TRABAJO FINAL

Introducción

Este Trabajo Final corresponde a la segunda evaluación del curso Computer Vision. El trabajo consta de dos ejercicios prácticos con múltiples partes. Cada parte tiene un puntaje asignado específico, completando las partes básicas del trabajo se puede llegar a un 9 de nota, para llegar a la nota máxima (12) es necesario realizar alguna de las partes *bonus* de cada ejercicio o alguna de las sugerencias extras al final del trabajo. Cada ejercicio tiene distintos objetivos y son distintos para los alumnos de grado que para los de maestría.

El trabajo debe ser realizado en grupos de a dos y la fecha de entrega de mismo es el martes 22 de junio a las 23:50. Para la entrega del trabajo final va a existir una tarea en el Moodle del curso. En conjunto con la entrega de los trabajos el Jueves 24 de junio cada grupo va a tener que realizar una presentación de 10-15 minutos explicando la solución implementada. La evaluación final del trabajo depende de los resultados y la presentación realizada.

Para el segundo ejercicio de este trabajo se va a liberar un set de *testing* para evaluar los resultados el día antes de la entrega (Lunes 21 de junio).

La solución a los problemas debe ser provista en Python, se pueden utilizar los notebooks de referencia ó implementar su propio código en scripts.

La implementación inicial de los ejercicios se encuentra en el github del curso y los archivos de entrenamiento se pueden descargar del siguiente link Sharepoint

Parte 1: Modelos Geométricos

Esta primer parte consiste en analizar una imagen de una cancha de fútbol y utilizando un modelo geométrico, extraer la posición del las lineas exteriores y la linea central de la cancha en la imagen. La imagen de objetivo se puede ver en la Figura 1 y las lineas objetivo se pueden ver en la Figura 2.

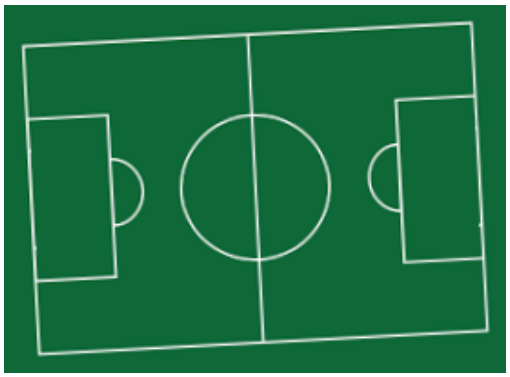


Figura 1: Imagen original de la cancha de Futbol

Para conseguir dicho objetivo es recomendable seguir los siguientes pasos:

1. Eliminar Ruido y obtener una imagen de bordes limpia de la cancha. (Se puede utilizar cualquier función de OpenCV).

2. Implementar una técnica de búsqueda y ajuste de lineas en la imagen. (Algunas de las opciones son RANSAC, Transoformada de Hough o hacer un barrido horizontal y vertical de la imagen para buscar lineas, Hay que implementar el método, solo se permite utilizar funciones para resolver sistemas por mínimos cuadrados)

3. Dependiendo de la técnica utilizada anteriormente va a ser necesario implementar un algoritmo de "Non-max-suppression"para elegir entre detecciones múltiples de la misma recta y definir una detección única entre lineas. (*Hint*: Para definir si dos rectas corresponden a la misma recta se sugiere analizar el ángulo entre las mismas y en que punto se intersectan con los bordes de la imagen ó el punto en que se intersectan las mismas). Es posible filtrar las lineas detectadas a mano, pero para una solución completa es necesario implementar no "Non-max-suppression".

La solución básica a este problema consiste en diseñar un algoritmo que sea capaz de detectar y dibujar automáticamente las rectas mostradas en la Figura 2.

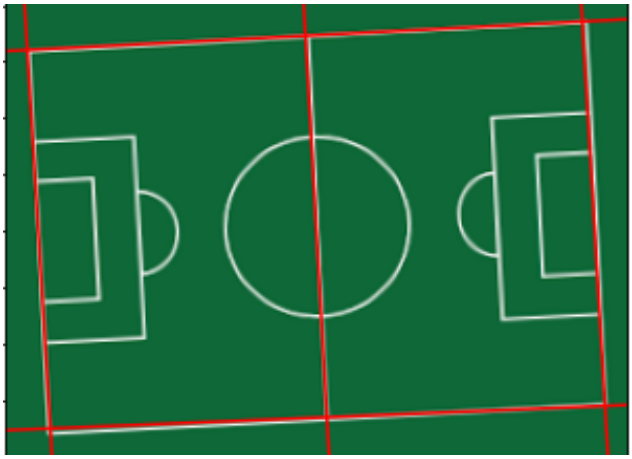


Figura 2: Rectas encontradas en la Figura

Para la evaluación de esta sección se van a obviar las lineas correspondientes a las áreas y el circulo central. Una vez obtenida las rectas de la imagen se pide reportar los puntos de intersección de las mismas. Para evaluar la calidad de la detección se va a medir el MSE (Mean Square Error) entre los puntos reales y los puntos encontrados.

