

Sistema de Recuperación de Imágenes Artísticas Basado en Contenido (CBIR)

Cristina Fernández Gómez, Ester Esteban Bruña, Fátima Fuchun Illana Guerra

Abstract—El siguiente informe contiene toda la documentación del proceso de creación de un sistema CBIR, así como la descripción detallada de la herramienta, resultados de rendimiento y conclusiones obtenidas.

Index Terms—CBIR, Histograma, Autoencoder, VGG19, BoW, Movimientos Artísticos, Python

I. INTRODUCCIÓN

En el siguiente proyecto (código en [1]) se plantea la idea de desarrollar un sistema de recuperación de imágenes artísticas basado en contenido. El objetivo será desarrollar una herramienta que, mediante técnicas de extracción de características y aprendizaje automático, sea capaz de buscar y recuperar obras de arte similares.

Un sistema de tales características es capaz de ofrecer diversas aplicaciones, no solo en el aspecto más administrativo como puede ser la clasificación de obras de arte, sino también en el ámbito analítico. Concretamente el análisis artístico, donde podríamos descubrir relaciones estilísticas o temáticas entre obras de diferentes épocas, estilos o artistas.

En el ámbito de las obras de arte, la extracción basada en contenido presenta varios desafíos, principalmente debido a la gran diversidad que existe en el arte. Incluso dentro de un mismo movimiento artístico las obras pueden variar significativamente en términos de color, textura o composición, lo que dificulta establecer patrones consistentes.

Además, existen casos en los que dos obras visualmente similares no comparten el mismo movimiento, complicando aún más la tarea de categorización. Por ello, en este estudio, aunque se ha intentado refinar todo lo posible, no nos centramos tanto en la precisión estricta ya que reconocemos y valoramos el carácter diverso y subjetivo del mundo artístico.

II. CONJUNTO DE DATOS

A. Obtención del Dataset

Para comenzar el proyecto, era fundamental contar con un conjunto de datos amplio y de alta calidad sobre el cual construir nuestro sistema de clasificación. Estuvimos buscando en varios foros y repositorios públicos de datos hasta que dimos con uno que se ajustaba a nuestras necesidades.

El dataset en cuestión lo encontramos en el repositorio de Kaggle, llamado 'WikiArt Art Movements/Styles' [2]. Se trata de un conjunto de imágenes ordenadas por

estilos, 13 en total: Arte Académico, Modernismo, Barroco, Expresionismo, Arte Japonés, Neoclasicismo, Primitivismo, Realismo, Renacentista, Rococó, Romanticismo, Simbolismo y Medieval Occidental.

El conjunto en su totalidad tenía un tamaño de 30GB en total, por lo que era inviable trabajar con el conjunto completo. A falta de herramientas para descargar solo una pequeña parte de cada carpeta decidimos descargar las 30 primeras imágenes de cada movimiento manualmente. Escogimos, además, aquellas que tuvieran menos peso para no sobrecargar la velocidad de procesado ya que había imágenes de entre 10 y 20 MB.

B. Reducción del Número de Clases

Una vez descargado el dataset, nos pusimos a realizar pruebas con el conjunto y, tras varios intentos, nos dimos cuenta de que, al tener tantos movimientos y tan pocas imágenes por cada uno, era inviable obtener, para cada consulta, suficientes imágenes similares que además pertenecieran al mismo movimiento. Esto se debía a que, dentro de las pocas imágenes de entrenamiento disponibles para cada uno, muchas no se parecían entre sí, lo que complicaba aún más la tarea de clasificación.

Por ello, para mejorar la precisión del sistema, decidimos trabajar únicamente con los movimientos que presentaban imágenes más o menos uniformes, lo que nos permitió obtener una clasificación más consistente. Además, de esta forma, pudimos, en contra posición, añadir un mayor número de imágenes de entrenamiento por movimiento, facilitando la búsqueda y detección de similitudes.

Concretamente decidimos mantener los siguientes estilos: Modernismo, Barroco, Arte Japonés, Realismo, Rococó y Medieval Occidental, con 60 imágenes cada uno.

