PRÁCTICA DE EVALUACIÓN

Extracción de rasgos

Curso 2024/2025

El objetivo de esta práctica de es extraer diferentes tipos de rasgos en imágenes y usarlos para aprender conceptos sencillos mediante el uso de clasificadores. Para ello se seguirá la misma filosofía del ejercicio realizado en clase, en el que se aprendió el concepto de "peatón" vs. "fondo", pero en este caso aplicado a otro conjunto de entrenamiento.

El estudiante podrá elegir entre distintas opciones, que se esquematizan en la Figura 1. El ejercicio base termina en el paso 6, por lo que deberá completarse al menos hasta ese punto; los pasos 7-9 son retos con los que el estudiante puede "dar un paso más" si quiere mejorar su práctica. Como se observa, el estudiante podrá tomar diferentes caminos, si bien deberá de pasar obligatoriamente por el paso 3 y 6 (clasificación HOG y LBP, respectivamente). La primera elección vendrá determinada por si opta por usar un data set que haya seleccionado el estudiante (paso 1) o usar el predeterminado (paso 2); la segunda elección será si opta por programar el LBP-básico (paso 4) o por usar una implementación ya existente (paso 5). A la hora de la evaluación, los pasos 1 y 4 tiene más peso que los pasos 2 y 5 (se valorará el esfuerzo adicional que tiene, por un lado, la búsqueda de un data set y, por otro, la implementación de un descriptor). En las siguientes secciones se describe con más detalle cada uno de estos pasos.

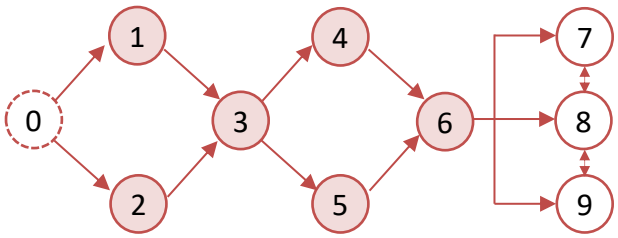

Figura 1. Esquema de opciones a elegir para elaborar las prácticas. (1) Data set seleccionado por el estudiante; (2) Data set MINIST; (3) Clasificación usando HOG; (4) Programación LBP-básico con bloques; (5) Búsqueda de librería con LBP-básico implementado; (6) Clasificación usando LBP básico; (7) Programación y clasificación con LBP Uniforme;(8) Ampliar a más clases; (9) Localización en imágenes.

Data set

Para realizar esta práctica deberá de usarse un conjunto de datos de entrenamiento; para ello, se plantean dos opciones:

1. Buscar un conjunto de entrenamiento de los muchos que se encuentran disponibles en internet (salvo los excluidos en la nota a pie de página[1]); dicho conjunto deberá de incluir, al menos, dos clases.
2. Usar la base de dígitos MNIST, que contiene 60.000 imágenes de entrenamiento y 10.000 imágenes de prueba (disponible en PRADO[2]).

En caso de optar por un conjunto de entrenamiento elegido por el estudiante, incluir en la documentación una breve descripción del mismo, así como la fuente de dónde se ha obtenido.

[1] Los siguientes data sets no pueden usarse para esta práctica: CIFAR-10

[2] A la base de datos MNIST se le ha añadido una clase adicional de "fondos".

Clasificación usando HOG

Usando la base de datos elegida, entrenaremos un clasificador SVM para predecir entre dos clases; las clases a predecir dependerán del conjunto de datos elegido:

- En caso de que el data set haya sido elegido por el estudiante, establecer las dos clases que se quieren entrenar.
- En el caso de que se haya optado por el data set MNIST, las clases a predecir corresponderán a los dos últimos números de tu DNI (en caso de coincidir, utilizar el siguiente distinto). Por ejemplo, si tu DNI es 22.333.123P, el sistema deberá de aprender a clasificar los números 3 y 2.

3

En una primera aproximación, se propone entrenar un clasificador SVM[3] con HoG[4] como descriptor. Para ello, se proponen una serie de mejoras respecto a lo que se hizo en el ejercicio de clase[5]: concretamente, (1) deberá de medirse la bondad del clasificador considerando un conjunto de medidas estándar. Además, se plantea que este análisis (2) considere diferentes conjuntos de entrenamiento y prueba, generados siguiendo las técnicas estándar que el estudiante considere (validación cruzada, etc.). Por último, en el estudio (3) se variarán los parámetros del SVM con el objetivo de encontrar el mejor clasificador[6].

Como resultado de este primer ejercicio, en la documentación se mostrarán las tablas con los diferentes datos numéricos obtenidos en este estudio. A partir de esos datos, se concluirá una valoración final de la combinación HoG+SVM. Además, deberá usarse el clasificador para predecir la clase de una serie de imágenes de entrada dadas; en la documentación, incluir ejemplos de cómo ha clasificado el sistema diferentes imágenes y una valoración motivada de los mismos.

Clasificación incorporando LBP-básico

En este ejercicio se propone usar el descriptor LBP y comparar los resultados con los obtenidos en el caso anterior (HoG). Para ello, existen dos opciones

- Programar el descriptor **LBP en su versión básica**. Por versión básica se entiende la descrita en teoría: cálculo de códigos LBP considerando sus 8-vecinos más cálculo por bloques de histogramas (recordemos que el descriptor corresponderá a la concatenación de los histogramas individuales de cada bloque)[7]. La implementación del descriptor deberá de hacerse **en una clase específica "LBPDescriptor"**, siguiendo la filosofía de OpenCV para el caso del descriptor HoG (esto es, constructor y método 'compute' para su cálculo)[8]. Dicha clase deberá incluirse

4

[3] Los parámetros del HOG deberán de adaptarse en función de las características del data set (esto es, del tamaño de sus imágenes). Por ejemplo, las imágenes de MNIST son de un tamaño 28x28, por lo que habrá que adaptar los parámetros de HoG a ese tamaño (no pueden usarse los mismos que en el ejemplo de clase, donde las imágenes eran de 128x64); en ese caso, unos posibles valores serían: tamaño de ventana 28x28, tamaño de celda 4x4, tamaño de bloque 8x8, desplazamiento 2x2 y número de bins 9. En el caso de optar por un data set propio, habrá que hacer el ajuste correspondiente.

[4] Como se indicó en clase, la biblioteca *OpenCV* utilizada para las prácticas incluye el descriptor HoG y se encuentra disponible en C++, Java y Python. Para más información, visitar la web de OpenCV. No obstante, el estudiante podrá usar la biblioteca que quiera (no tiene por qué ser la *OpenCV*).

[5] Recordemos que, en el ejercicio de clase, si bien se ha entrenado un clasificador SVM y se ha usado para predecir la clase de una imagen dada, (1) no se han obtenido medidas de la bondad de la clasificación usando el conjunto de imágenes test, (2) ni tampoco se han barajado diferentes particiones entre datos de entrenamiento y prueba. Además, (3) se usaron los parámetros básicos en el SVM (con un kernel lineal), sin probar otros valores que pudieran mejorar el clasificador.

[6] La *OpenCV* permite establecer diferentes *kernels* (polinomial, sigmoide, etc.), así como otros parámetros. Más información aquí.

[7] Al igual que ocurría con HoG, es necesario elegir los parámetros adecuados (tamaño de celda, desplazamiento, etc.) teniendo en cuenta el tamaño de las imágenes del data set.

[8] Siguiendo la filosofía *OpenCV*, se ha de crear una clase propia de la forma (ejemplos Java y Python):

en un único fichero LBPDescriptor.[java|py|cpp]. Además de los comentarios incluidos en el código, la documentación deberá incluir una descripción detallada de la implementación, justificando el enfoque y decisiones tomadas.

- Usar una implementación ya existente. En este caso, el estudiante podrá usar bibliotecas existentes que tengan incorporado el LBP. En este caso, en la documentación deberá de **indicarse la biblioteca usada y de dónde se ha descargado**. 5

Una vez incorporado el LBP, se hará una evaluación de la combinación LBP+SVM en la misma línea que en el ejercicio anterior para el caso HoG+SVM. Así, se mostrarán las tablas con los diferentes datos numéricos obtenidos en este estudio y se concluirá una valoración final de la combinación LBP+SVM comparándola con la HoG+SVM. 6

De cara a la evaluación, tendrá más peso la elección de la primera opción, esto es, la implementación del descriptor LBP por parte del estudiante. En caso de optar por usar una biblioteca externa, se podrá valorar su uso para entrenar un clasificador (pero no la capacidad de implementar un descriptor). En caso de optar por la implementación propia, se aconseja programar el descriptor desde cero, sin partir de ninguna implementación previa (de lo contrario, el sistema anti-plagio marcará la práctica como copia).

Un paso más…

Continuando con los ejercicios anteriores, se proponen las siguientes mejoras:

- **LBP-uniforme**

Incorporar el descriptor LBP uniforme y comparar los resultados con los obtenidos en los casos anteriores (HoG y LBP básico). Al igual que para el caso básico, se puede implementar el descriptor o usar uno ya existente. 7

En caso de que se opte por la implementación[9] del descriptor, deberá realizarse en una clase específica “LBPUDescriptor” (que incluya método ‘compute’ para su cálculo) incluida en un único fichero LBPUDescriptor.[java|py|cpp]. Además de los comentarios incluidos en el código, la documentación deberá incluir una explicación motivada de la implementación. A la hora de evaluar, tendrá mayor puntuación el hecho de haber implementado el descriptor.

