Programación para la ciencia de datos - PEC4

En este Notebook encontraréis el ejercicio que supone la cuarta y última actividad de evaluación continuada (PEC) de la asignatura. Esta PEC intenta presetaros un pequeño proyecto en el cual debéis resolver diferentes ejercicios, que engloba muchos de los conceptos cubiertos durante la asignatura.

El objetivo de este ejercicio será desarrollar un **paquete de Python** fuera del entorno de Notebooks, que nos permita resolver el problema dado. Trabajaréis en archivos Python planos `.py`. Este tendrá que incluir el correspondiente código organizado lógicamente (separado por módulos, organizados por funcionalidad,...), la documentación del código (*docstrings*) y tests. Además, tendréis que incluir los correspondientes archivos de documentación de alto nivel (`README`) así como los archivos de licencia y dependencias (`requirements.txt`) comentados en la teoría.

Hacer un `setup.py` es opcional, pero si se hace se valorará positivamente de cara a la nota de la práctica y del curso.

Se nos pide que implementemos un paquete (o módulo) de Python que sea capaz de realizar un análisis de imágenes de diferentes ciudades europeas tomadas entre 2015 y 2019. Por un lado tendremos las imágenes y por otro los objetos presentes y su posición dentro de la imagen.

Enunciado:

Nos han encargado analizar imágenes de calles de distintas ciudades europeas para un proyecto relacionado con *smart-cities*. Para empezar a trabajar, tenemos un dataset de imágenes de tres ciudades que han estado tomadas desde dentro de un coche circulando por distintos puntos de cada ciudad. El dataset completo lo podéis encontrar aquí (https://www.cityscapes-dataset.com). Junto con las imágenes nos han dado también unos ficheros de texto donde podemos encontrar los tipos de objeto que hay en cada una de ellas y sus posiciones. Para **cada imagen tenemos un archivo de texto** con uno o más objetos. Nos indican que esta información se ha extraído mediante la utilizacion de YOLOv5 (https://docs.ultralytics.com). YOLO (*You Only Look Once*) es un algoritmo basado en redes convolucionales muy potente para la detección de objetos en imágenes o vídeos en tiempo real.

En esta PEC tendréis que trabajar con estos ficheros para analizar las imágenes y extraer conclusiones sin tener que mirar cada una de las imágenes. Los datos los tenéis en el fichero **dataset.tar.gz**. Allí encontraréis las siguientes carpetas:

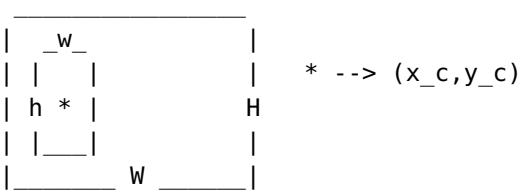
- **images**: Carpeta que contiene todas las imágenes. Fijaos que en el nombre del fichero sale la ciudad y la fecha en que fue tomada la fotografía.
- **labels**: En esta carpeta encontraréis los archivos .txt con el mismo nombre base de la imagen a la que corresponde. En cada archivo habrá tantas líneas como objetos encontrados en la imagen. Para cada objeto hay 6 columnas con la siguiente información:

* **identificador del objeto**: entre 0 y 80, que son la cantidad de objetos que puede detectar YOLO (os damos relación entre identificador y tipo de objeto en otro archivo explicado más abajo).

* **coordenadas del objeto x^n_c,y^n_c, w^n, h^n**: La posición del objeto detectado se define con las coordenadads de la *bounding box*, que el rectangulo que contiene el objeto. Este viene definido por 4 coordenadas: el valor central *(x^n_c,y^n_c)* y la anchura y la altura del rectángulo *(w^n, h^n)*. YOLO da estos valores normalizados, por lo que serán entre 0 y 1, las coordenadas horizontales y la anchura van divididos por la anchura total de la imagen mientras que las coordenadas verticales y la altura del rectángulo van divididos por la altura total de la imagen:

$$x_c^n = \frac{x_c}{W}, y_c^n = \frac{y_c}{H}$$
$$w^n = \frac{w}{W}, h^n = \frac{h}{H}$$

donde x_c, y_c es el valor central de la imagen en píxeles, w y h son la anchura y la altura del rectángulo y W y H son la anchura y altura de la imagen. Os lo mostramos en el siguiente esquema:



- **confian<a de la detecció**: En la última columna tenemos la probabilidad que da el modelo de YOLO de que la posición del objeto detectado sea correcto.

```
9 0.760986 0.140137 0.0229492 0.104492 0.285246
58 0.960693 0.693359 0.0786133 0.210938 0.293333
9 0.928955 0.0634766 0.0405273 0.0996094 0.332471
9 0.908691 0.059082 0.0791016 0.114258 0.374223
9 0.801514 0.254395 0.0336914 0.135742 0.390878
9 0.887451 0.0537109 0.0395508 0.107422 0.554214
9 0.243896 0.267578 0.0209961 0.109375 0.591291
2 0.438232 0.438965 0.0541992 0.0478516 0.740896
2 0.753662 0.459473 0.0825195 0.100586 0.745214
2 0.530273 0.453613 0.0576172 0.0771484 0.814936
2 0.384766 0.450195 0.0634766 0.107422 0.829835
...
```

