Herramientas Software para Tratamiento de Imágenes
Máster Oficial en Visión Artificial
Hoja de Problemas Evaluable

Instrucciones para la entrega

Una vez resueltos los problemas enunciados a continuación, el alumno utilizará el formulario habilitado en el moodle de la asignatura para subir los fuentes en un fichero zip. El nombre del fichero deberá formarse siguiendo el siguiente esquema: "Apellido1_Apellido2_Nombre.zip".

Fecha límite de entrega: 14 de Noviembre de 2017

Ejercicio 1

Desarrollar un módulo de aumentado de datos para redes de convolución en Python. Para trabajar con las imágenes (cargar/savar/modificar) utilizaremos el módulo PIL/pillow de Python. Además los alumnos trabajarán con un mismo dataset de imágenes: *tiny-imagenet200,* disponible para su descarga desde ImageNet.

El módulo de aumentado deberá permitir las siguientes operaciones:

- Blur, para un radio de kernel mínimo de 2 y máximo de 10
- Resize, para un factor de escala mínimo de 0.25 y máximo de 2.5
- rotate/transpsose, soportando cualquiera de las siguientes operaciones: PIL.Image.FLIP_LEFT_RIGHT, PIL.Image.FLIP_TOP_BOTTOM, PIL.Image.ROTATE_90, PIL.Image.ROTATE_180, PIL.Image.ROTATE_270 or PIL.Image.TRANSPOSE

El programa Python a desarrollar recibirá un factor de aumentado (ejemplo: 5x, 10x, 20x), generando (5, 10, 20) imágenes por cada imagen de entrada. El nuevo dataset resultante del aumentado se guardará en un directorio de salida. Por cada imagen de entrada el sistema **seleccionará aleatoriamente** una combinación de operaciones de aumentado a aplicar, junto con sus parámetros correspondientes (siempre dentro de los rango establecidos).

A continuación se proporciona la secuencia de argumentos que deberá soportar el programa presentado:

```
python data_augmentation.py --input_dataset=./tiny_imagenet --factor=20
                --output_dataset=./augmented_tiny_imagenet
```

Ejercicio 2

Desarrollar un sistema de visión artificial que permita hacer seguimiento visual por color de un objeto. El sistema debe ser capaz de detectar el objeto (en el caso de que se encuentre presente) en cada fotograma. El objeto puede moverse libremente, pudiendo cambiar su escala y la perspectiva con la que se ve el mismo. Para ello utilizaremos algunas de las rutinas del paquete de procesamiento de imagen de OpenCV.

Lo primero que necesitamos es ser capaces de segmentar el objeto a seguir en cada fotograma. Para ello convertiremos cada fotograma al espacio de color HSV Dado el rango de color en el que se mueve nuestro objeto, segmentaremos el fotograma generando una imagen binaria. A modo de ayuda, junto con el enunciado se proporciona una secuencia de video y los rangos de color HSV en los que se mueve el objeto que se quiere segmentar: 29<H>88, 43<S<255, 126<V<255. Nótese que el código presentado debe ser lo suficientemente general como para funcionar con otras secuencias cuyos objetos se muevan en rangos de color diferente.

El sistema de seguimiento visual propuesto consta de las siguientes etapas:

- Para cada fotograma de la secuencia
    - Aplicar un filtro gaussiano al fotograma para eliminar ruido
    - Convertir el fotograma al espacio de color HSV
    - Segmentar el fotograma utilizando el rango de color seleccionado
    - Eliminar pequeños focos de ruido en la imagen resultante:
        i. Aplicar dos operaciones de erosión consecutivas utilizando un elemento estructurante cuadrado de tamaño 3x3.
        ii. Aplicar dos operaciones de dilatación consecutivas utilizando un elemento estructurante cuadrado de tamaño 3x3.
    - Detectar los distintos contornos que aparecen en la imagen y quedarse con aquel que presente mayor área
    - Presentar en pantalla el resultado del seguimiento: mínimo círculo que engloba el objeto detectado en rojo, su centroide en verde y la trayectoria del mismo en azul.

Por último, se proporciona la secuencia de argumentos que deberá soportar el programa presentado.

```
python visual_tracking.py --video=./media/lapiz.avi --min_values=29 43
                    126 --max_values=88 255 255
```

Opcionalmente, el resultado del seguimiento puede querer serializarse en un video, para ello se debe proporcionar (o no), un cuarto parámetro llamado `--output`:

```
python visual_tracking.py --video=./media/lapiz.avi --min_values=29 43
        126 --max_values=88 255 255 --output=-/out/result.avi
```

Ejercicio 3

Queremos comparar los resultados obtenidos por un algoritmo de reconocimiento de objetos 3D. Por cada objeto reconocido se conoce su area2D, area3D y nivel de complejidad expresado como un número entero. Tanto los resultados generados por el algoritmo, como el *groundtruth*, se encuentran en ficheros csv diferentes (ver ficheros adjuntos con el enunciado).

Se desea analizar cada característica por separado, generando una gráfica por cada una de ellas. Interesa que este paso se lleve a cabo de forma automática, el algoritmo evoluciona, y no queremos perder tiempo generando manualmente las gráficas cada vez que mejoramos (o empeoramos) el método. Por su simplicidad a la hora de manejar grandes colecciones de datos y calcular estadísticos, elegiremos Python como lenguaje para desarrollar la tarea. Pasos a seguir:

- Leer los dos ficheros .csv y cargar los datos en dos NumPy arrays (`numpy.loadtxt`)
- Generar una gráfica de barras para el área2D similar a la que se muestra a continuación. Cada barra muestra el porcentaje de objetos cuya área2D difiere del original en un rango establecido (`matplotlib.pyplot.bar`). De forma adicional, la gráfica debe mostrar el porcentaje de objetos para los que no se pudo calcular área porque el proceso de reconocimiento falló (nota: un fallo en el cálculo de la una característica se marca con un carácter '-' en el fichero csv).
- Generar una gráfica equivalente, pero con el Área 3D.
- Generar una tercera gráfica donde se muestren las diferencias en nivel de complejidad. La complejidad se mide como un valor entero, por lo que cada barra representa el porcentaje de objetos cuya complejidad difiere en una determinada cantidad. Mostrar también una barra en la gráfica que indique el porcentaje de objetos para los que no se pudo calcular su complejidad.

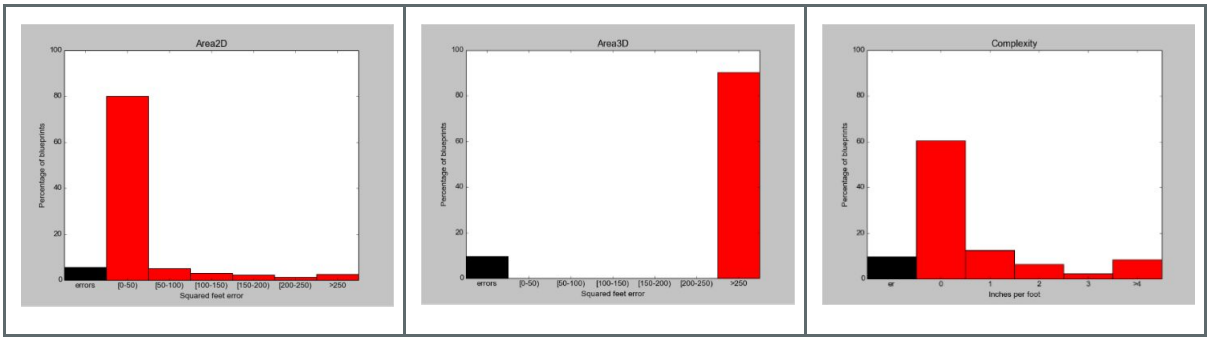
Para la resolución de la práctica se recomienda el uso de las siguientes funciones sobre NumPy arrays: `np.count_nonzero`, `np.isnan`, `np.sum`

El código deberá presentar como mínimo una función en Python llamada `compute_stats` que reciba tres argumentos:

- Dos rutas donde se encuentren los ficheros a comparar
- Y una tercera donde guardar las imágenes correspondientes a cada gráfica.

