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: -

Enunciados

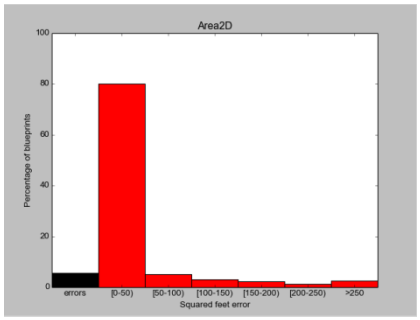
1. Queremos comparar los resultados obtenidos por un algoritmo de reconocimiento de objetos 3D. Por cada objeto reconocido se conoce su area2D, area3D y nivel de complejidad expresado como un número entero. Tanto los resultados generados por el algoritmo, como el *groundtruth*, se encuentran en ficheros 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 representará el porcentaje de objetos cuya complejidad difiere en una determinada cantidad. Mostrar también una barra en la

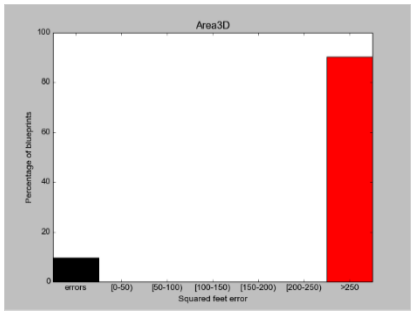
gráfica que indique el porcentaje de objetos para los que no se pudo calcular su complejidad.

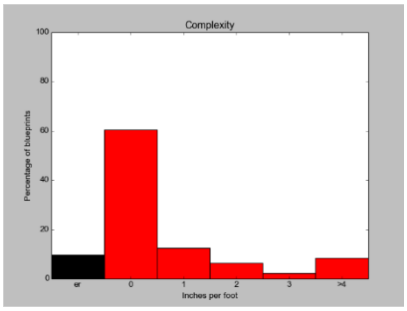
Para la resolución de la práctica se recomienda el uso de las siguientes funciones sobre NumPy arrays: `np.count_nonzero`, `np.isnan`, `np.sum`

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

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







2. 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

Para ello utilizaremos el paquete `argparse`:

```
import argparse
…
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", required = False,
    help = "path to where the video file resides")
args = vars(ap.parse_args())
…
path = args['video']
```

3. 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 del paquete `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