Por ejemplo en la primera línea del archivo que os mostramos arriba tenemos un objeto con identificador igual a 9, con coordenadas normalizadas: x_c=0.760986, y_c=0.140137, w=0.0229492 h=0.104492 y la probabilidad de que esté bien detectado igual a 0.285246.

- **class_name.txt**: En este fichero encontraréis la relación entre el identificador del objeto y el nombre. Ej:

```
0 person
1 bicycle
2
```

Presentación de los resultados: Para hacer la entrega más fácil y homogénea os pedimos que organicéis el código de tal manera que **desde el fichero principal retorne todas las respuestas que se os pida en la PEC** haciendo uso de funciones que tendréis que definir en módulos. Para eso, en cada ejercicio, os indicaremos el formato que tiene que tener cada respuesta. De tal manera que ejecutando `python fichero_principal.py` se vaya respondiendo a toda la PEC. Si valoráis que es mejor hacerlo de otra manera tendréis que documentarlo muy bien en el README para que se pueda ejecutar sin problema. Os recordamos que en el README también tenéis que indicar como ejecutar los test y comprobar la cobertura

de éstos

Control y revisión del dataset:

Cuando empezamos a trabajar en un proyecto de análisis de datos, una buena práctica es asegurarnos de que los datos son correctos. En otras palabras, es necesario hacer un análisis exploratorio inicial para detectar errores o casos especiales y tomar decisiones sobre como abordarlos. Aquí os proponemos hacer:

Ejercicio 1.

Leed todos los ficheros tanto de imágenes como de texto y juntadlos en un dataframe con las columnas que creáis interesantes para reolver la PEC.

Ejercicio 1.2.

Es este caso tenemos pocas imágenes, pero en un caso realista (ej, cámaras control de tráfico) se podrían en tener muchísimos frames que analizar. En caso de tener millones de archivos o archivos muy pesados como lo haríais? (**No hace falta implementar la solución**, solo justificarla).

Mostrad por pantalla las primeras filas del dataframe y contestad a la pregunta 1.2 con un `print`.

Ejercicio 2.

A veces, nos encontramos con datos corrompidos, tanto por errores humanos como por algún *bug* en el código. Para detectar si hay algun fichero no válido, cread una función `check_yolo` que tome como input un fichero de texto y devuelva un `boolean` dependiendo de si tiene formato YOLO o no. Para esto comprobad que el número de las columnas y las caracterísitcas de cadauna de ells para saber si cumple el formato descrito en el enunciado.

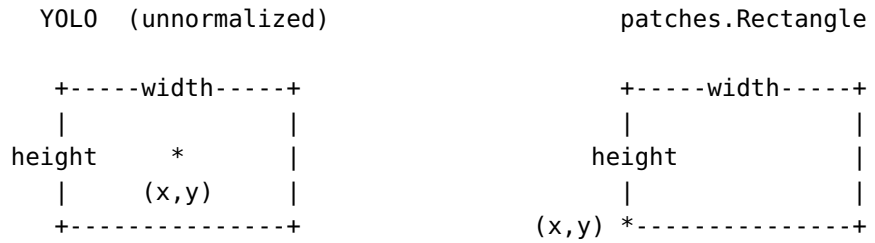
Si detectamos ficheros con alguna línea o valor incompatible los eliminearemos del dataset y trabajaremos sin ellos el resto de la práctica.

Llamad la función desde el códgio principal y **Escribid por pantalla** el nombre de los ficheros que no siguen el formato YOLO o si fuera el caso mostrar un mensaje de que no se ha encontrado ningu archivo incompatible.

Ejercicio 3.

También es importante comprobar que las predicciones son correctas, por ejemplo que la información del archivo corresponde con los objetos de la imagen. La manera más sencilla es visualizándolo. Para esto dibujaremos las *bounding boxes* de los objetos encima de la imagen. Podéis usar funciones de la librería `matplotlib.pyplot` tanto para cargar las imágenes (`imread`), para generar el rectángulo (`patches.Rectangle`) y visualizar el resultado (`imshow i add_patch`).

