



Grado en Ingeniería Informática (Semipresencial)

Asignatura: “Visión Artificial”

Curso 2016/17 – Convocatoria: Junio 2017

Práctica obligatoria del Curso

Título: “Clasificación de tipos de pastillas usando Visión Artificial”

ALUMNO: JORGE IGNACIO BALANZ PINO

ÍNDICE

1	INTRODUCCIÓN:	3
2	OBJETIVOS A ALCANZAR	3
3	PLANTEAMIENTO DE LA SOLUCIÓN	4
3.1	Captura o adquisición de la imagen:	4
3.2	Preproceso y filtrado:	4
3.3	Segmentación:	4
3.4	Extracción de características:	5
3.5	Reconocimiento de objetos:	6
4	ARCHIVOS DE LA ENTREGA	7
5	ESTRUCTURA DEL PROGRAMA	8
6	INSTRUCCIONES DE USO	10
7	PRUEBAS, AJUSTES Y RESULTADOS	13
7.1	Umbral para la Binarización de Imágenes	13
7.2	Área y perímetro	13
7.3	K-vecinos	14
7.4	Ecualización	14
7.5	Histogramas	14
7.6	Otros descriptores	14
8	MEJORAS	14
9	FUENTES CONSULTADAS:	15

1 INTRODUCCIÓN:

El presente trabajo tiene por objeto identificar o clasificar imágenes de diferentes tipos de pastillas, píldoras y cápsulas (pastillas, en adelante) de entre una serie de tipos que se han facilitado como ejemplo. Se trata de crear una aplicación que a partir de unas imágenes de muestra, con pastillas de diferentes tipos, sea capaz de averiguar de qué tipo es una pastilla que se encuentre en una imagen y que no esté, necesariamente, entre las imágenes de muestra. De las imágenes de muestra se deben deducir las clases de pastillas con que se va a trabajar y las características que nos van a permitir distinguir unos tipos de pastillas de otros, para más tarde poder deducir de qué tipo es una pastilla a partir de su imagen.

2 OBJETIVOS A ALCANZAR

El producto de este trabajo será una aplicación capaz de realizar las siguientes operaciones:

1. Recibir una serie de imágenes de muestra de cada uno de los tipos de pastillas a reconocer, extraer sus características y almacenarlas de forma adecuada para permitir su uso posterior en las identificaciones (fase de aprendizaje);
2. Recibir una serie de imágenes de muestra de cada uno de los tipos de pastillas a reconocer, cuyo tipo es conocido, para comprobar el correcto funcionamiento de la aplicación (fase de test);
3. Reconocer o clasificar la imagen de una pastilla, cualquiera, a partir de los datos extraídos en la fase correspondiente al punto 1.

Es importante destacar que el presente trabajo se restringe a reconocer imágenes de pastillas con características similares a las imágenes que se han facilitado para las fases de aprendizaje y de test. Éstas tienen en común que las imágenes de cada pastilla tienen una iluminación más o menos regular (ni demasiado luminoso ni demasiado sombrío) y que las pastillas se encuentran sobre un fondo oscuro, casi negro. Los resultados obtenidos en esta práctica se basan en pruebas realizadas sobre imágenes con estas características (las facilitadas para la práctica): si se utilizan otros escenarios (fondos claros, iluminación demasiado alta o demasiado baja, etc...), los resultados pueden ser diferentes.

3 PLANTEAMIENTO DE LA SOLUCIÓN

Todas las imágenes procesadas por el programa se tratan del mismo modo. En general, se trata de recibir una imagen, procesarla, extraer de ella una serie de características y clasificarla según éstas. En el caso de las imágenes de muestra, las características (descriptores) extraídas servirán para caracterizar un tipo concreto de pastilla. En el caso de intentar clasificar o identificar una pastilla, se extraen sus características, igualmente, pero, en este caso, se utiliza la caracterización de cada clase de pastillas para estimar en qué clase encaja mejor.

El planteamiento para construir la solución se basa en el procesamiento clásico de imágenes para el reconocimiento, que incluye las siguientes etapas:

3.1 Captura o adquisición de la imagen:

Esta etapa no se incluye en la práctica, pero sí puede influir en los resultados que ofrece el programa que se ha construido. El programa se diseña para el proceso de imágenes que muestran **una o más pastillas**, sobre **fondo muy oscuro** y con **iluminación media** (ni demasiado baja ni demasiado alta). Las pastillas **no deben estar en contacto entre sí**, en caso de haber más de una pastilla en la imagen. Las pastillas que estén **muy próximas al borde de la imagen, no se procesan**. Tampoco se procesarán las pastillas cuyo tamaño sea excesivamente pequeño (se considerarán ruido). Así que la forma en que se captura la imagen a reconocer, es relevante para el resultado.

3.2 Preproceso y filtrado:

Las imágenes se toman desde un archivo con formato “jpg”, “png” o “bmp”, y se realizan las operaciones de preproceso y filtrado. Este proceso incluye las siguientes operaciones, en este orden:

- 1) Se convierte la imagen de color a escala de grises (función `cv2.cvtColor()`) para poder realizar correctamente los siguientes pasos;
- 2) Se aplica un filtrado de “media” (función `cv2.blur()`) para suavizar la imagen y eliminar ruido;
- 3) Se binariza la imagen (se pasa a blanco y negro, con la función `cv2.threshold()`) para poder extraer los contornos de los objetos contenidos en la imagen. En esta fase, cabe destacar que, dado que se cuenta con que las pastillas a caracterizar se van a encontrar sobre un fondo oscuro, el umbral para la binarización se toma muy bajo (se ha utilizado el 30).
- 4) Se realiza una operación de cierre morfológico (una operación morfológica de dilatación seguida de otra de erosión) para intentar unir las componentes inconexas de la imagen en blanco y negro que posiblemente pertenezcan a un mismo objeto (se hace con la función `cv2.morphologyEx()`).