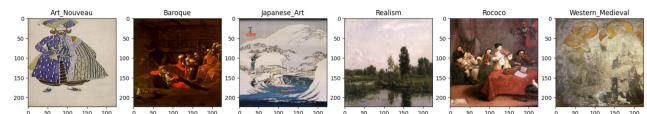


Fig. 1. Previsualización

C. División del Dataset y Procesado Inicial

Ahora que tenemos concretado cuál es el conjunto final y cómo lo obtuvimos, procedemos a explicar los pasos iniciales de tratamiento:

Primero dividimos el conjunto en 2 conjuntos: el conjunto de entrenamiento, para extraer las características y generar los índices de búsqueda, y el conjunto de test, con las imágenes de prueba que podremos ingresar en las consultas. Como en total contamos con 360 instancias y sabemos que es clave tener un número considerable de vectores de características por cada movimiento, decidimos utilizar las primeras 54 imágenes de cada movimiento para el entrenamiento y las últimas 6 para test. De esta forma tendremos el 90% de instancias en 'Train' y el 10% en 'Test'.

Para ello creamos un nuevo directorio con las imágenes separadas en 2 carpetas, Train y Test, y añadimos al nombre de cada imagen la etiqueta correspondiente a su movimiento artístico para poder estudiar posteriormente la precisión de los resultados obtenidos.

Después, en esta parte, aprovechamos también para crear una base de datos con los nombres de los imágenes pertenecientes al entrenamiento, almacenándolos en un fichero csv. Esto será necesario para el correcto funcionamiento de la interfaz posteriormente. Además, como las imágenes han sido seleccionadas manualmente podemos asegurar que no existen duplicados en el conjunto.

El desarrollo de esta primera parte del proyecto se encuentra en el fichero 'PreprocesadoImagenes.ipynb'.

III. EXTRACTORES DE CARACTERÍSTICAS

Para poder desarrollar este sistema de Recuperación de Imágenes Basadas en el Contenido, es crucial realizar una extracción de características de las diferentes imágenes, ya que, de esta manera, podremos representarlas y compararlas correctamente. Estos extractores analizan elementos clave de una imagen, dependiendo del extractor utilizado, se centrará en unas características u otras. Permiten convertir una imagen en valores numéricos, facilitando así la labor de comparar las imágenes y buscar las más similares.

En este trabajo, hemos utilizado un total de 5 extractores de características, unos más sencillos, como el histograma de color, hasta otros más complejos, como una red neuronal. Para elegir qué extractores emplear nos centramos en buscar características que puedan ser representativas en obras de arte. Concretamente tenemos:

- *Hisograma de Color*: Como muchos movimientos artísticos tienen paletas de colores distintivas y el color es un elemento crucial en cuadros, este descriptor nos va a proporcionar información útil en el contexto artístico.
- *Hisograma de Textura*: Esta característica nos va a permitir distinguir técnicas artísticas, como pinceladas al óleo, acuarela o grabados.
- *Bag of Words*: Este método nos va a ser útil para identificar elementos recurrentes, detalles arquitectónicos o formas específicas que se repitan entre movimientos.
- *VGG19*: Esta red CNN nos será útil porque está preentrenada para extraer características visuales jerárquicas

y complejas, como texturas, patrones, formas y colores, que son esenciales para distinguir entre diferentes estilos artísticos.

- *Autoencoder*: Es capaz extraer representaciones abstractas que no son fácilmente detectables con métodos tradicionales.

Crearemos un cuaderno jupyter para cada extractor, almacenando los índices correspondientes de cada uno en la carpeta database, para luego ser utilizados en la interfaz y comprobar su funcionamiento.

A. Histograma de Color

El histograma de color es una representación gráfica de la distribución de los colores de los que están compuestas las imágenes. Es decir, representan el número de píxeles que tienen los colores de cada una de las franjas de color de las que está compuesta la imagen. Se puede construir sobre distintos espacios de color, como RGB o HSV.

Su uso nos permitirá identificar patrones cromáticos, comparar estilos artísticos y agrupar cuadros que pertenezcan al mismo movimiento. El extractor podrá identificar paletas de colores similares en las pinturas, ya que estas varían mucho según el movimiento artístico.

Para llevar a cabo este proceso, comenzaremos realizando un preprocesamiento de las imágenes, redimensionándolas y convirtiéndolas en formato RGB para después convertirlas en un array.

Realizado el preprocesado, el siguiente paso será crear un dataframe donde almacenaremos las imágenes y sus correspondientes versiones preprocesadas. Creamos dos columnas: una con la ruta de la imagen y otra con los valores preprocesados. Después creamos una lista que almacene los valores de la columna preprocesada, que será el conjunto de valores con el que trabajaremos a continuación. Este proceso se repite para todos los extractores.

Una vez hecho esto, pasamos a calcular el histograma con la función cv2.calcHist(), concatenaremos los histogramas de cada canal (Red, Green, Blue) y aplandaremos el vector para que sea unidimensional. Extraídas las características, las procesaremos para convertirlas en un array numpy y nos aseguraremos de que estén en el formato float32.