Para hacer este ejercico tendréis que hacer un canvio de coordenadas, ya que la manera de definir el rectángulo en YOLO y en `patches.Rectanlge` es diferente:



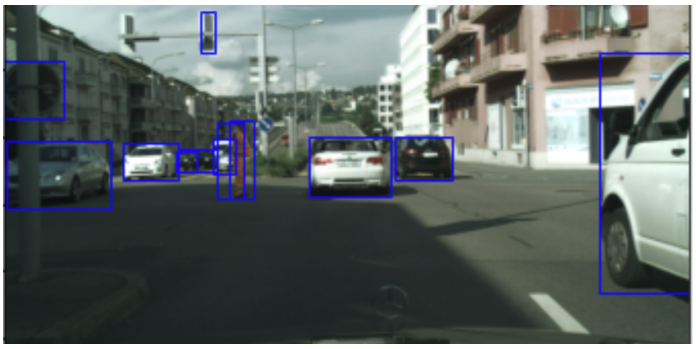
Recordad "desnormalizar" las coordenadas YOLO para que estén en número de píxeles (valores enteros). Para esto necesitarés la medida original de las imagenes que es W = 2048 y H = 1024.

Para visualizar las imágenes podéis seguir este ejemplo (https://www.adamsmith.haus/python/answers/how-to-draw-a-rectangle-on-an-image-in-python)

Comprueba que las *bounding boxes* estan englobando los objetos con la primera foto de cada ciudad

(teniendo en cuenta el orden lexicográfico).

Durante la ejecución del código principal presentad la **visualització de estas tres imagenes con los contornos de los objetos detectados**. Os dejamos aquí una imagen de ejemplo:



Análisis de datos:

Ejercicio 4.

En esta parte trabajaremos con objetos con una **confiança mayor a 0.4** y solo con los ficheros que haya pasado el test de **candidatos a YOLO**.

Ejercicio 4.1

Econtrar y representar gráficamente la distribución de objetos en todo el dataset. Es decir, queremos saber para cada objeto cuantos hay detectos en total. **Mostrad por pantalla** los identificadores y los nombres de los 5 objetos más comunes y cuantas veces aparecen. **Mostrad un gráfico tipo barras** con el número total de objetos de cada clase.

Ejercicio 4.2.

En el apartado anterior hemnos encontrado los 5 objetos más populares del dataset. De estos 5 objetos queremos saber si siguen un patrón en su aparición por las imagenes. Queremos saber si hay muchos por imagen o si por ejemplo se conectran en unas pocas imagenes. Para eso mirarmos la distribución del número de aparciones por imagen con un histograma.

Mostrad en un solo gráfico las distribuciones de estos 5 objetos de manera que sean comparables (usar histogramas normalizados).

Ejercicio 4.3.

Cuál es el número medio de objetos (sin importar tipo) por imagen? **Mostrad por pantalla** el resultado explicado y formatado.

Ejercicio 4.4.

Queremos saber cuáles son los tres elementos más populares por imagen. Para definir los elementos más populares por imagen os pedimos los siguientes pasos:

a) Crear una función que dado un dataframe os devuelva un diccionario ordenado según popularidad del objeto. Las claves del diccionario serán el nombre o identificador del objeto y como valor las veces que ha estado entre los 3 objetos más populares de una imagen.

b) **Mostrar per pantalla** los tres elementos objetos que han sido más veces populares en las imagenes.

c) Coinciden los elementos más populares por imagen con los del dataset encontrados en el 4.1? Si es que no: Explicad porque puede pasar. Si es que sí: Dad un ejemplo donde podría no ser así. **Responder en un print** por pantalla.

Nota: Reglas para escoger los objetos más populares de una imagen:

1. Si tiene **menos de 3 objetos o exactament 3 objetos** diferentes cogeremos todos aquellos objetos que aparezcan.
2. Si tiene **más de 3 objetos diferentes**:

2.1) Si la frecuencia más alta aparece en más de tres objetos, cogeremos todos esos objetos como los más populares (en este caso pueden ser más de tres).

Ejemplo (4 gatos, 4 perros, 4 patos, 4 ratas, 1 paloma) -> objetos más populares: (gato, perro, pato, rata)

2.2) Si no hay empate entre la popularidad entre el 3o y 4o objeto más popular cogemos los 3 objetos más frecuentes.

Ejemplo (5 gatos, 4 perros, 4 patos, 2 ratas, 1 paloma) -> objetos más populares: (gato, perro, pato)

2.3) Si el empate de popularidad se produce entre el 3er y el 4o objeto más frecuentes haremos lo siguiente:

2.3.1) Si el empate se produce entre el tercer y el cuarto objeto más frecuentes cogeremos solo los dos objetos más populares.

Ejemplo (4 gatos, 4 perros, 2 patos, 2 ratas, 1 paloma) -> objetos más populares: (gato, perro)

2.3.2) Si el empate de frecuencia se produce además entre el segundo y el tercero cogeremos solo el objeto más popular.

Ejemplo (4 gatos, 2 perros, 2 patos, 2 ratas, 1 paloma) -> objetos más populares: (gato)

Ejercicio 5.