3.3 Segmentación:

Una vez realizado el preproceso y filtrado de la imagen completa se realiza una operación de extracción de los contornos que puedan encontrarse en la imagen binarizada. Esto se consigue con la función `cv2.findContours()`. La función `findContours()` devuelve una lista de los contornos localizados en la imagen. Cada uno de estos contornos es susceptible de considerarse un objeto válido (una pastilla), pero se ha comprobado que existen imágenes con algunos objetos que no son una pastilla y que deben ser considerados ruido (u objetos extraños que no se deben procesar). Para diferenciar lo que debe considerarse una pastilla de lo que no, se han establecido los siguientes criterios:

- 5) El contorno debe tener una superficie mínima, que se ha establecido en 250px.
- 6) El contorno no debe estar junto al borde. Se ha establecido un margen de 30px.

Los contornos que cumplen estas restricciones son considerados pastillas y se tratan por separado. Para cada contorno con estas características, se calcula el rectángulo que lo contiene (función `cv2.boundingRect()`), se aplica un margen, para asegurar que el rectángulo comprende el objeto entero (se ha establecido en 10px por cada lado), y ese área, dentro de la imagen original, se procesa por separado como una imagen individual (una subimagen, realmente). Esta etapa es la que permite procesar varios objetos (pastillas) que se encuentran en una misma imagen.

Teniendo en cuenta que cada subimagen es una parte de la imagen original, en color, se le debe aplicar, de nuevo, el preproceso y filtrado planteados para la imagen original concreta, pero en este caso se hace sobre una imagen más “especializada”: ya sabemos que lo único que hay que encontrar es una pastilla. Al localizar los contornos correspondientes de la subimagen, sólo nos interesará el que tenga mayor superficie.

3.4 Extracción de características:

Esta es una etapa que implica decisiones que son cruciales para el correcto funcionamiento del programa y para optimizar sus resultados.

Con análisis visual de las imágenes que se facilitan como muestra, pueden extraerse las siguientes conclusiones:

- Las pastillas pueden tener varias formas: rombo, círculo, cápsula, ovalado o rectángulo redondeado. Esta diferencia entre diferentes tipos de pastillas indica que la forma puede ser un buen criterio discriminante entre diferentes clases de pastillas.
- Los tamaños de las imágenes varían incluso para el mismo tipo de pastilla. Este factor hace pensar en que necesitaremos seleccionar descriptores que sean invariantes frente al escalado.
- Las pastillas en las imágenes de prueba **no** se presentan siempre en la misma posición: unas están giradas o colocadas en diferente posición con respecto a otras. Esta circunstancia empuja a pensar en que se necesitarán descriptores que sean invariantes frente a rotaciones y traslaciones.
- Las pastillas tienen diferentes colores. Para cada clase existe un color. En algunos casos, el color de varias clases es similar, en cuyo caso, las pastillas se distinguen por su forma.

Se han investigado varios descriptores invariantes frente a rotaciones, escalados y traslaciones (momentos de Hu, SIFT o SURF, por ejemplo), pero finalmente se ha optado por dos tipos de descriptores: de forma y de color. Con estos dos tipos de descriptores se han obtenido unos resultados aceptables.

En cuanto a la forma, se ha seleccionado un descriptor invariante frente a rotaciones, escalados y traslaciones, que es la redondez (lo parecida que es una silueta o un contorno a un círculo). Este dato se consigue con la fórmula:

$$R = [P^2 / 4 * \pi * A]$$

Donde R es el factor de redondez, P es el perímetro de la figura en estudio y A es el área de esa misma figura. El valor obtenido está en el intervalo (0,1).

Teniendo en cuenta que, en muchos casos, la iluminación y las sombras de los objetos dan lugar a contornos irregulares, primeramente se optó por calcular el área y el perímetro a partir de los rectángulos que contienen a los contornos (calculados con `cv2.boundingRect()`). Posteriormente se ha

reajustado este cálculo, para mejorar los resultados en el reconocimiento, como se verá posteriormente (apartado *pruebas y ajustes*).

En cuanto al color, se han contemplado dos posibilidades:

- a. Utilizar como descriptores las medias de cada una de las componentes de color (RGB), calculadas con `cv2.mean()`.
- b. Utilizar como descriptores los componentes RGB de los colores más significativos. Para obtener los dos colores más significativos se utiliza el método de clusterización no supervisado de las K-Medias. Se ha probado a crear dos y tres centroides con este método, seleccionando los dos más representativos (en el caso de crear 3 centroides), comprobando que se obtienen mejores resultados al crear 3 centroides y seleccionar los dos más representativos.

Al comparar estas dos estrategias, se ha comprobado que se obtienen mejores resultados (más casos de éxito en los tests) utilizando el método de las k-medias (opción b), utilizando 3 clusters (tomando los dos centroides más representativos de entre 3 generados).

Los valores obtenidos mediante este método ofrecen valores enteros en el intervalo [0,255]. Para asimilar su peso frente al descriptor de forma, se normalizan los valores dividiéndolos entre 255.

Con estos descriptores se han obtenido unos resultados bastante aceptables en la fase de test y son, además, relativamente sencillos e intuitivos, así que se han establecido como los descriptores que se utilizarán para la caracterización y el posterior reconocimiento o clasificación de pastillas. Se han probado otros descriptores (por ejemplo, momentos de Hu, histogramas de color, etc.) pero no han ofrecido resultados mejores.

En total, los parámetros que se generan para cada contorno y con los que se caracteriza cada tipo de pastilla, son 7:

- Por un lado, el factor de redondez, asociado a la forma de la pastilla (1 descriptor).
- Por otro lado, los descriptores asociados al color de la pastilla. Son 2 colores (los predominantes) con 3 componentes cada uno (RGB) lo que hacen un total de 6 descriptores de color.

Por tanto, cada pastilla, en este programa, se representa mediante un vector de 7 componentes. Las componentes son de tipo real (*float32*, en el programa) y sus valores se encuentran en el intervalo (0,1).

3.5 Reconocimiento de objetos:

Una vez que sabemos las características que nos van a permitir representar una pastilla de tal forma que podamos clasificarla en un tipo concreto necesitamos un elemento que nos permita reconocer una pastilla: averiguar de qué tipo es una pastilla (al menos, de entre los tipos conocidos), aunque no la hayamos visto antes.

Para realizar este reconocimiento de imágenes se utiliza un clasificador. Se trata de un componente software que, con un entrenamiento previo, consistente en indicarle cómo son las pastillas de cada uno de los tipos, es capaz de aproximar una imagen nueva de una pastilla a alguna de las clases para las que ha sido entrenado.

En el caso de este trabajo, se ha utilizado un clasificador basado en el algoritmo **KNN** (*K-Nearest Neighbours*), que clasifica una imagen por su proximidad (“semejanza”) a otras imágenes que se le han

facilitado en la fase de entrenamiento. La **K** indica el número de vecinos más próximos que localizará el algoritmo. La clase de objeto mayoritaria entre esos **K** vecinos será la que se le asigne a la imagen que se está intentando reconocer. Este es un algoritmo supervisado y se ajusta a nuestras necesidades: conocemos el número de clases de objetos de antemano (son 20).

En el programa que se ha construido se utiliza un objeto de tipo [cv2::ml::KNearest](#), que se obtiene con la función `cv2.ml.KNearest_create()`. Dispone de las dos funciones necesarias para nuestros propósitos:

1. Función ***train()***: entrena el clasificador con los patrones y las clases que se le pasen como parámetros. Los patrones se introducen en forma de array bidimensional en el que las filas son cada uno de los patrones de entrenamiento (un patrón es vector de características que representa cada pastilla) y las columnas son cada una de las características (los descriptores). El parámetro con las clases es un array unidimensional de tipo numérico, con un número de elementos igual al número de filas (de patrones) de entrenamiento, cuyos valores representan la clase a la que pertenece el patrón de la fila que ocupa la misma posición en la colección de patrones. Una vez realizada esta fase, el clasificador será capaz de reconocer nuevos patrones mediante el algoritmo explicado previamente.
2. Función ***findNearest()***: Esta función realiza la aproximación de la clase a la que pertenece un patrón. Recibe dos parámetros: el patrón de entrada a clasificar y el valor de **K** para el algoritmo KNN que se ha explicado previamente.

Además de estas funcionalidades, el programa que se ha creado ofrece una serie de facilidades para indicar los directorios de donde se desea extraer la información para las fases de aprendizaje y de test, y que, además, realiza la fase de test de forma automática. El programa también es capaz de reconocer los objetos (pastillas) de una imagen individual, que se pase como parámetro, una vez que el clasificador ha sido entrenado.

4 ARCHIVOS DE LA ENTREGA

En la entrega se facilitan varios archivos distribuidos en varias carpetas.

En la raíz (el directorio raíz) se encuentran este documento, que es la memoria de la práctica, y el ejecutable (archivo por lotes “.bat”) que lanza el programa en un sistema Windows que tenga instalado el software necesario para ejecutar programas codificados en Python (por ejemplo, Anaconda). Además se encuentran los siguientes directorios:

1. **Carpeta “images”**: Contiene las imágenes que se han facilitado para la práctica. Las carpetas cuyo nombre empieza por “Train” contienen subcarpetas con imágenes para el entrenamiento, y las que empiezan por “Test” son imágenes para comprobar el funcionamiento del programa (comprobar que el aprendizaje ha sido eficaz y el programa realiza su función).
2. **Carpeta “PruebasSVA”**: En esta carpeta se encuentran 3 archivos de muestra para comprobar el funcionamiento del programa. Son imágenes con varias pastillas de diferentes tipos. La imagen “muestras_todo.jpg”, por ejemplo, contiene una imagen de cada uno de los tipos de pastillas para realizar una comprobación rápida de todas las clases de pastillas.
3. **Carpeta “ObjectRecognition”**: Es la carpeta que contiene el programa en Python que da respuesta al problema planteado para la práctica. Se explica en el siguiente apartado.

5 ESTRUCTURA DEL PROGRAMA

El programa se ha planteado con orientación a objetos. Se ha procurado hacer un programa lo más versátil posible, de tal manera que reprogramando sólo un componente resulte útil para solucionar otros problemas de reconocimiento de objetos en imágenes.

A continuación se expone cómo se distribuyen los ficheros que componen el programa y qué contiene cada uno de ellos. Todos los ficheros que componen el programa están en la carpeta **“ObjectRecognition”**, que contiene los ficheros y subdirectorios siguientes:

- **Directorio Raíz:** Contiene, únicamente, el fichero **“ObjectRecognition.py”**. Es el programa principal de la aplicación, con el código para presentar un menú con las opciones que ofrece el programa.
- **Subdirectorio “transformations”:** Es un subdirectorio donde he incluido los ficheros relacionados con el tratamiento de la imagen: preproceso, filtrado, segmentación y extracción de características. Contiene los siguientes ficheros:
 - **Pattern.py:** Fichero donde se define la clase **“Pattern”** que representa un patrón que caracteriza una imagen de un objeto concreto (una pastilla, en el caso de esta práctica). Define el método **“extractFromImage()”** que es capaz de extraer una serie de descriptores de una imagen que recibe en forma de array n-dimensional de Numpy (que puede ser, por ejemplo, una subimagen de la imagen original). Actualmente, tiene implementada la extracción de una serie de descriptores bastante amplia, que se mantiene en la entrega a modo demostrativo (posibilita y/o facilita la ejecución de pruebas experimentales). La modificación de esta clase puede adaptar el programa a otro tipo de problemas.
 - **ProcessedContour.py:** Este archivo contiene la definición de la clase **“ProcessedContour”**, que representa un contorno procesado. Hereda de la clase **“Pattern”** (o sea, los objetos de esta clase son también *Pattern*), pero incluye información adicional sobre un contorno que ha sido procesado: su clasificación y el rectángulo que lo contiene. Define métodos para superponer esta información sobre una imagen.
 - **ProcessedImage.py:** Este fichero contiene la definición de la clase **“ProcessedImage”**, que representa una imagen completa procesada. A partir de una imagen (que recibe en forma de array n-dimensional de Numpy) realiza un proceso en el que extrae los contornos de los posibles objetos que existen en ella. Si alguno de contornos encontrados cumple una serie de requisitos, por cada uno de éstos se genera (y mantiene) un objeto de tipo **“ProcessedContour”**. Por tanto, mantiene la información de todos los objetos que tienen algún interés, dentro de la imagen.
- **Subdirectorio “recognition”:** Contiene los ficheros con el código relacionado con operaciones de reconocimiento de objetos. Cuenta con los siguientes ficheros:
 - **Recognizer.py:** Archivo donde se define la clase **“Recognizer”**, que representa un clasificador. Esta clase cuenta con dos métodos básicos: **“train()”** que entrena el clasificador con una serie de imágenes de entrenamiento y **“classify()”** que clasifica un patrón que se le pase como parámetro en una de las clases que tenga definida el clasificador, mediante el algoritmo *KNN*. En la clase se encuentran definidas varias funciones y posibilidades que no están implementadas por falta de tiempo, pero que se mantienen a efectos de disponer de la información para futuras ampliaciones de funcionalidades.
 - **FileRecognizer.py:** Archivo donde se define la clase **“FileRecognizer”**. En realidad, se trata de una clase que sólo implementa métodos de clase (“estáticos”) que realizan las operaciones necesarias para el reconocimiento de imágenes sobre archivos o sobre directorios completos. Tiene implementadas un método que realiza el reconocimiento de pastillas sobre un archivo concreto (**recognizeImage()**) y otra que hace lo mismo sobre todos los archivos de un directorio (**recognizeDirectory()**). Añade una función que realiza un Test sobre todos los archivos de un

subdirectorio: cada subdirectorio se considera que contiene imágenes con pastillas de un tipo concreto. También se ha implementado el método ***recognizeTestDirectory()*** que es capaz de realizar la fase de Test de forma automatizada, si la estructura de directorios de las imágenes de test es igual que la de las imágenes con las que se ha realizado el entrenamiento: se asignan las clases a los subdirectorios de forma automática, y la función es capaz de determinar si una clasificación es correcta o no. Esta función es la que se ha utilizado para las pruebas del programa y para las estadísticas que se han extraído para evaluar su efectividad.

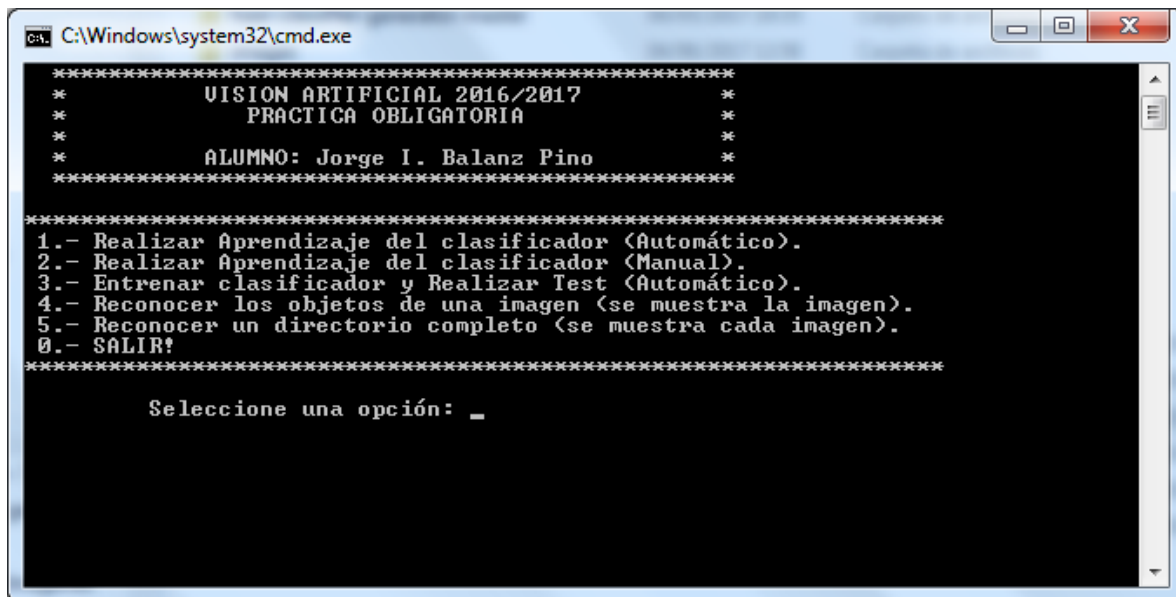
- **Subdirectorio “training”**: este subdirectorio únicamente contiene el fichero **“Training.py”** en el que se define la clase **“Training”**. La clase Training cuenta con los métodos que permiten crear y realizar el entrenamiento de un clasificador (de la clase **“Recognizer”**) a partir de las imágenes de un directorio. El directorio debe estar subdividido en subdirectorios, cada uno de los cuales contendrá imágenes de entrenamiento para un tipo diferente de objetos (pastillas, en nuestro caso). El método **“learning()”** realiza un recorrido por todos los subdirectorios del directorio base que recibe como parámetro. Cada subdirectorio lo considera de una clase de objetos diferente y, según el modo en que se ejecute, asigna un nombre de clase automáticamente o pide un nombre para la clase, presentando como muestra la primera imagen (archivo de imagen) que encuentre en el subdirectorio. Cuando el nombre se asigna automáticamente, los nombres son del tipo:

<nombre-subdirectorio>-[Cod. <código-clase>]

Una vez recorridos todos los subdirectorios, el método retorna un objeto **“Recognizer”** ya entrenado y listo para ser usado en la clasificación de nuevos patrones, así como un listado de los nombres de las clases asignadas y otro listado con los códigos asignados a las clases que se han generado (una por subdirectorio, como ya se ha dicho).

6 INSTRUCCIONES DE USO

Al arrancar el programa se presenta un menú de opciones como el de la imagen siguiente:



```
C:\Windows\system32\cmd.exe
*****
*          VISION ARTIFICIAL 2016/2017          *
*          PRACTICA OBLIGATORIA                  *
*          ALUMNO: Jorge I. Balanz Pino          *
*****

*****
1.- Realizar Aprendizaje del clasificador (Automático).
2.- Realizar Aprendizaje del clasificador (Manual).
3.- Entrenar clasificador y Realizar Test (Automático).
4.- Reconocer los objetos de una imagen (se muestra la imagen).
5.- Reconocer un directorio completo (se muestra cada imagen).
0.- SALIR!
*****

Seleccione una opción: _
```

A continuación se explica lo que hacen las 6 opciones que se ofrecen:

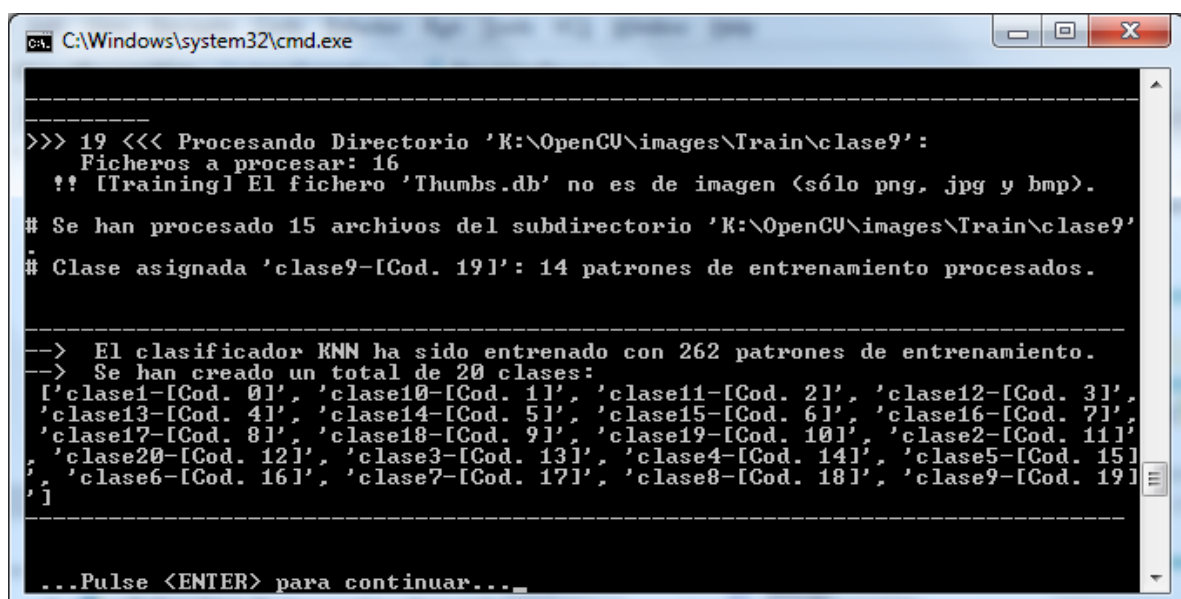
1. Realizar Aprendizaje del Clasificador (Automático):

La opción 1 realiza un aprendizaje automático del clasificador. Solicita un directorio en el que se encontrarán las muestras que se utilicen para el aprendizaje. Cada subdirectorio del directorio que se introduzca, deberá tener imágenes de una sola clase de objeto y serán las que se utilicen para entrenar al clasificador para que pueda reconocer esa clase de objetos. Los nombres y códigos de cada clase se asignan dinámicamente: los códigos son números enteros correlativos y los nombres tendrán el formato: <nombre-subdirectorio>-[Cod. <código-clase>]

También se presenta una breve información sobre las operaciones que se han procesado

(directorios procesados, patrones de entrenamiento obtenidos, nombres de clases asignados...

Después de realizar esta operación, el clasificador está listo para reconocer nuevos objetos en otras imágenes.



```
C:\Windows\system32\cmd.exe

>>> 19 <<< Procesando Directorio 'K:\OpenCV\images\Train\clase9':
      Ficheros a procesar: 16
      !! [Training] El fichero 'Thumbs.db' no es de imagen (sólo png, jpg y bmp).
# Se han procesado 15 archivos del subdirectorio 'K:\OpenCV\images\Train\clase9'
# Clase asignada 'clase9-[Cod. 19]': 14 patrones de entrenamiento procesados.

-----
-> El clasificador KNN ha sido entrenado con 262 patrones de entrenamiento.
-> Se han creado un total de 20 clases:
['clase1-[Cod. 01]', 'clase10-[Cod. 11]', 'clase11-[Cod. 21]', 'clase12-[Cod. 31]',
'clase13-[Cod. 41]', 'clase14-[Cod. 51]', 'clase15-[Cod. 61]', 'clase16-[Cod. 71]',
'clase17-[Cod. 81]', 'clase18-[Cod. 91]', 'clase19-[Cod. 101]', 'clase2-[Cod. 111]',
'clase20-[Cod. 121]', 'clase3-[Cod. 131]', 'clase4-[Cod. 141]', 'clase5-[Cod. 151]',
'clase6-[Cod. 161]', 'clase7-[Cod. 171]', 'clase8-[Cod. 181]', 'clase9-[Cod. 191]']

...Pulse <ENTER> para continuar...
```

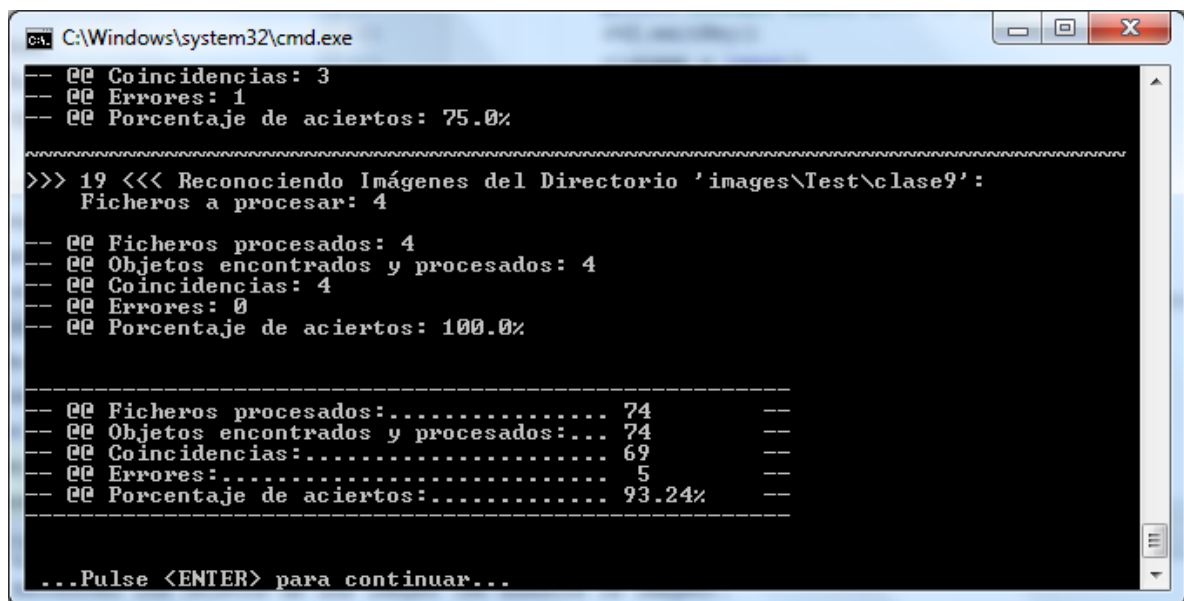
2. Realizar Aprendizaje del Clasificador (Manual):

Esta opción realiza exactamente la misma operación que la opción 1, pero en este caso se pide un nombre de clase para cada clase de objeto (se crea una clase por cada subdirectorio). Para facilitar la asignación de nombres, se presenta la imagen del primer archivo de imagen que se encuentre en cada subdirectorio. También se presenta una breve información sobre lo que se ha procesado (directorios procesados, patrones de entrenamiento obtenidos, nombres de clases asignados...). Después de realizar esta operación, el clasificador está listo para reconocer nuevos objetos en otras imágenes.

3. Entrenar el Clasificador y Realizar Test (Automático):

Esta opción realiza la misma operación que la opción 1 (*Realizar Aprendizaje del Clasificador (Automático)*). Cuando finaliza el aprendizaje, solicita un nuevo directorio donde estarán las imágenes con las que se realizará la fase de test. La estructura de subdirectorios del directorio de test, debe tener la misma estructura que el directorio de entrenamiento: los códigos y nombres de clase se asignan dinámicamente, así que es muy importante que las imágenes de test estén distribuidas en subdirectorios que coincidan con los de la fase de aprendizaje, para que el programa pueda comprobar si pertenecen a la misma clase.

Al finalizar la fase de test se presenta un resumen de los resultados:



```
C:\Windows\system32\cmd.exe
-- @@ Coincidencias: 3
-- @@ Errores: 1
-- @@ Porcentaje de aciertos: 75.0%

>>> 19 <<< Reconociendo Imágenes del Directorio 'images\Test\clase9':
      Ficheros a procesar: 4

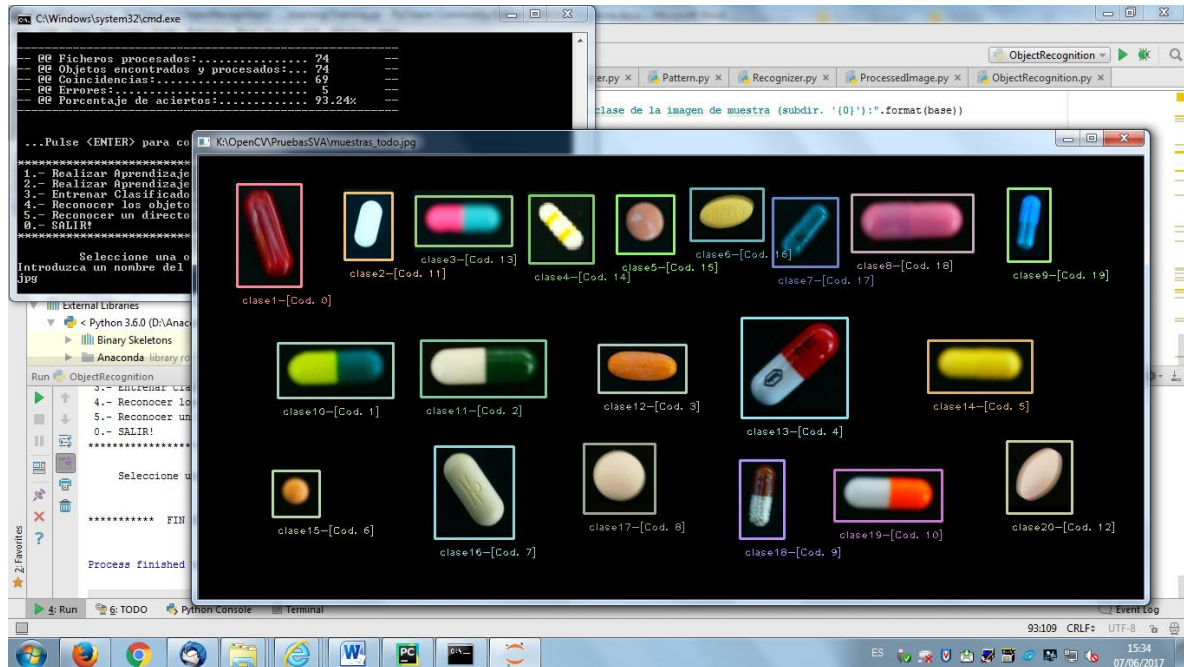
-- @@ Ficheros procesados: 4
-- @@ Objetos encontrados y procesados: 4
-- @@ Coincidencias: 4
-- @@ Errores: 0
-- @@ Porcentaje de aciertos: 100.0%

-- @@ Ficheros procesados:..... 74      --
-- @@ Objetos encontrados y procesados:... 74      --
-- @@ Coincidencias:..... 69      --
-- @@ Errores:..... 5      --
-- @@ Porcentaje de aciertos:..... 93.24%      --

...Pulse <ENTER> para continuar...
```

4. Reconocer los objetos de una imagen (se muestra la imagen):

Una vez entrenado el clasificador, con esta opción se puede ver cómo el programa es capaz de identificar (clasificar) una imagen con uno o más objetos dentro de la misma (los objetos a reconocer son pastillas, en este caso). Una vez clasificados los objetos, el programa presenta una imagen en la que se enmarcan los objetos y se etiquetan con la clase que les corresponda:



5. Reconocer un directorio completo (se muestra cada imagen):

Esta opción hace lo mismo que la opción 4 pero para cada uno de los ficheros de que se encuentren en el directorio que se indique cuando el programa lo solicite. Muestra cada imagen identificando los objetos que contenga. Se ha facilitado el directorio “*PruebasSVA*” en la entrega, para facilitar la comprobación de esta función.

0. SALIR:

Esta opción finaliza el programa.

7 PRUEBAS, AJUSTES Y RESULTADOS

Se han realizado diversas pruebas para ajustar varios de los parámetros hasta alcanzar el máximo porcentaje de acierto en la fase de test, que se ha podido conseguir. La mayoría de las pruebas se han realizado por la técnica del “*ensayo-error*”, intentando ajustar los diferentes parámetros según se iba observando mejoría o no en los resultados. Las pruebas se han realizado utilizando la opción 3 de las que se muestran en el menú del programa, que ofrece el resultado del test de las imágenes que se han facilitado para la práctica. **El mejor resultado que se ha obtenido ha sido un porcentaje de acierto del 93,24%.**

Se han realizado muchas pruebas, pero a continuación, sólo expongo los ajustes que pueden considerarse más destacables:

7.1 Umbral para la Binarización de Imágenes

La selección de un umbral adecuado ha sido algo decisivo a la hora de binarizar la imagen para después aislar diferentes objetos dentro de ella. Como ya se ha comentado anteriormente, las imágenes facilitadas, tanto para la fase de aprendizaje como para la fase de test, presentan pastillas sobre un fondo muy oscuro, casi negro. Existen algunos tipos de pastillas que presentan colores oscuros y sombras que dificultan la extracción de contornos bien definidos si se selecciona un umbral demasiado alto.

Por otro lado, existen imágenes con objetos o sombras que nos son una pastilla. La selección de un umbral demasiado bajo permite a estos objetos “extraños” puedan ser confundidos con pastillas, al proporcionar un contorno tras el proceso de segmentación.

Finalmente se ha optado por seleccionar el número 30 como umbral, que es el que ofrece mejores resultados. Seleccionando umbrales de 25 o de 35 los porcentajes de acierto en el test bajan al 90-91%.

7.2 Área y perímetro

Tras realizar las primeras pruebas se observa que el programa no clasifica bien las pastillas que, teniendo un color similar, presentan una forma diferente. Esto hizo que ajustase mejor la forma en que se calcula el descriptor que se corresponde con la circularidad.

En un principio, se calculaba el área y el perímetro de los objetos (se considera un objeto lo que genera un contorno de cierto tamaño y con cierta separación del borde de la imagen) a partir del rectángulo que los contenía. Esa primera aproximación se hizo a raíz de que los contornos de algunas pastillas no se extraían bien. Para el reajuste de la circularidad se ha estado jugando con las áreas y perímetros tanto del contorno del objeto (obtenidos con `cv2.contourArea()` y `cv2.arcLength`, respectivamente) como del rectángulo que envuelve a dicho contorno (obtenido con `cv2.boundingRect()`). La mejor combinación que se ha conseguido ha sido utilizar el perímetro del contorno y calcular el área con una media ponderada entre el área del rectángulo y el área del contorno, aumentando la tasa de acierto del 87% al 93%. La fórmula finalmente empleada ha sido la siguiente: **$\text{Área} = (\text{área_rectángulo} + 3 * \text{área_contorno}) / 4$**

Se estimó el aplicar un mayor y menor peso a este descriptor para ver si se mejoraba el resultado, pero no se obtuvieron mejores resultados.

7.3 K-vecinos

Se han realizado pruebas variando el número de K-vecinos que utiliza el algoritmo KNN para calcular la clase a la que pertenece una pastilla. Los mejores resultados se han obtenido con K=3 y con K=4 (con K=5 o K=2 la tasa de acierto baja al 90%). Se ha dejado el valor K=3.

7.4 Ecualización

Se han realizado pruebas aplicando ecualización previa de la imagen (o las subimágenes). Por un lado, se ha probado a ecualizar cada uno de los canales de color y, por otro lado, se ha ecualizado el canal 'Y' una vez transformada la imagen al formato "YUV" (luego vuelve a transformarse en BGR).

Ninguna de las dos ecualizaciones ha mejorado los resultados, así que no se aplica ninguna ecualización.

7.5 Histogramas

Se ha intentado agregar un descriptor extraído del histograma de la imagen, para intentar clasificar mejor las imágenes. Por un lado, se ha probado extraer un descriptor de los histogramas de los canales de color BGR y, por otro lado, se ha extraído un descriptor de los histogramas de los canales HSV, una vez transformada la imagen a ese formato.

Ninguno de estos descriptores ha mejorado los resultados.

7.6 Otros descriptores

También se han hecho pruebas con otros descriptores:

Descriptores de color extraídos de las medias de los canales de cada color (3 descriptores). Se han conseguido resultados próximos al 85% de acierto.

Tomar como descriptores los momentos invariantes de Hu, únicamente, ofrece una tasa de acierto por debajo del 50%. En combinación con los descriptores de color se alcanza una tasa del 87%. Si agregamos los descriptores de forma se eleva al 89%. Por tanto no mejora (sino que empeora) la combinación de descriptores de color y forma.

8 MEJORAS

Con respecto al enunciado de la práctica, se podrían considerar mejoras los siguientes puntos:

1. Realizar la fase de entrenamiento del clasificador de forma automatizada, sólo indicando el directorio con las imágenes de entrenamiento;
2. Realizar la fase de entrenamiento y test de forma automatizada, sólo indicando los directorios correspondientes a las imágenes de cada fase;
3. Rotular y etiquetar las pastillas clasificadas en una imagen;
4. Permitir identificar pastillas en una imagen individual;
5. Permitir identificar pastillas en todas las imágenes de un directorio.

9 FUENTES CONSULTADAS:

<http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20110318OscarBoullosa.pdf>
<https://uvadoc.uva.es/bitstream/10324/17054/3/TFG-P-355.pdf>
http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html
http://hpcclab.uccentral.edu.co/wiki/index.php/Descriptores_de_color
<http://www.pyimagesearch.com/2014/03/03/charizard-explains-describe-quantify-image-using-feature-vectors/>
<https://robologs.net/2014/07/02/deteccion-de-colores-con-opencv-y-python/>
<http://shubhamagrawal.com/opencv/opencv-playing-with-brightness-contrast-histogram-blurness/>
<http://cromwell-intl.com/3d/histogram/>
<http://www.pyimagesearch.com/2014/05/26/opencv-python-k-means-color-clustering/>
<http://reyesalfonso.blogspot.com.es/2010/08/implementacion-de-k-means-en-opencv.html>
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_kmeans/py_kmeans_opencv/py_kmeans_opencv.html
<http://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/>
http://docs.opencv.org/3.2.0/d5/d26/tutorial_py_knn_understanding.html
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html
http://docs.opencv.org/3.1.0/db/d7d/classcv_1_1ml_1_1StatModel.html
http://docs.opencv.org/3.1.0/d3/d46/classcv_1_1Algorithm.html
<https://pyformat.info/>
http://www.sromero.org/wiki/programacion/tutoriales/python/recurrer_arbol
<https://docs.python.org/3/library/os.path.html>
https://www.tutorialspoint.com/python/os_walk.htm
<http://python-para-impacientes.blogspot.com.es/2015/06/>