El siguiente paso será almacenar el índice de características FAISS. Al ser el primer extractor, lo denominaremos 'feat_extract_1'. En caso de que el archivo no exista, se creará extrayendo los histogramas de las imágenes haciendo uso de la función anterior, se preprocesarán y almacenarán en el índice. Este índice se creará haciendo uso de la distancia euclídea L2.

Para comprobar su uso, utilizaremos el conjunto de test, introduciremos una imagen a comparar de prueba e imprimiremos el resultado [Figura 2].



Fig. 2. Resultados Extractor 1

Todo el proceso se encuentra detallado en el cuaderno 'Extractor1-HistogramaColor.py'

B. Histograma de Textura

Para analizar la textura de las imágenes vamos a utilizar la Matriz de Co-Ocurrencia de Niveles de Gris (GLCM), que consiste en una representación estadística de cómo se distribuyen los niveles de gris en una imagen considerando las relaciones espaciales entre píxeles. Específicamente se trata de una matriz bidimensional donde los puntos (*i, j*) corresponden al número de veces que un píxel con intensidad *i* aparece adyacente a un píxel con intensidad *j*, según la distancia y dirección seleccionada como parámetro.

A partir de dicha matriz vamos a calcular distintas propiedades:

- **Contraste:** Mide la diferencia de intensidad entre píxeles adyacentes. En obras de arte, un valor alto de contraste se traduce, por ejemplo, en bordes nítidos o sombras marcadas, por ejemplo.
- **Disimilitud:** Similar al contraste pero con un peso lineal. Una baja disimilitud en obras de arte puede significar que los colores se difuminan o mezclan.
- **Homogeneidad:** Evalúa cuán similares son los valores de intensidad de los píxeles vecinos. Con esto podremos evaluar si las superficies son suaves y uniformes, como ocurre en muchos paisajes, por ejemplo.
- **ASM (Angular Second Moment):** Mide la uniformidad. Un patrón alto podría indicar la presencia de estructuras más repetitivas como mosaicos.
- **Energía:** Mide la uniformidad. La energía es relevante para detectar texturas ordenadas y con poca variabilidad, lo que es característico de pinturas que usan técnicas de pintura plana o fondos uniformes.
- **Correlación:** Mide la relación lineal entre los píxeles. Este valor es útil cuando se buscan texturas coherentes, como en obras donde se usan patrones regulares

Además, para el cálculo de la matriz establecemos una distancia de 5, para ser capaces de detectar detalles como pinzeladas de distintos tonos, y un ángulo de 45, puesto que en las obras de arte no suelen estar del todo presentes las formas completamente rectas y, por ejemplo, las pinzeladas suelen estar trazadas con algo de inclinación.

Repetimos entonces el proceso de extracción de características del conjunto de entrenamiento y la creación y almacenamiento del índice FAISS, en este caso en el fichero 'feat_extract_2.index'. Para comprobar su uso, utilizaremos el conjunto de test, introduciremos una imagen a comparar de prueba e imprimiremos el resultado [Figura 3].



Fig. 3. Resultados Extractor 2

Todo este proceso se encuentra desarrollado en el cuaderno 'Extractor2-Texturas.ipynb'.

C. Bag Of Words

La técnica de Bag of Words consiste en representar un documento, en este caso las imágenes, como bolsas de palabras. Estas palabras son las características que hemos extraído de las imágenes, haciendo uso de extractores como ORB o SIFT.

Al igual que en los extractores anteriores, el primer paso será preprocesar las imágenes de entrenamiento y almacenarlas. Comenzamos creando la función `get_patches`, con la cual se extraerán una serie de subimágenes (parches) de cada imagen. Estos parches son pequeñas áreas de la imagen con las que crearemos el diccionario visual, tendremos que establecer el número de parches que queremos como parámetro. Extraídas estas partes de la imagen, la función las redimensionará, quedando todos aplazados en 2D para poder utilizarlos como vectores con la función `extract_patches_2D`.



Fig. 4. Patches

La parte principal de la función definirá el tamaño del patch a (30,30), extrayendo 250 parches aleatorios para cada imagen dada. Para extraer las características de los parches generados, usaremos el algoritmo de detección ORB, encargado de extraer los puntos clave, los cuales son las posiciones de la imagen fácilmente distinguibles, y

descriptores, que consisten en representaciones de cada punto clave. Para ello, crearemos dos listas: keypoints y descriptors, donde almacenaremos los datos. Iremos recorriendo cada parche extraído y le aplicaremos ORB para almacenar las características en las listas creadas.

En la función principal `model_feature_extractor2` extraeremos las características. Comenzamos cargando el codebook, el cual consiste en un diccionario que contiene un conjunto de palabras visuales. Es decir, son patrones visuales representados por vectores que describen las características locales de las imágenes. Este codebook se genera de la siguiente manera. Una vez extraídos los descriptores mediante la técnica ORB, se agrupan en k clusters, donde cada cluster representa una palabra visual. Los centroides de esos clusters serán los que compongan el codebook. Al procesar imágenes nuevas, los descriptores se asignan a un centroide del codebook, generando un histograma de palabras visuales que se usan para la clasificación. Este codebook actuará como diccionario de características visuales.

Cargado el notebook, inicializamos el detector ORB, con el que generaremos los descriptores de la imagen. Para cada parche extraído, calcularemos los descriptores usando el detector y lo asignaremos al centroide más cercano del codebook, haciendo uso de la función `vq`, obteniendo como resultado un conjunto de palabras visuales.

El siguiente paso será la creación de un vector de frecuencias, el cual capturará la distribución de palabras visuales en la imagen. Cada posición del vector representa cuántas veces una palabra visual aparece en los descriptores de un parche. Estos vectores serán lo que retorne la función.

Obtenidos los vectores, pasamos a preprocesarlos para poder llevar a cabo el análisis. Los convertiremos en arrays que se aplanarán para que estén todos en la misma dimensión.

Finalmente, crearemos el índice FAISS, el cual almacenaremos en el archivo '`feat_extract_3.index`'. Se procesarán las imágenes de entrenamiento y extraerán los descriptores, generando un conjunto de vectores de características que se añadirán al índice para usarlos en consultas futuras.

Para comprobar su uso, utilizaremos el conjunto de test, introduciremos una imagen a comparar de prueba e imprimiremos el resultado [Figura 5].

Todo este proceso se encuentra desarrollado en el cuaderno '`Extractor3-BagOfWords.ipynb`'.

D. CNN-VGG19

Para el cuarto extractor hemos implementado el CNN-VG199, una red neuronal con 19 capas ideal para el procesamiento de imágenes. Está preentrenado con miles de imágenes de la web ImagenNet, por lo que es perfecta para



Fig. 5. Resultados Extractor 3

nuestro trabajo. Esta compuesto por capas convolucionales y capas totalmente conectadas. Estas últimas tienen un tamaño excesivamente grande y se adaptan mucho a las imágenes de entrenamiento. Aunque son características útiles, para nuestro CBIR son demasiado específicas, y es mejor quedarnos con unas más generales, como las que ofrecen las capas convolucionales.

Como en los extractores anteriores, el primer paso será preprocesar las imágenes y almacenarlas. Una vez hecho esto, pasamos a cargar el modelo VGG19 con pesos ya preentrenados a partir de millones de imágenes. La salida será la penúltima capa, antes de que se reduzcan a etiquetas específicas, es decir, antes de que estén ligadas a una clase en particular, se usa como un descriptor visual más genérico.

Para que la red funcione correctamente, redimensionaremos las imágenes de entrada a un tamaño de 224x224. Dado que espera recibir un lote, también tendremos que añadirle una dimensión adicional con la función `np.expand_dims`, de manera que los array de las imágenes queden convertidos en un batch. Además, se normalizarán los píxeles para que queden en un rango de [-1,1] y se harán las transformaciones necesarias mediante la función `preprocess_input`.

Preprocesadas las imágenes, pasarán por el modelo para obtener sus características profundas, obteniendo como salida un tensor multidimensional que habrá que aplanar para convertirlo en un vector unidimensional.

Extraídas las características, el siguiente paso será almacenarlas en el índice FAISS, en este caso, en el archivo '`feat_extract_4.index`'. Para comprobar su uso, utilizaremos el conjunto de test, introduciremos una imagen a comparar de prueba e imprimiremos el resultado [Figura 6].



Fig. 6. Resultados Extractor 4

Todo este proceso se encuentra desarrollado en el cuaderno 'Extractor4-CNNN-VGG19.ipynb'.

E. Autoencoder

Autoencoder es un tipo de red neuronal diseñada para aprender representaciones compactas de los datos de entrada, generalmente con el objetivo de reducir la dimensionalidad o para eliminar ruido. Es un modelo de aprendizaje no supervisado y se utiliza principalmente para codificar (comprimir) y decodificar datos, con la intención de que la salida reconstruida sea lo más parecida posible a la entrada original.