Representa gráficamente y **muestra por pantalla** el número de coches encontrados por año, para cada ciudad. **Representa en una sola gràfica** los resultados de las tres ciudades.

Ejercicio 6.

En el dataset de **zurich** ha habido un poco de descontrol y se han añadido algunas imágenes que no han sido tomadas por la misma cámara ni en el mismo entrono. Diseña una función que sea capaz de encontrar una manera de identificar las imágenes que no pertenezcan a la ciudad de la forma más automatizada posible. Este es un **ejercicio libre** no hay una única manera de abordarlo, se valorará la capacidad de encontrar las imagenes y la creatividad de la respuesta. Razonad la respuesta y explicad los motivos de porque lo habéis hecho así. **Mostrad por pantalla**, los nombres de los ficheros de las imagenes intrusas y la motivación de la solución que proponéis.

Ejercicio 7.

Guarda toda la información en un fichero .csv con las siguientes columnas: nombre de la imagen, número de cocher, número de semáforos, número de personas, ciudad, año, si pertence o no a una ciudad (si la habéis detectado en ej.6). **Muestra por pantalla**, debidamente formatado, el nombre del archivo generadi y donde lo habéis guardado.

Observad que tenéis que generar código que permita **representar los resultados del ejercicio 3, 4 y 5 gráficamente y por pantalla (ejercicios 1, 2, 4, 5, 6 y 7)**.

El código tiene que estar ordenado, especialmente el fichero principal, que se utilizará básicamente para responder a la PEC con las llamadas a funciones necesarias. El código tendrá que estar correctamente comentado, con *docstrings* e incluyendo comentarios en el código para clarificar si es necesario. También tiene que estar testeado utilizando la librería `unittest`. Los tests proporcionados tendrán que tener una cobertura de como mínimo el 50% de la funcionalidad propuesta.

Cobertura de los tests

La medida de la cobertura de los tests se utiliza para evaular la eficacia de los tests propuestos. En particular, sirve para determinar la calidad de los tests y determinar las partes críticas del código que no han sido testadas. Para tal de medir este valor os proponemos el uso de la herramienta `Coverage.py` (https://coverage.readthedocs.io/en/coverage-5.3/). En la documentación, podéis encontrar como instalarla (https://coverage.readthedocs.io/en/coverage-5.3/install.html#install) y como usarla (https://coverage.readthedocs.io/en/coverage-5.3/#quick-start).

Criterios de corrección

Esta PEC se valorará siguiento los siguientes criterios:

- **Funcionalidad** (5.75 punts): Se valorará que el código implemente todo lo que se pide.
  - Ejercicio 1 (0.25 puntos)
  - Ejercicio 2 (0.75 puntos)
  - Ejercicio 3 (1 punto)
  - Ejercicio 4 (1.75 puntos)
  - Ejercicio 5 (0.5 puntos)
  - Ejercicio 6 (1 puntos)
  - Ejercicio 7 (0.5 puntos)
- **Documentación** (0.5 puntos): Todas las funciones de los ejercicios de esta PEC tendrán que estar debidamente documentadas utilizando docstrings (en el formato que prefiráis).
- **Modularidad** (1 punto): Se valorará la modularidad del código (tanto la organización del código en módulos como la creación de funciones).
- **Estilo** (0.5 puntos): El código tiene que seguir la guía de estilo de Python (PEP8), exceptuando los casos donde hacerlo complique la legibilidad del código.
- **Tests** (1.25 puntos): El código tiene que contener uno o diversas *suites* de tests que permitan comprobar que el código funciona correctamente, con un mínimo del 50% de cobertura.
- **Requeriments** (0.5 puntos): Tenéis que incluir un fichero de *requirements* que contenta la lista de librerías necesarias para ejecutar el código.
- **README** y **licencia** (0.5 puntos): Tenéis que añadir también un fichero README, que presente el proyecto y explique como ejecutarlo, así como la inclusión de la licencia bajo la que se distribuye el código (podéis escoger la que queráis).

Importante

Nota 1: De la misma manera que en las PECs anteriores, los criterios transversales se valararán de manera proporcional a la parte de funcionalidad implementada.

Por ejemplo, si el código solo implementa la mitad de la PEC y la documentación está perfecta la puntuación correspondiente a documentación será de 0.25.

Nota 2: Es imprescindible que el paquete que libréis se ejecute correctamente en la máquina virtual y que el fichero de REAMDE explique claramente como ejecutar el código con tan de generar los resultados pedidos. Además en el README tiene que explicarse también como se ejecutarán los test y como se comprueba su cobertura.

Nota 3: Entregad el paquete como un único archivo .zip que contega solo el código en el Registre d'Avaluació Continua. **El código de Python tendrá que estar escrito en ficheros planos de Python.**

In [ ]: