Means*, llamado *K-Means++* [12], el cual inicializa los centroides de forma que estén distantes entre sí. Esto incrementa la probabilidad de obtener mejores resultados que con una inicialización aleatoria o mediante tanteo.

Existe otra alternativa para resolver este problema relacionado con la autonomía de los procesos: aplicar redes neuronales en las que la intervención humana en la segmentación de la imagen sea opcional. No obstante, presentan el problema de que necesitan ser entrenadas cada vez que se requiere segmentar una nueva imagen, y esto hará que los datos de entrenamiento no puedan ser reutilizados con cualquier otra imagen que tenga diferentes características.

Por otro lado, se ha comentado anteriormente que el coste computacional del algoritmo *K-Means* es O(*n*k*i*), donde *n* es el número de datos del *dataset*, *k* es la cantidad de centroides seleccionados e *i* es el número de iteraciones. Sin embargo, existe otra variante de este algoritmo, llamado *Enhanced K-Means Algorithm* (Algoritmo *K-Means* mejorado) [31], cuyo coste es de O(*n*k*). Mientras que el algoritmo original computa la distancia entre cada punto y todos los centroides, lo cual es muy costoso, especialmente si el *dataset* es muy grande, la variante mejorada de *K-Means* almacena la distancia de cada punto al *cluster* más cercano, y en cada iteración se compara la nueva distancia con la distancia almacenada. Si la nueva distancia es menor o igual que la anterior, el punto se clasifica en ese *cluster*, y no hace falta computar las demás distancias de ese punto al resto de *clusters*.

5) Mala visualización del grafo mediante la librería Networkx

Una de las intenciones de este proyecto ha sido intentar programarlo en su mayoría en Python. Sin embargo, los modelos de grafo obtenidos mediante las funciones de la librería *Networkx* (*spring_layout, draw, draw_networkx_edge_labels*) no ofrecen una estética adecuada debido que los pesos de las diferentes aristas se superponen entre sí, dificultando extremadamente la comprensión del grafo en caso de que la cantidad de nodos (colores de la paleta) sea mayor que 3.

Como solución, se ha utilizado la herramienta *Graph Online* [30] que permite introducir una matriz de pesos, para a partir de ahí, editar manualmente los nodos, aristas, pesos y la estética del grafo para que sea comprensible a primera vista.

6) Coste computacional de replicar el patrón de diseño

Tal y como se ha explicado en la explicación del proceso a seguir y en la demostración realizada, el coste computacional de replicar un patrón de diseño es de O(k^r), donde *r* es la cantidad de regiones en las que se ha segmentado la imagen sobre la que se quiere replicar el patrón, y *k* es el número de colores de la paleta. Además, para cada posible combinación de los colores habría que calcular su correspondiente matriz de

pesos, y calcular la distancia respecto a la matriz de pesos de la imagen original. Hacer este proceso para k^r imágenes es inviable y muy costoso. Por tanto, la solución por la que se ha optado consiste en mostrar un pequeño número de soluciones totalmente aleatorias, y una buena solución obtenida tras observar los valores del grafo.

Otra posible solución sería agrupar aquellas regiones con un área inferior a un umbral determinado, de forma que se reduciría considerablemente la cantidad *r* de segmentos. Además, esto permitiría obtener más precisión a la hora de replicar el patrón de diseño, ya que gran número de las regiones segmentadas con un área pequeña se deben a imprecisiones del algoritmo de segmentación, y pueden ser agrupadas. Sin embargo, es posible que la cantidad de combinaciones siga siendo elevada.

Este apartado queda pendiente de mejorar en trabajos futuros, especialmente el proceso de obtención de la mejor solución, que realmente es lo que interesa.

9. Trabajos futuros

Como líneas futuras a seguir en este proyecto, una opción que mejoraría el modelo implementado a nivel de producto sería integrarlo en una aplicación iterativa, de forma que el usuario únicamente deba seleccionar dos imágenes de entrada (una sobre la que se extrae el patrón de diseño, y otra sobre la que se replique) y el número de colores que constituyen la paleta. Para que la aplicación sea totalmente robusta en cuanto al procesado de imágenes, la función *ProcesarImg()* debe ser mejorada para que realice el correcto *background subtraction* de forma automática, sin necesidad de indicarle las dimensiones de la imagen y las posiciones del objeto central.

También es posible valorar otros algoritmos de *machine learning* que funcionen más eficientemente que el algoritmo *K-Means,* como por ejemplo los nombrados en el apartado de problemas surgidos en el proyecto *(K-Means ++*, *Enhanced K-Means Algorithm* o redes neuronales).

Respecto al problema sobre llevar a cabo la replicación del patrón de diseño en una imagen segmentada en regiones, se debe integrar en dicha aplicación la forma óptima de calcular la mejor combinación, es decir, aquella que más se asemeje al patrón extraído de la imagen original.

En cuanto a las métricas para observar la calidad de los resultados, en este TFG se ha empleado la distancia euclídea. Sin embargo, también se podrían calcular otras métricas como el MSE (*Mean Square Error*) o RMSE (*Root Mean Square Error*) que permiten comparar una matriz real frente a una matriz predicha, lo cual se ajustaría también al error que queremos calcular.

Por otro lado, podrían ampliarse los objetivos a conseguir en el proyecto, desde un análisis totalmente focalizado en comparar paletas de colores de prendas de diferentes épocas para conocer cómo varía la tendencia de la moda, hasta un análisis de predicción de la moda en el futuro.

10. Anexos

ANEXO I: Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**		x		
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6**. Agua limpia y saneamiento.**		x		
ODS 7. **Energía asequible y no contaminante.**			x	
ODS 8. **Trabajo decente y crecimiento económico.**			x	
ODS 9. **Industria, innovación e infraestructuras**		x		
ODS 10. **Reducción de las desigualdades.**				x
ODS 11. **Ciudades y comunidades sostenibles.**		x		
ODS 12. **Producción y consumo responsables.**	x			
ODS 13**. Acción por el clima.**	x			
ODS 14. **Vida submarina.**		x		
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Tabla 5. Objetivos de desarrollo sostenible

Reflexión sobre la relación del TFG con los ODS y con el/los ODS más relacionados

A día de hoy, la industria textil es la segunda más contaminante del mundo, por detrás de la petrolera y delante de la ganadera, y causa graves problemas medioambientales como la contaminación de ríos y aire o el gasto masivo de agua. En Estados Unidos, la concienciación a la población sobre este grave problema ha provocado un gran incremento en el reciclado de ropa (50.000 toneladas en 1960 frente a 2'5 millones de toneladas en 2020) [33]. En España tan sólo se recicla un 12'16% de la ropa fabricada, una cantidad de 108.296 toneladas, respecto al total de 890.244 toneladas de ropa tirada anualmente [34].

Tal y como el sentido común lo manifiesta, el modelo de producción actual es insostenible, y no tomar medidas por parte de las empresas y los gobiernos hará que la situación sea irreversible en un futuro cercano. Según la ONU, la industria de la moda es responsable de la emisión del 10% del total de gases de efecto invernadero anuales (en el caso de la quema de las prendas, cada kilo de prendas quemadas emite 1'36 kilos de CO_2), además de que los tintes utilizados suponen un gran problema medioambiental debido a que contaminan el agua y destruyen la fauna y la flora de los ecosistemas [35]. Una solución propuesta por la ONU es la creación de la iniciativa llamada Alianza de las Naciones Unidas sobre la Moda Sostenible, la cual pretende presionar a los gobiernos y empresas para implantar un modelo más respetuoso con el medio ambiente.

Por otra parte, la producción textil es responsable de aproximadamente el 20 % de la contaminación mundial de agua potable, debido a todas los productos vertidos. El lavado de materiales sintéticos genera cada año unos 0,5 millones de toneladas de microfibras que acaban en los océanos [36].
Según datos de la Agencia Medioambiental Europea (*European Environment Agency*), la industria textil utilizó 79.000 millones de metros cúbicos de agua en 2015, mientras que las necesidades de toda la economía de la UE ascendieron a 266.000 millones de metros cúbicos en 2017. Para elaborar una sola camiseta de algodón, las estimaciones indican que se necesitan 2.700 litros de agua dulce, es decir, una cifra cercana a la cantidad de agua que una persona bebe en dos años y medio [37].

El 10 de abril de 2022 entró en vigor la nueva Ley de Residuos y suelos contaminados (Directiva 2018/851 de la legislación española) la cual obliga a la industria textil a reciclar sus prendas. Prohíbe al sector textil destruir los tejidos que no se puedan vender (excedente de producción o *deadstock*) y obliga a reutilizar la ropa desechada pagando una 'ecotasa' para favorecer su reciclaje y contaminar menos [38]. La normativa europea (Directiva (UE) 2018/851 del Parlamento Europeo y del Consejo, de 30 de mayo, sobre residuos) establece como obligatoria la recogida de la fracción textil de los residuos municipales, antes del 1 de enero de 2025 [34].

Como científico de datos, mediante este TFG se pretende lanzar una propuesta de posible solución a este problema. La idea principal es conseguir que las marcas de ropa vendan todos sus productos, con el objetivo de minimizar la cantidad de unidades que quedan en *stock* sin vender (y que por tanto se desechan). El objetivo a conseguir es "comprar ropa que todavía no está fabricada", de forma que mediante opiniones populares y el gusto de los usuarios, la industria textil se centre en la fabricación de

prendas con aquellas combinaciones de colores preferidas y más populares, garantizando así un gran porcentaje de las ventas con el consecuente aprovechamiento de la ropa. Para averiguar qué estilo de ropa es preferido (con estilo me refiero a la combinación de colores), se pueden hacer encuestas entre los diferentes países del mundo, o directamente se puede comenzar a analizar los patrones de moda que siguen las grandes multinacionales textiles, ya que internamente consideran los gustos de sus usuarios.

Una vez se haya recopilado toda la información anterior, se procedería a hacer un análisis similar al contenido de este TFG, elaborando las paletas de colores más populares y su distribución en diferentes prendas, de forma que este patrón sea replicable en cualquier prenda.

ANEXO II: Funciones creadas

En este anexo se muestra el código Python de elaboración propia para implementar las funciones o procesos que se han empleado en la solución propuesta.

```
def AsignarColorFondo(img, color_fondo):
    seed = (0,0)
    ImageDraw.floodfill(img, seed, color_fondo, thresh = 100)
    return img

-----------------------------------------------------------------------

def EliminarFondo(img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _,thresh = cv2.threshold(gray_img, 127, 255, cv2.THRESH_BINARY_INV
+ cv2.THRESH_OTSU)

    img_contorno = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE) [-2]
    img_contorno = sorted(img_contorno, key=cv2.contourArea)

    for i in img_contorno:
        if cv2.contourArea(i) > 10000:  #10000
            break

    mask = np.zeros(img.shape[:2], np.uint8)
    cv2.drawContours(mask, [i], -1, 255, -1)
    new_img = cv2.bitwise_and(img, img, mask=mask)
    new_img = new_img.astype('uint8')
    im2 = Image.fromarray(new_img)

    return im2

-----------------------------------------------------------------------

def procesarImg(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    mask = np.zeros(img.shape[:2],np.uint8)
    bgdModel = np.zeros((1,65),np.float64)
    fgdModel = np.zeros((1,65),np.float64)
    rect = (50,40,700,600)
    cv.grabCut(img, mask, rect, bgdModel, fgdModel, 5,
cv.GC_INIT_WITH_RECT)
    mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
    img = img*mask2[:,:,np.newaxis]
    plt.axis('off')
    plt.imshow(img)
    return img

-----------------------------------------------------------------------

def show_img_compar(img_1, img_2 ):
    f, ax = plt.subplots(1, 2, figsize=(10,10))
    ax[0].imshow(img_1)
    ax[1].imshow(img_2)
    ax[0].axis('off')
    ax[1].axis('off')
    f.tight_layout()
    plt.show()
```

```
def flattenColors(image, n):
    I = np.array(image)
    x,y,z = I.shape
    pixels = I.reshape(x*y, z)

    k_means = KMeans(n_clusters=n, random_state = 7).fit(pixels)
    preds = k_means.predict(pixels)
    clusters = preds.reshape((x,y))
    centroides = k_means.cluster_centers_
    centroides = centroides.astype(int)
    colores = [c[:3] for c in centroides]
    colores = [k.astype(int) for k in colores]

    preds_3d = np.zeros((x,y,3))
    for i in range(n):
        idx = np.where(clusters==i)
        color = colores[i]
        preds_3d[idx[0], idx[1], :] = color
    preds_3d = preds_3d.astype('uint8')
    im2 = Image.fromarray(preds_3d)

    #Para mostrar la paleta de colores
    width=300
    palette = np.zeros((50, width, 3), np.uint8)
    steps = width/len(colores)
    for idx, centers in enumerate(colores):
        palette[:, int(idx*steps):(int((idx+1)*steps)), :] = centers

    plt.imshow(palette)
    return show_img_compar(image, im2), centroides, clusters

-----------------------------------------------------------------------

def crearMAdy(b):
    ncol = 7
    M = np.zeros((ncol,ncol))
    W = b.shape[1]    #W es la anchura de la imagen
    H = b.shape[0]    #H es la altura

    for i in range(1, H-1):
        for j in range(1, W-1):
            ColorActual = b[i][j]
            ColorDcha = b[i][j+1]
            ColorAbajoDcha = b[i+1][j+1]
            ColorAbajo = b[i+1][j]
            ColorAbajoIzda = b[i+1][j-1]

            M[ColorActual][ColorDcha] += 1
            M[ColorDcha][ColorActual] += 1

            M[ColorActual][ColorAbajoDcha] += 1
            M[ColorAbajoDcha][ColorActual] += 1

            M[ColorActual][ColorAbajo] += 1
            M[ColorAbajo][ColorActual] += 1

            M[ColorActual][ColorAbajoIzda] += 1
            M[ColorAbajoIzda][ColorActual] += 1

    M2 = np.round(M / np.sum(M) * 100, 4)
    return M2
```

```
def OrdenarDataframe(df):
    df1 = df.sort_index(axis = 0)
    df2 = df1.sort_index(axis = 1)
    return df2

-----------------------------------------------------------------------

def DrawGraph(G):
    pos = nx.spring_layout(G, k=24.25)
    edge_labels = dict([((u, v,), d['weight']) for u, v, d in
G.edges(data=True)])
    nx.draw(G, pos, with_labels=True, node_color = l_colores)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
    plt.show()

-----------------------------------------------------------------------

#Proceso para segmentar una imagen en regiones y etiquetarlas

painting = imread('ruta de la imagen')
painting = cv2.cvtColor(painting, cv2.COLOR_BGR2RGB)
#plt.imshow(painting)

gray_painting = rgb2gray(painting)
binarized = gray_painting<0.7
#imshow(binarized)

square = np.array([[1,1,1], [1,1,1],[1,1,1]])

def multi_dil(im, num, element=square):
    for i in range(num):
        im = dilation(im, element)
    return im

def multi_ero(im, num, element=square):
    for i in range(num):
        im = erosion(im, element)
    return im

multi_dilated = multi_dil(binarized, 1)
area_closed = area_closing(multi_dilated, 1)
multi_eroded = multi_ero(area_closed, 1)
opened = opening(multi_eroded)
#plt.imshow(opened)

label_im = label(opened)
regions = regionprops(label_im)
plt.imshow(label_im)

-----------------------------------------------------------------------

#Pseudocódigo para el proceso de asignación de los colores de una
paleta a una imagen segmentada y con regiones etiquetadas

lista = label_im.tolist()

#paleta = lista de n tuplas de 3 elementos

for li in lista:
    for index, value in enumerate(li):
```

```
        if value == 0:
            li[index] = paleta[0]  #Es el color principal de la
prenda. Debe ser el color con más píxeles entre sí, sin incluir el
color del fondo

        elif value == 1:
            li[index] = paleta[1]  #El el color del fondo de imagen

        elif value == 2 or value == 3 or ... or value == a-1:
            li[index] = paleta[u] #Un color de la paleta distinto al
del fondo de imagen y al principal de la prenda

        elif value == a or ... or value == b-1:
            li[index] = paleta[w]

        .
        .
        .

        elif value == h ... or value == j-1:
            li[index] = paleta[y]

        else:
            li[index] = paleta[z]
```

```
#Obtener el error (distancia euclídea) entre dos matrices (la imagen
original de la que se ha extraído el patrón de diseño
# y la nueva imagen sobre la que se ha replicado)

op1 = np.linalg.norm(df de la imagen original - df de la nueva imagen
obtenida)
op2 = np.round(np.sqrt(np.sum(np.square(df de la imagen original - df
de la nueva imagen obtenida))),3)
```

11. Referencias

[1] *StyleSage*. Recuperado el 25 de enero de 2022. Disponible en https://stylesage.co/

[2] Abadía, J. *Análisis de colores: cómo analizar tendencias de moda automáticamente*. (20 de marzo de 2017). [Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=yiWlwPrIqzo

[3] Lai P. y Westland S. *Machine Learning for colour Palette extraction from fashion runways images*, International Journal of Fashion Design, Technology and Education. (2020). DOI: 10.1080/17543266.2020.1799080

[4] Pan, Q., y Westland, S. *Comparative evaluation of color differences between color palettes.* Proceedings of the IS&T/SID Color Imaging Conference, Society for Imaging Science and Technology, 110–115. (2018) doi.org/10.2352/issn. 2169-2629.2018.26.110

[5] Vittayakorn, S., Yamaguchi, K., Berg, A. C., y Berg, T. L. *Runway to realway: Visual analysis of fashion. Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, WACV, 2015, 951–958. (2015) doi.org/10. 1109/WACV.2015.131

[6] Al-Halah, Z., Stiefelhagen, R., y Grauman, K. *Fashion forward: Forecasting visual style in fashion*. Proceedings of the IEEE International Conference on Computer Vision, 2017, 388–397. (2017) doi:10.1109/ICCV.2017.50

[7] Lin, Y., y Yang, H. *Predicting Next-Season Designs on High Fashion Runway*. (Agosto de 2019). Recuperado de http:// arxiv.org/abs/1907.07161 en febrero de 2022.

[8] Álvarez Nuñez, M. y Parra Muñoz, J. *Teoría de grafos. (2013).*

[9] Lentin y Ribaud, *Algebra Moderna*, ED-Aguilar.

[10] Jordan Lluch, C.). Aplicaciones biyectivas [Archivo de Vídeo]. Universitat Politècnica de València (Polimedia). (30 de septiembre de 2021). https://media.upv.es/#/portal/video/207cc6d0-d066-11e8-85bc-5bc83fa995d1

[11] Colaboradores de la Universidad de Oviedo. *El algoritmo k-means aplicado a clasificación y procesamiento de imágenes.* https://www.unioviedo.es/compnum/laboratorios_py/new/kmeans.html#El-algoritmo-k-means

[12] Scikit-learn, https://scikit-learn.org/stable/modules/clustering.html#k-means

[13] Simovici, Dan A. *Linear Algebra Tools for Data Mining*, Chapter 15. World Scientific. (2012).

[14] Plataniotis, K., y Venetsanopoulos, A. *Color Image Processing and Applications (1ª ed.). Springer.* (2000).

[15] Wyszecki, G., y Stiles, W. *Color Science, Concepts and Methods, Quantitative Data and Formulas (2a ed.).* John Wiley. (1982).

[16] Barber, F. *¿Qué es un píxel y cuál es su función en fotografía digital?* (2016). Recuperado en abril de 2022. Disponible en https://www.cocoschool.com/pixel-funcion-fotografia-digital/

[17] Gonzalez, R., y Woods, R. *Digital Image Processing (3era ed.).* Pearson Prentice Hall. (2008).

[18] Pérez Vega, C., *VISION, LUZ Y COLOR*. UNIVERSIDAD DE CANTABRIA, Dpto. de Ingeniería de Comunicaciones. (2006).

[19] Cortés Parejo, J., *La Percepción del Color*. (2000).