[**Maestría**] Para los alumnos de maestría suma un punto **Bonus** Implementar un algoritmo capaz de encontrar el circulo central.

Parte 2: Detección y Reconocimiento Facial

2.1: Detección Facial

Para este segundo ejercicio vamos a implementar y entrenar un sistema de detección facial. Un *approach* clásico para este problema consiste en utilizar imágenes etiquetadas para construir un clasificador binario *face/non-face*. Una vez que se cuenta con un clasificador es posible utilizar una estrategia de *sliding-window* sobre una imagen de prueba para clasificar cada parche de la imagen. Las coordenadas de los *bounging-boxes* donde el clasificador tiene un *score* alto pueden ser devueltas como posibles caras en la imagen. Utilizando una estrategia de *non-max suppression* se pueden filtrar las detecciones repetidas. Para finalizar, como se vio en clase vamos a generar la curva de *precision-recall* del sistema y evaluar el AP (*Average Precision*) de la curva.

Para esta parte del trabajo se proveen los siguientes datos:

1. Un sub-set de caras recortadas del data-set LFW[1](Live Faces on the Wild) como imágenes positivas para entrenar el clasificador.

2. Un sub-set de imágenes con recortes random del *background* de las imágenes de LFW, para ser utilizados como ejemplos negativos de entrenamiento.

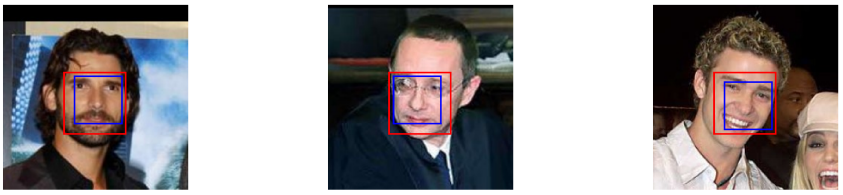
3. Un sub-set de imágenes sin recortar de LFW [2] para utilizar como set de validación.

4. Un *notebook* "FaceDetection.pynb" y código suplementario con una implementación simple del problema. En esta implementación se hace resize de las imágenes a 64x64, se vectoriza utilizando los píxeles de la imagen como *features* y se entrena un clasificador KNN. Una vez hecho esto se extraen parches aleatorios de la imagen y se clasifican. Luego se realiza el non-max suppression y se construye la curva de *precision-recall* para evaluar el AP del clasificador.

Si corren el código de ejemplo, van a poder ver que se carga una pequeña cantidad de imágenes y se entrena un clasificador KNN . Luego hay un sección que permite visualizar las detecciones (Rojo cara real, azul caras detectadas), esto se puede visualizar en la Figura 3. Para finalizar se utiliza el set de validación completo para evaluar el detector y calcular el AP.

La implementación provista para hacer detección facial no es muy buena. La consigna consiste en usar los conocimientos provistos en clase para diseñar un mejor detector de caras. La implementación provista debería obtener alrededor de 0.01 AP. Una buena solución a este problema que obtenga todos los puntos debe tener entre 0.35 AP y 0.60 AP. Una solución para obtener puntos **bonus** debe estar por encima de 0.60 AP.

[1] http://conradsanderson.id.au/lfwcrop/
[2] http://vis-www.cs.umass.edu/lfw/



(a) Eric Bana (b) Hans Leistritz (c) Justin Timberlake

Figura 3: Detección de caras en LFW

[**Maestría**] Para los alumnos de maestría se pretende que una solución que tenga todos los puntos esté por encima de 0.5 AP y una solución que tenga puntos **bonus** consiste en tener un 0.7 AP o mayor.

Para obtener una mejor solución se sugiere:

1. Obtener mejores features para clasificar, pueden usar *scikit-image* para obtener features HoG o LBP.
2. Entranar un clasificador más rápido y más preciso que KNN, se sugiere utilizar el clasificador SVC de sklearn y hace una busqueda de hyperparámetros.
3. Implementar un algoritmo de *sliding-window* poder extraer parches de la imagen.

Si realizan estos pasos es posible obtener resultados entre 0.35-0.60 AP. Para sobrepasar estos resultados es necesario:

- Descargar más datos positivos y negativos de LFW para entrenar el clasificador.
- Afinar los parámetros de HoG, LBP y el SVM utilizado.
- Ajustar su algoritmo de sliding window para que trabaje a múltiples escalas.
- Utilizar Deep Features. (Redes neuronales para extraer features)

Si se realiza esto es posible obtener resultados muy por encima de 0.60 AP. Cualquiera de estas mejoras puede sumar puntos **(bonus)** si el AP sobrepasa el esperado.