En este caso, el autoencoder nos sirve como una herramienta para generar representaciones compactas y significativas (embeddings) de las imágenes que capturan sus características visuales más relevantes. El codificador del autoencoder toma como entrada una imagen y la transforma en un vector latente de menor dimensionalidad. Este vector contiene una representación compacta de las características visuales importantes, como formas, colores y texturas.

Aunque el decodificador está presente para garantizar que el espacio latente preserve suficiente información para reconstruir la imagen original, en CBIR, el decodificador no se usa directamente durante la búsqueda. Solo utilizaremos el codificador para extraer las características.

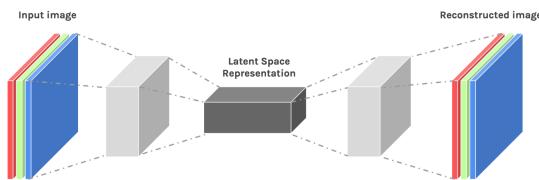


Fig. 7. Autoencoder Architecture [8].

Para la extracción de características del autoencoder sí que normalizaremos los valores de las imágenes y los vectores para reducir el tiempo de entrenamiento de la red y reducir su sensibilidad a la escala de los datos.

Ahora, la arquitectura consta de dos partes principales:

- El encoder se encarga de reducir las dimensiones de la imagen de entrada a través de varias capas convolucionales con activación ReLU, cada una con un stride de 2, lo que reduce progresivamente el tamaño espacial de la imagen mientras extrae características más abstractas.
- El decoder, por su parte, toma la representación comprimida del encoder y aumenta las dimensiones espaciales mediante capas de upsampling, utilizando convoluciones para reconstruir la imagen original.

Lo entrenamos específicamente para minimizar la pérdida entre la imagen original y la reconstruida, optimizando

el MSE, aprendiendo así representaciones compactas y significativas para las imágenes.

Concretamente la estructura de la red queda de la siguiente forma:

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
enc_conv1 (Conv2D)	(None, 112, 112, 32)	896
enc_conv2 (Conv2D)	(None, 56, 56, 64)	18,496
enc_conv3 (Conv2D)	(None, 28, 28, 64)	36,928
enc_conv4 (Conv2D)	(None, 14, 14, 128)	73,856
dec_conv0 (Conv2D)	(None, 14, 14, 128)	147,584
dec_upsampling1 (UpSampling2D)	(None, 28, 28, 128)	0
dec_conv2 (Conv2D)	(None, 28, 28, 64)	73,792
dec_upsampling2 (UpSampling2D)	(None, 56, 56, 64)	0
dec_conv3 (Conv2D)	(None, 56, 56, 64)	36,928
dec_upsampling3 (UpSampling2D)	(None, 112, 112, 64)	0
dec_conv4 (Conv2D)	(None, 112, 112, 32)	18,464
dec_upsampling4 (UpSampling2D)	(None, 224, 224, 32)	0
dec_conv5 (Conv2D)	(None, 224, 224, 3)	867

Fig. 8. Autoencoder Network

La capa 'dec_conv0' está ubicada justo después de las operaciones de codificación y antes de las de decodificación. Es decir que en esta etapa las imágenes ya han sido comprimidas en un espacio de características más reducido, concretamente (14, 14, 128). Con esto conseguiremos comparar las imágenes basándonos en sus características principales como bordes, texturas y patrones básicos.

El modelo autoencoder keras entrenado está almacenado en el fichero 'autoencoder.keras', desde donde podemos cargar la capa de convolución mencionada y utilizarla para extraer características de imágenes. Lo cual hacemos con las imágenes del conjunto de entrenamiento preprocesadas. Dichos vectores se almacenan en un índice Flat Index con FAISS al igual que los anteriores extractores y quedan registrados en el fichero 'feat_extract_5.index'.

Para comprobar su uso, utilizaremos el conjunto de test, introduciremos una imagen a comparar de prueba e imprimiremos el resultado [Figura 5].



Fig. 9. Resultados Extractor 5

Todo este proceso se encuentra desarrollado en el cuaderno 'Extractor5-Autoencoder.ipynb'.

IV. ÍNDICE FAISS

Como ya hemos mencionado, para la indexación de los vectores de características empleamos FAISS (Facebook AI Similarity Search), una biblioteca optimizada para búsquedas de similaridad. Como el tamaño de nuestro conjunto de entrenamiento no es demasiado grande empleamos, concretamente, **Flat Index L2**, que es más simple y directo que otros métodos de indexación. Este método encuentra el vector más cercano mediante el cálculo de distancias, en nuestro caso la distancia euclídea.

V. INTERFAZ

Para el desarrollo de la interfaz hemos utilizado el código de referencia proporcionado [8] donde se permite insertar una imagen, en este caso únicamente perteneciente al conjunto de test, y seleccionar cuál de los 5 extractores se quiere utilizar para el cálculo de similitud. En el caso de que quisieramos insertar una imagen externa, no perteneciente al conjunto proporcionado, tendríamos que guardarla en la carpeta de test y renombrarla con la misma estructura que el resto de imágenes (ID-MovimientoArtístico.jpg).

Además, la interfaz permite seleccionar el área de la imagen que queremos detectar. La posibilidad de seleccionar un área específica de la imagen permite focalizar la búsqueda en regiones que podrían contener información más relevante para la similitud visual. Sin embargo, dado que el sistema CBIR no clasifica objetos específicos, sino que analiza características visuales globales como colores, texturas o patrones, cualquier cambio en el área seleccionada altera los vectores de características extraídos. Esto puede mejorar los resultados si la región elegida contiene información representativa o distintiva, pero también puede empeorarlos si se excluyen detalles importantes que contribuían a una similitud más precisa en el contexto global. El código perteneciente a la interfaz está en el fichero 'Interfaz.py'

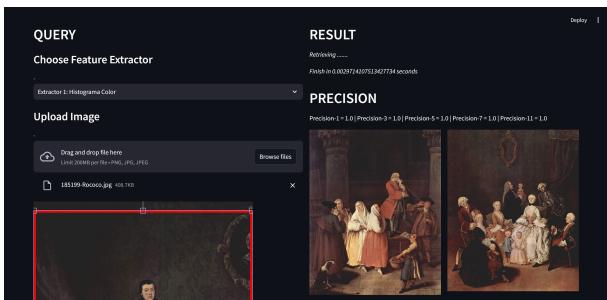


Fig. 10. Interfaz

VI. RESULTADOS

Una vez tenemos la implementación del sistema terminada procedemos a estudiar los resultados obtenidos. Para la

evaluación del rendimiento del sistema proporcionamos una métrica que se puede observar en la interfaz. Nos centramos en evaluar la precisión del sistema, comparandola en distintos casos según el número de imágenes tomadas en cuenta o devueltas. Es decir, estudiamos la precisión del sistema si se extrajera solo 1 imagen similar, 3, 5, 7 u 11. Además calculamos dicha métrica dividiendo el número de imágenes devueltas que pertenecen al mismo movimiento que la imagen de consulta entre el número total de resultados devueltos.

En función de esto podemos observar varias cosas. Para empezar, el rendimiento medio del sistema por cada extractor. Es decir, la media total del rendimiento de todas las imágenes de test en todos los casos k [Tabla 1]:

TABLE I
MEDIAS POR EXTRACTOR

Extractor	Media
Histograma de Color	0.65
Texturas	0.27
Bag Of Words	0.44
VGG19	0.53
Autoencoder	0.83

Como podemos ver, en general el extractor que mejor precisión ha mostrado es el Autoencoder y el que peor rendimiento ha tenido son las texturas. Esto tiene sentido ya que el Autoencoder, debido a su mayor complejidad, es capaz de aprender características más precisas de las imágenes. Por otra parte, le sigue como segundo mejor extractor el histograma de color que, pese a su simplicidad de concepto, se sigue manteniendo como una de las mejores opciones para detección de estilos artísticos.

Además, podemos observar también cuál es el número de consultas óptimo para cada extractor [Tabla 2].

En el caso del histograma de color, hasta 3 resultados consigue mantener una precisión constante y después comienza a disminuir. Por otra parte, las texturas presentan una precisión bastante baja, sin importar demasiado el número de consultas que se realicen. Después, para el Bag Of Words parece que la primera consulta suele fallar un poco más pero entre 3 y 5 consultas mejora ligeramente. 11 vuelve a ser demasiado. VGG19, por otro lado, mantiene unos resultados bastante consistentes en todos los casos aunque a medida que vamos incrementando el k, al igual que el resto, la precisión va disminuyendo. El Autoencoder muestra también una precisión bastante constante.

Por último vamos a observar también el rendimiento de cada extractor por movimiento artístico [Tabla 3].