[20] González, R. y Woods, R. *Digital Image Processing (2nd ed.)*. Pearson Prentice Hall. (2002).

[21] EcuRed. *Modelo HSV*. (2002). Disponible en https://www.ecured.cu/Modelo_HSV Recuperado en febrero de 2022.

[22] N. Y. K. L. Y. V. D. C. S.H. Ong, *Segmentation of color images using a tow-stage self-organizing network*, Elsevier, pp. 279-289, 2002.

[23] McKormick, C., *HOG Person Detector Tutorial*. (2021). http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/

[24] Pardo Herrera, C.J. *Detección de peatones con Python y OpenCV (HOG + SVM). Visión por computador.* Disponible en https://carlosjuliopardoblog.wordpress.com/2018/06/28/deteccion-de-peatones-con_D-python-y-opencv-hog-svm/ Recuperado en mayo de 2022.

[25] Colaboradores de Wikipedia. *Local Binary Patterns.* Wikipedia, La enciclopedia libre, 2022. Disponible en https://en.wikipedia.org/wiki/Local_binary_patterns

[26] Castillo García, J., *Algoritmos para la restauración de imágenes digitales en colores*, Ciudad de México, 2015, pág 21.

[27] Colaboradores de Wikipedia. *Segmentación (procesamiento de imágenes)*. Wikipedia, La enciclopedia libre, 2022. Disponible en https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)

[28] Algoritmo *GrabCut* para extraer objetos en primer plano. *Programador Clic.* Disponible en https://programmerclick.com/article/1702575732/

[29] *Scikit-image*. Disponible en https://scikitimage.org/docs/stable/api/skimage.morphology.html#skimage.morphology.opening Recuperado en abril de 2022.

[30] *Graph Online*. Disponible en https://graphonline.ru/en/

[31] Fahim A.M., Salem A.M., Torkey F.A. y Ramadan M.A*., An efficient enhanced k-means clustering algorithm*. Egipto. (15 de mayo de 2006).

[32] *Procesamiento de imágenes con Python: componentes conectados y etiquetado de regiones*. ICHI.PRO. Disponible en Procesamiento de imágenes con Python: componentes conectados y etiquetado de regiones (ichi.pro)

[33] United States Environmental Protection Agency (EPA). *Facts and Figures about Materials, Waste and Recycling*. Recuperado el 3 de junio de 2022. Disponible en https://www.epa.gov/facts-and-figures-about-materials-waste-and-recycling/textiles-material-specific-data

[34] MODA RE. *Análisis de la recogida de la ropa usada en España.* (Madrid, 2021). Disponible en https://modare.org/

[35] Aramendía, H. *Luchar contra la 'fast-fashion', o cómo evitar que se tiren a la basura más de 16.000 toneladas de ropa al año*. laSexta. (31 de julio de 2019). Disponible en https://www.lasexta.com/noticias/ciencia-tecnologia/luchar-fastfashion-como-evitar-que-tiren-basura-mas-16000-toneladas-ropa-ano_201907305d41a6010cf28aa877751e79.html

[36] Noticias Parlamento Europeo. *El impacto de la producción textil y de los residuos en el medio ambiente.* (21 de abril de 2022). Disponible en https://www.europarl.europa.eu/news/es/headlines/society/20201208STO93327/el-impacto-de-la-produccion-textil-y-de-los-residuos-en-el-medio-ambiente#:~:text=Seg%C3%BAn%20las%20estimaciones%2C%20la%20producci%C3%B3n,que%20acaban%20en%20los%20oc%C3%A9anos.

[37] European Environment Agency. *Use of freshwater resources in Europe*. (1 de junio de 2022). Disponible en https://www.eea.europa.eu/data-and-maps/indicators/use-of-freshwater-resources-3/assessment-4

[38] Pavés, V. *El precio de la ropa subirá porque su reciclaje será obligatorio. Verde y Azu*l. (21 de noviembre de 2021). Disponible en https://verdeyazul.diarioinformacion.com/el-precio-de-la-ropa-subira-porque-su-reciclaje-sera-obligatorio.html