Así, el programa presentado deberá responder a la siguiente interfaz de llamada:

```
python compute_stats.py --inference=./results/detection.csv
--groundtruth=./gt/groundtruth.csv --output_graphs=./output_stats
```



Ejercicio 4

Convertir el ejercicio 3 en un Jupyter Notebook

Ejercicio 5

Dado un vídeo correspondiente al tráiler de una película (por ejemplo), utilizar un clasificador cascada (previamente entrenado) para detectar las caras y los ojos de las personas que aparecen en escena. El módulo de detección de objetos de OpenCV incorpora la posibilidad de trabajar con este tipo de clasificadores. Su uso es muy sencillo:

```
faceCascade = cv2.CascadeClassifier(faceCascadePath)

rects = faceCascade.detectMultiScale(image,
                                     scaleFactor = 1.1,
                                     minNeighbors=5,
                                     minSize=(30,30),
                                     flags = cv2.CASCADE_SCALE_IMAGE)
```

La primera llamada carga un clasificador previamente entrenado (utilizar los ficheros .xml proporcionados con la práctica para caras y ojos). La segunda detecta el objeto de interés, buscándolo en diferentes escalas de la imagen.
Nuestro programa en Python debe aceptar al menos un parámetro de entrada `-video` donde se indica la ruta del vídeo a cargar, y otro parámetro `-out` que indique la ruta donde se va a guardar el video resultante de la detección. La llamada a nuestro detector debe poder hacerse de la siguiente manera:

```
python eyefacedetector.py -video=./media/avengers.avi
                          -out=./avengers_result.avi
```

Ejercicio 6

Seguimos interesados en reconocer objetos en imágenes. Esta vez es el turno de peatones. Para ello utilizaremos la secuencia de imágenes proporcionada por el profesor. Para leer de una forma sencilla el directorio de imágenes se recomienda el uso de módulos como `os` o `glob`.

En esta ocasión utilizaremos un detector basado en histogramas de gradientes orientados (previamente entrenado para detectar personas).

```
hog = cv2.HOGDescriptor()
hog.setSVMDetector( cv2.HOGDescriptor_getDefaultPeopleDetector() )
...
rects, weights = hog.detectMultiScale(img, winStride=(8,8),
                         padding=(32,32), scale=1.05)
```

Nuestro programa en Python debe aceptar al menos un parámetro de entrada `-images` donde se indica la ruta donde se encuentra almacenada la secuencia, y otro parámetro `-out` que indique la ruta donde se va a guardar el video resultante de la detección. La llamada a nuestro detector debe poder hacerse de la siguiente manera:

```
python pedestriandetector.py -images=./images
                             -out=./pedestrian_result.avi
```

Ejercicio 7

Estamos interesados en identificar imágenes. Para ello vamos a trabajar en un enfoque basado en la detección y *matching* de puntos característicos. Por suerte, OpenCV presenta un módulo que implementa diferentes descriptores (SURF, SIFT, ORB, etc…) que simplifican la identificación de puntos característicos en imagen. Para comenzar a familiarizarnos con estos puntos característicos lo primero que haremos será seguir el tutorial que se nos proporciona desde la documentación de OpenCV. En él se detallan los pasos a seguir para identificar puntos homólogos en dos imágenes distintas.

Una vez tengamos el código del tutorial funcionando lo vamos a utilizar para identificar imágenes similares. Para ello nos crearemos un conjunto de fotos (podemos hacerlas con el móvil o descargarlas de internet) de carátulas de libros, cds, dvds…(lo que el alumno considere). A partir de aquí el alumno desarrollará una función en Python que reciba dos argumentos:

- Ruta donde se encuentra la imagen a emparejar
- Colección de imágenes candidatas

```
python mediamatcher.py –query=./cover_The_Hobbit
                       –covers=./my_media_database/
```

La función deberá mostrar en una ventana la imagen con la queda emparejada la imagen de consulta. De todas las imágenes se seleccionará aquella que obtenga mayor número de *matches* (o puntos emparejados). Fijaremos un umbral de un mínimo de N puntos emparejados. Si varias imágenes superan dicho umbral, se mostrará aquella que presente mayor número de *matches*.

Ejercicio 8

- Aplicar el módulo de detección y reconocimiento de texto (`text`) de OpenCV al problema de identificación de matrículas
    - el programa puede recibir como entrada una imagen o un vídeo, mostrando el resultado de la detección/reconocimiento en una ventana
    - En su versión más simple, el programa debería responder a la siguiente interfaz:

```
python text_recognition.py --image=./media/car.png
python text_recognition.py --video=./media/car.avi
```

- Hacer uso del módulo de seguimiento visual de OpenCV (`tracking`) para hacer seguimiento de un objeto o región de interés en una secuencia de video:
    - mostrar primer fotograma
    - seleccionar objeto o región de interés
    - arrancar seguimiento visual
    - mostrar resultado del seguimiento en una ventana

  En su versión más simple, el programa debería responder a la siguiente interfaz:

  ```
  python visual_tracking_cv.py --image=./media/bolt.avi
  ```