- **Incluir más clases**

En los ejercicios anteriores se han considerado dos clases, por lo que se propone como mejora ampliarlo a tres. En el caso de que se haya optado por el data set de MNIST, incluir un dígito más: concretamente el siguiente en el DNI (siguiendo la secuencia desde atrás). En caso de que el data set haya sido otro elegido por el estudiante, incluir la tercera clase que se considere (dependiendo del conjunto). Al igual que en los ejercicios anteriores, incluir en la documentación tablas con las medidas de bondad y un análisis y conclusiones sobre las mismas. 8

```
public class LBPDescriptor {
   // Declaración de las variables de instancia
  public LBPDescriptor( parámetros ){
    //Inicialización de variables de instancia
  }
  public  MatOfFloat compute(Mat img){
    //Calcula LBP sobre 'img' y devuelve el descriptor
  }
}

class LBPDescriptor():
    def __init__(self, parámetros ):
     # Declaración e inicialización de variables de instancia
    def compute(self, img):
     # Calcula LBP sobre 'img' y devuelve el descriptor
     return descriptors
```

y usarla de la misma forma que se usaba `HOGDescriptor` en el ejercicio de clase.

[9] Como vimos en clase, la solución más sencilla es un mapeo de códigos básicos a códigos uniformes (mapa aquí).

El reto final…

9

Partiendo de los modelos aprendidos en los bloques anteriores, se proponen como nuevo reto la localización de objetos en imágenes. En los ejemplos anteriores hemos trabajado con imágenes de un tamaño dado (p.e. 28x28 en MNIST) donde había un solo objeto por imagen (p.e., un dígito en MNIST), si bien no se ha aplicado al caso de imágenes de mayor tamaño con objetos en diferentes localizaciones y diferentes escalas. En este ejercicio se propone usar los clasificadores aprendidos anteriormente para localizar objetos en una imagen dada, sin importar el tamaño de la imagen, la posición de los dígitos o su escala[10]. Estos objetos dependerán del data set utilizado en el entrenamiento (p.e., dígitos si se ha usado MNIST). Los objetos localizados se marcarán con un rectángulo en la imagen original.

En caso de que se haya usado el data set de MNIST, al margen de otras posibles imágenes de test, crear una propia de la siguiente forma: en un folio escribir a mano un conjunto de dígitos aleatorio; además, incluir el DNI del estudiante. Fotografiar esa imagen y utilizarla para testear el algoritmo de detección realizado en este apartado[11].

Al igual que en ejercicios anteriores, incluir en la documentación los resultados obtenidos, así como un análisis y justificación de los mismos.

Entrega y documentación

La entrega se realizará a través de PRADO (Tema 2→Entrega ejercicio de evaluación). Se deberá entregar un fichero comprimido (.zip o .rar) que incluya:

- Fichero(s) que permitan la ejecución de los ejercicios (.jar en caso de que las prácticas se hayan realizado en java o el fichero ejecutable compilado si ha sido en C++). Deberá estar en la raíz del fichero comprimido junto con las librerías[12].
- Código fuente. Deberá incluirse la carpeta completa asociada al proyecto, de forma que éste pueda abrirse con el entorno IDE correspondiente (indicar con qué entorno se ha desarrollado la práctica); en particular, han de ser claramente identificables los ficheros fuentes asociados a la implementación de las clases de los descriptores. En caso de realizarla en Python y usar Jupyter Notebook, incluir también ficheros .py con el código.
- Documentación en PDF (localizable en la raíz del fichero comprimido). La documentación deberá de incluir los resultados obtenidos en cada uno de los ejercicios, así como conclusiones parciales y finales. **En esta documentación se basará la evaluación** y, por tanto, la nota de este bloque; por este motivo, se recomienda extenderse y reflejar todo el trabajo realizado. **NO SE ADMITE COMO DOCUMENTACIÓN LA EXPORTACIÓN A PDF DE UN JUPYTER NOTEBOOK**, es obligatorio hacer una documentación específica en la línea indicada anteriormente; de no ser así, la práctica se considera no superada

  Al comienzo de la documentación, deberá de indicarse el "camino" elegido siguiendo el esquema de la Fig. 1; por ejemplo: (1,3,4,6), o (2,3,4,6), o (1,3,4,6,7), o (1,3,4,6,7,8,9) …

En caso de defensa, el estudiante será convocado, si procede, por correo electrónico.

[10] Recordemos que, como vimos en clase, trabajaremos con ventanas del tamaño de las usadas en el entrenamiento, que iremos desplazando a lo largo de la imagen original. Además, iremos escalando la imagen para trabajar a diferentes escalas.

[11] Convertir la imagen a niveles de gris para garantizar compatibilidad con OpenCV.

[12] En caso de que se use NetBeans y el proyecto que se vio en clase, el ejecutable se generará en la carpeta */dist* del proyecto, dentro de la cual estarán las bibliotecas en la carpeta */lib*.