[**Maestría**] Para los alumnos de maestría en este ejercicio ademas de los objetivos planteados se pide hacer transfer learning o reentrenar un modelo de redes neuronales para clasificar cada parche de la imagen suplantando al clasificador KNN o SVM. No es necesario que mejore los resultados, pero si que este entrenado y validado correctamente. Pueden tomar como punto de partida el modelo MobileNet disponible en Keras.

Reglas

- Se permite utilizar cualquier extractor de features. Se permite utilizar cualquier clasificador siempre y cuando sea entrenado por los estudiantes.
- No se permite utilizar soluciones pre-entrenadas ni soluciones de detección de objetos *of-the-shelf*. En necesario que implementen su propio sliding window y clasifiquen cada ventana de una imagen para este trabajo.

2.2: Reconocimiento Facial

El reconocimiento Facial consiste en solucionar un problema de clasificación entre C clases conocidas. En general se resuelve utilizando un clasificador capaz realizar predicciones *multi-class*. Para entrenar un sistema de reconocimiento facial es necesario tener suficientes imágenes de entrenamiento de las C clases en las que se quiere clasificar.

En esta parte vamos a entrenar un clasificador para hacer reconocimiento facial, para ello es necesario:

- Cargar los datos de los archivos 'face_recognition_tr.mat' y 'face_recognition_va.mat' para entrenar y validar el clasificador respectivamente. Estos datos fueron extraidos de LFW.
- Se provee una implementación simple del problema en el *notebook* "Face_Recognition.pynb".

Si corren el código de ejemplo se puede ver que el sistema de reconocimiento implementado tiene un *accuracy* 16 %. Teniendo en cuenta que se va a clasificar en 35 clases distintas, un sistema de reconocimiento basado en decisiones aleatorias debería obtener 1/35=2.8 % Accuracy.

Para mejorar estos resultados se pueden realizar las siguientes mejoras.

1. Extraer mejores Features. Los features HoG y LBP de la primera parte son un buen punto de partida. Optimizar los feature HOG o implementar un descriptor LBP basado en celdas como en el siguiente blog. [3]
2. También se pueden intentar mejorar utilizando un BoW normalizado.
3. Utilizar un mejor clasificador. Un SVM lineal como el utilizado en la primera es un buen comienzo.

[3] ff

Para este ejercicio se pide tener un algoritmo de reconocimiento con al menos 40 % de accuracy. Implementando mejoras es posible obtener alrededor de 62 % de accuracy en reconocimiento. Se puede utilizar el código provisto para visualizar los resultados de la tarea como se visualizan en la Figura 4. Si se sobrepasa un Accuracy de 62 % se obtiene un punto **(bonus)**.

[**Maestría**] Para los alumnos de maestría el mínimo para obtener todos los puntos es 50 % de accuracy.

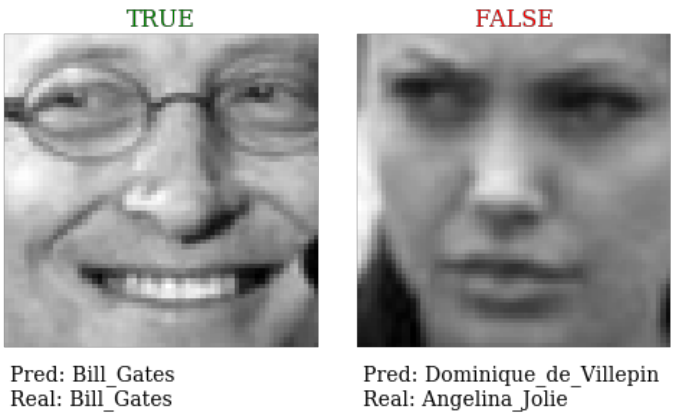


Figura 4: Reconocimiento Facial LFW

Extras

- Generar una aplicación que haga en conjunto detección y reconocimiento Facial sobre cualquier imagen.
- Implementar alguna aplicación interesante que utilice cualquiera de las tecnologías implementadas en este trabajo.
- Para los alumnos de grado implementar alguno de los extras de maestría implica puntaje extra.

Presentación

Para la presentación se sugiere que tenga 9 slides. Una con el titulo del trabajo y nombres de los estudiantes. Dos slides para la primera parte una para describir la metodología usada y otra para reportar los resultados. Para la segunda parte del trabajo se sugieren dos slides por sección para mostrar la metodología y resultados de cada parte. Para finalizar se pueden usar

los slides restantes para mostrar los resultados de las partes bonus y los extras implementados con sus resultados.

En conjunto con la presentación se debe mostrar un demo de las distintas partes funcionando.

Importante La presentación también lleva parte de la nota del trabajo y es importante que sepan explicar el trabajo que hicieron y la metodología más alla de los resultados.

Importante No es necesario que implementen todas las partes bonus, el trabajo debería llevarles unas 10-15 hs por persona.