# Detector de peatones: código de ejemplo

El código de ejemplo que os proporcionamos implementa las tres fases de un detector de objetos explicamos a lo largo del curso: **generación de candidatos, clasificación de candidatos** y **refinación de la decisión**. Se trata de una implementación básica que sólo incluye las partes imprescindibles de un detector y que no se ha optimizado con el objetivo de mantener el código simple y fácil de entender. Además el código incorpora también un módulo para poder realizar la **evaluación del resultado** de la detección sobre un conjunto de imágenes, mostrando las gráficas con la evaluación del rendimiento del detector. Tamién os proporcionamos **3 bases de datos de imágenes de peatones**, que podréis utilizar para entrenar y evaluar el detector.

Para cada una de las fases de un detector, los métodos que se han implementado son los siguientes:

- Para **la generación de candidatos** se utiliza una pirámide de imágenes a diferentes escalas, combinada con el método de *ventana deslizante*.
- Para la **clasificación de candidatos** el código incorpora dos descriptores alternativos (**LBP**, explicado en la semana 2 del curso, y **HOG**, explicado en la semana 4) y dos métodos de clasificación (**regresión logística**, explicado en la semana 2, y **SVM**, explicado en la semana 4). El código se puede utilizar con cualquier combinación de descriptor y clasificador, incluso combinando LBP y HOG como descriptores sobre la misma imagen.

  Hemos incluido también los **modelos** de los **dos clasificadores** (regresión logística y SVM) entrenados con los **dos descriptores** (LBP y HOG) en una de las bases de datos de imágenes que os proporcionamos, *Pedestrians*. Estos modelos ya entrenados se pueden utilizar directamente sobre las imágenes de test para comprobar el resultado del detector. Sin embargo, el código también permite entrenar nuevos modelos con diferentes configuraciones del descriptor y del clasificador sobre otras bases de datos de imágenes. Para ello podéis utilizar las bases de datos que os proporciornamos nosotros, pero también podéis utilizar otras imágenes de peatones o de otros objetos.
- Para la refinación de decisión, se ha implementado un método de ***Non-Maximal Suppression***.

## 1. Instalación y configuración

Para poder utilizar el código, seguid los siguientes pasos:

1. Descargar y descomprimir el fichero `Pedestrian_Detector.zip` que podéis encontrar en la página del curso, en la sección dedicada al código del detector. Este fichero contiene el código en Python, los modelos pre-entrenados y una imagen de test.
2. Consultar el fichero `README.txt` incluido junto con el código. En caso necesario, seguid los pasos que se indican para instalar en vuestro ordenador Python, las librerías adicionales que se utilizan en el código y un IDE para trabajar con Python.
3. Descargar y descomprimir el fichero `Datasets.zip` que podéis encontrar en la página del curso, en la sección dedicada al código del detector, y que contiene las bases de datos de imágenes.
4. Abrir el entorno de Python y cargar el fichero `test_image.py`. Si lo ejecutáis se aplicará el detector sobre una imagen ejemplo de test utilizando la configuración que se especifica en el fichero `config.py`. Por defecto, se utilizará LBP y regresión logística. En las secciones siguientes del documento os explicamos cómo cambiar la configuración del detector, cómo re-entrenar los modelos de clasificación con otras variantes de descriptor y método de clasificación y cómo aplicarlo y evaluarlo con las distintas bases de datos que os proporcionamos.

## 2. Estructura del código de ejemplo

El código está estructurado en ocho ficheros:

- **`config.py:`** contiene la definición de todos los parámetros de configuración del detector en todas sus fases (tipo y parámetros del descriptor y clasificador, parámetros de la ventana deslizante, directorios de imágenes, etc.). En la sección 3 de este documento especificamos cuáles son estos parámetros y sus posibles valores.
- **`extract_features.py:`** sirve para **extraer características** sobre un conjunto de imágenes y guardarlas en un directorio determinado para poderlas utilizar luego para entrenar el detector.

  Al ejecutar este módulo se calcula un determinado descriptor (LBP o HOG) para un conjunto de imágenes de entrenamiento. El descriptor y las imágenes a utilizar se especifican en el fichero `config.py`. El resultado se guarda en una subcarpeta `Features`, para poderlo utilizar en el paso siguiente al entrenar el modelo de clasificador.

- **`train_model.py:`** este módulo sirve para **entrenar un modelo de clasificador** con un conjunto de características ya pre-calculadas.

  Tanto el tipo y configuración del clasificador (regresión logística o SVM) como el tipo de descriptor utilizado (HOG o LBP) se especifican en el fichero `config.py`. Las características deben estar pre-calculadas en la subcarpeta `Features`, como resultado de ejecutar el módulo `extract_features.py`. El modelo del clasificador aprendido se

guarda en una subcarpeta `Models`, para ser utilizado al aplicar el detector en una imagen de test.

- **`test_image.py:`** este módulo sirve para **aplicar el detector sobre una imagen** completa, aplicando la generación de candidatos con pirámide y *non-maximal suppression*.

  Tanto los parámetros del detector, como el tipo de descriptor y el modelo de clasificador que se utilizan se especifican en el fichero `config.py`. El modelo del clasificador debe estar pre-calculado en la subcarpeta `Models`, como resultado de ejecutar el módulo `train_model.py`. El resultado de la detección se muestra en una nueva imagen, mostrando las ventanas de detección con la confidencia de cada una de ellas.

- **`test_folder.py:`** este módulo sirve para **aplicar el detector sobre un conjunto de imágenes.**

  El conjunto de imágenes deberá estar en una subcarpeta que se indica en el fichero `config.py`. El modelo del clasificador debe estar pre-calculado en la subcarpeta `Models`, como resultado de ejecutar el módulo `train_model.py`. El resultado de la detección se guarda en la subcarpeta `Results`.

- **`show_results.py:`** este módulo sirve para **mostrar el resultado de la detección** sobre un conjunto de imágenes.

  Al ejecutarlo, toma todos los resultados guardados en la subcarpeta `Results` y genera, para cada uno de las imágenes de test, una nueva imagen, superponiendo a la imagen original los rectángulos con el resultado de la detección.

- **`evaluate_results.py:`** este módulo sirve para **evaluar los resultados del detector**.

  A partir de los resultados guardados en la subcarpeta `Results`, genera la gráfica con la evaluación del rendimiento que muestra la curva *tasa de error (miss rate) vs. FFPI*, tal como se ha explicado en los vídeos.

- **`main.py:`** este módulo permite ejecutar los pasos anteriores (extracción de características, aprendizaje del clasificador, aplicación del detector a una imagen y evaluación de los resultados) de forma conjunta y secuencial. Comentando o descomentando las llamadas a cada uno de los módulos anteriores podremos ejecutar diferentes secuencias de ejecución, según lo que queramos probar. Utilizaremos este módulo cuando queramos ejecutar varios de estos pasos de forma secuencial.

Podéis analizar el código de estos módulos para ver cómo se han implementado cada uno de los pasos del detector. La mayoría de pasos (extracción de características, aprendizaje del clasificador, generación de la pirámide) se han implementado a partir de llamadas a las funciones de las librerías `scikit-image` y `scikit-learn`.

## 3. Fichero de configuración `config.py`

En esta sección vamos a dar detalles de algunos de los principales parámetros del detector que podemos fijar en el fichero de configuración `config.py`. Los hemos agrupado según el módulo para el que son necesarios, que puede no coincidir exactamente con el orden en el que están dentro del fichero `config.py`.

**Parámetros para la extracción de características**

- **`featuresToExtract:`** determina qué descriptor se utiliza. Puede tomar el valor de uno de los dos descriptores, `['HOG']` o `['LBP']`, o especificar que se calculen los dos descriptores a la vez `['HOG', 'LBP']`, concatenándolos en una única descripción final.
- **`LBP Parameters:`** parámetros utilizados para el cálculo de LBP. Los más relevantes son `lbp_win_shape`, que determina el tamaño de los bloques y `lbp_win_step`, que determina la separación entre bloques.
- **`HOG Parameters:`** parámetros utilizados en el cálculo del descriptor HOG: nº de intervalos del histograma de orientaciones, nº de píxeles en cada celda y nº de celdas por bloque.

**Parámetros para el aprendizaje del modelo del descriptor**

- **`datasetRoot:`** ruta del directorio donde se encuentra la base de datos con las imágenes que se utilizarán en el aprendizaje de los modelos.
- **`positive_folder, negative_folder:`** subcarpetas del directorio de la base de datos donde se encuentran las imágenes positivas y negativas para el aprendizaje.
- **`positiveFeaturesPath, negativeFeaturesPath:`** directorio donde se guardan las características de las imágenes positivas y negativas utilizadas para entrenar el modelo.
- **`model:`** método de clasificación que se utiliza. Puede tomar los valores `'SVM'` or `'LogisticRegression'`
- **`modelPath:`** directorio donde se guarda el modelo de clasificador que se ha aprendido.
- **`SVM.LinearSVC parameters:`** parámetros utilizados para entrenar un clasificador SVM. El más relevante será `svm_C`, que indica el factor de regularización.
- **`LogisticRegression parameters:`** parámetros utilizados para entrenar un clasificador con regresión logística. El más relevante será `logReg_C` que permite especificar el factor de regularización.

**Parámetros para la ejecución del detector**

- **`window_shape:`** tamaño de la ventana de detección al aplicar la ventana deslizante.
- **`window_step:`** desplazamiento de la ventana deslizante entre paso y paso.
- **`decision_threshold:`** umbral que se fija para determinar si la evaluación de una ventana se debe dar como detección positiva. Se puede fijar un umbral diferente para cada combinación de descriptor y clasificador.
- **`downScaleFactor:`** factor de escalado al generar la pirámide de imágenes.
- **`nmsOverlapThresh:`** factor de solapamiento para considerar equivalentes dos ventanas al aplicar el *non-maximal suppresion*.
- **`testFolderPath:`** directorio que contiene las imágenes que se van a utilizar como test.
- **`resultsFolder:`** subcarpeta donde se guardan los resultados de las detecciones y las imágenes que se generan con las ventanas detectadas.
- **`annotationsFolderPath:`** directorio que contiene la anotación con la localización correctar de las imágenes de test, para poder realizar la evaluación.

## 4. Datasets

Os proporcionamos tres bases de datos con imágenes de peatones que podéis utilizar para entrenar y ejecutar el detector:

- **Pedestrians:** base de datos completa con imágenes de peatones preparadas para poder realizar el aprendizaje de los modelos. Contiene imágenes positivas y negativas (extraídas del conjunto de entrenamiento de la base de datos de INRIA) ya recortadas todas al mismo tamaño para poder extraer los descriptores y entrenar los modelos. La podéis utilizar para hacer un aprendizaje completo del detector.
- **Pedestrians_dummy:** es un subconjunto de la base de datos anterior, con pocas imágenes positivas y negativas de peatones. Se puede utilizar para hacer un aprendizaje rápido del detector y comprobar cómo pueden influir los diferentes parámetros. Evidentemente, el rendimiento del detector entrenado con esta base de datos no será óptimo.
- **INRIA:** base de datos con imágenes de test. Contiene imágenes completas (extraídas del conjunto de test de INRIA), algunas de las cuales incluyen peatones mientras que otras no. Son las imágenes que podéis utilizar para comprobar el funcionamiento del detector.