Para empezar, el estilo que mejor parecen detectar el Histograma de Color es el estilo Barroco y el que peor detecta es el Arte Medieval. Después el extractor de texturas detecta mejor el estilo de Arte Japonés y Arte Medieval y, por el contrario, bastante mal el Barroco. El Bag Of Words no

TABLE II
MEDIAS POR EXTRACTOR Y K

Extractor	k	Media
Histograma de Color	1	0.69
	3	0.69
	5	0.64
	7	0.62
	11	0.61
Texturas	1	0.25
	3	0.28
	5	0.28
	7	0.27
	11	0.25
Bag Of Words	1	0.44
	3	0.47
	5	0.45
	7	0.43
	11	0.39
VGG19	1	0.56
	3	0.57
	5	0.54
	7	0.50
	11	0.47
Autoencoder	1	0.86
	3	0.84
	5	0.84
	7	0.83
	11	0.78

TABLE III
MEDIAS POR EXTRACTOR Y ESTILO

Extractor	Estilo	Media
Histograma de Color	Rococo	0.78
	Baroque	0.91
	Realism	0.64
	Japanese_Art	0.46
	Art_Nouveau	0.75
	Western_Medieval	0.35
Texturas	Rococo	0.29
	Baroque	0.15
	Realism	0.29
	Japanese_Art	0.30
	Art_Nouveau	0.27
	Western_Medieval	0.30
Bag Of Words	Rococo	0.47
	Baroque	0.56
	Realism	0.33
	Japanese_Art	0.25
	Art_Nouveau	0.48
	Western_Medieval	0.50
VGG19	Rococo	0.67
	Baroque	0.16
	Realism	0.69
	Japanese_Art	0.40
	Art_Nouveau	0.64
	Western_Medieval	0.62
Autoencoder	Rococo	0.70
	Baroque	0.91
	Realism	0.99
	Japanese_Art	0.72
	Art_Nouveau	0.82
	Western_Medieval	0.85

consigue detectar del todo bien el Arte Japonés pero procesa mejor el estilo Barroco. Por otra parte, el VGG19 tiene mejores resultados en obras del Rococó y peores en obras del Barroco y, por último, el Autoencoder tiene una precisión casi perfecta en cuadros realistas y algo menor en cuadros del Rococó.

La recolecta de resultados se encuentra en el cuaderno 'Resultados.ipynb' y almacenados en el fichero excel 'Resultados_CBIR.xlsx'.

VII. CONCLUSIONES

Una vez tenemos las métricas obtenidas, podemos extraer conclusiones a partir de ellas, teniendo muy en cuenta el contexto, es decir, las características principales de cada movimiento artístico.

Para empezar, se puede ver claramente que los estilos Barroco y Rococó son muy parecidos. El Barroco se originó en el siglo XVII y el Rococó durante el siglo XVIII, a finales del Barroco, por lo que no hay duda de por qué son tan similares. Esto puede ocasionar que, visualmente una consulta puede parecer 100% exitosa cuando realmente está juntando estos 2 estilos. En ese aspecto el sistema no está fallando a la hora de extraer imágenes similares puesto que, al ser movimientos tan similares es normal que se mezclen durante la extracción. Esta es una de las razones por la cual la precisión de las etiquetas no es del todo fiable en nuestro caso, que no estamos buscando una clasificación de estilos como tal sino la obtención de imágenes similares.

No obstante, si nos centramos en el Barroco, podemos ver que se caracteriza por el uso de colores más intensos que Rococó, lo cual explica los mejores resultados en el histograma de color y, como predominan los contrastes, los detalles, la profundidad y el simbolismo, el extractor de Bag of Visual Words es capaz de capturar mejor palabras visuales características.

Por otra parte, en el Rococó predominan los colores vivos, tonos pastel, mayor luminosidad y menores cambios de intensidad. Además, se caracteriza por representar en especial reuniones sociales de la época, o por lo que permite detectar cambios más finos en las frecuencias, haciendo que el análisis de textura sea más preciso. Esta repetición de colores y elementos parece contribuir a su alta precisión en otros métodos como el hisograma de color, VGG19 y Autoencoder.

El arte medieval, con su énfasis en figuras humanas esquemáticas, jerarquía de proporciones, y un estilo bidimensional, se adapta bien a técnicas como Autoencoders, Bag of Words y VGG19 debido a su capacidad para captar patrones estructurales, características locales y complejidades en las formas y composiciones. Sin embargo, los histogramas de colores y extractores de texturas no son tan efectivos, ya que las obras medievales utilizan una paleta limitada y patrones de textura simples debido al uso de frescos y el oro, lo que limita la variabilidad en estos dominios visuales, haciendo que estas técnicas pierdan información clave para

su identificación.

Después, el arte japonés, caracterizado por su uso de acuarelas, colores brillantes y trazos suaves y ondulados, se detecta mejor con métodos de textura que otros movimientos porque ambos capturan la fluidez y sutileza de los patrones creados con pincel. Con poca complejidad tridimensional o relieve, hace que las técnicas basadas en texturas sean especialmente efectivas al resaltar los gradientes suaves y los detalles del trazo.

El realismo, por el contrario destaca por no tener un buen rendimiento en Bag of Words. Como las imágenes que tenemos sobre este movimiento son en su totalidad de paisajes, no es posible detectar palabras visuales características en ellas.

Y, por último, podemos ver que el Modernismo tiene un buen rendimiento (similar al resto) en todos los extractores por lo general. Esto puede deberse a que tiene unos rasgos muy característicos como, por ejemplo, elementos exóticos con rasgos y vestimentas exageradas. Contornos pronunciados y líneas ondulantes y entrelazadas. Sin embargo, si nos fijamos en su clasificación nos damos cuenta de que generalmente se suelen confundir con Arte Japonés y Arte Medieval. Con el primer estilo tiene sentido, puesto que el Modernismo es un movimiento que tiene influencias de otros estilos, entre ellos el arte japonés. Además, debido a sus colores vivos también se puede confundir con varias obras de arte medieval.

En conclusión, lo que podemos ver es que el sistema no es capaz de clasificar bien las imágenes debido a que nos encontramos con un problema: los movimientos artísticos no son uniformes. Es decir, por más que intentemos estudiar las características generales de los cuadros, lo cierto es que obras pertenecientes al mismo estilo artístico pueden ser visualmente completamente distintas y, por otra parte, imágenes de estilos distintos pueden ser bastante similares. Es por esto que, además, debemos tener en cuenta que, para mejorar la detección habría que hacer uso de un número mucho mayor de imágenes, puesto que al final te devuelve las imágenes más similares dentro del limitado conjunto disponible en comparación con la gran variedad de obras que hay dentro de un mismo estilo. Además, al trabajar con estilos y no objetos es mucho más difícil encontrar los patrones necesarios para la diferenciación.

Por ello, concluimos que la concordancia con los movimientos no tiene especial relación con la similitud de imágenes, que actualmente es complicado diferenciar estilos artísticos con las herramientas presentadas y que harían falta muchas más imágenes para mejorar el rendimiento.

REFERENCES

- [1] Github. *Github Repository with the code*. https://github.com/Fatima-Illana/CBIR_2024_GRUPO_H
- [2] Kaggle. *Original Art Movements Dataset*. <https://www.kaggle.com/datasets/sivarazadi/wikiart-art-movementsstyles>
- [3] Medium. *Feature Extraction of Images using GLCM*. <https://medium.com/@girishajmera/feature-extraction-of-images-using-glcm-gray-level-cooccurrence-matrix>
- [4] Pinecone. *Color Histogram Code Reference*. <https://www.pinecone.io/learn/series/image-search/color-histograms/>
- [5] Pinecone. *Bag of Visual Words Code Reference*. <https://www.pinecone.io/learn/series/image-search/bag-of-visual-words/>
- [6] Github. *Autoencoder Code Reference*. https://github.com/upm-classes/image-understanding-2021-2022/blob/main/practice2/practice_2.ipynb
- [7] Image. *Autoencoder Code Architecture Image*. <https://i0.wp.com/sefiks.com/wp-content/uploads/2018/03/convolutional-autoencoder.png>
- [8] Github. *App Reference Code*. https://github.com/upm-classes/aapi_2024/blob/main/cbir/app.py
- [9] Humanidades. *Características Barroco*. <https://humanidades.com/barroco/>
- [10] Humanidades. *Características Rococó*. <https://humanidades.com/rococo/>
- [11] Cultura Genial. *Características Modernismo*. <https://www.culturagenial.com/es/art-nouveau/>
- [12] Futuro Pasado. *Características Arte Japonés*. <https://futuropasado.com/2019/09/14/el-arte-japones/>
- [13] Canvas Lab. *Características Arte Medieval*. https://canvaslab.com/blogs/arte/como-entender-el-arte-medieval?srsltid=AfmBOor0UNYr49ufFlw5vlCRL1BmTDZWnmQ2gvBWr_3kG8bR6MautoLk