TRABAJO FINAL 2023

Importante: *(Eviten el Plagio)*

En los ejercicios del trabajo final es posible utilizar funciones de librerías existentes o código sacado de internet siempre y cuando no se usen para resolver explícitamente lo que el ejercicio pide implementar y siga los lineaminetos del ejercicio.

Si eligen utilizar código sacado de interenet es necesario agregar el link en comentarios de donde fue sacado ese código.

Cruzar información entre grupos y utilizar el mismo código para resolver los problemas es considerado plagio

Introducción

Este trabajo final corresponde a la segunda evaluación del curso Computer Vision. El trabajo consta de tres ejercicios prácticos con múltiples partes. Cada parte tiene un puntaje asignado específico. Completando las partes 3 partes básicas del trabajo se puede llegar a un 10 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.

El trabajo debe ser realizado en grupos de a dos y la fecha de entrega de mismo es el Domingo 18 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 Miercolés 21 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 los ejercicios 2 y 3 de este trabajo se va a liberar un set de *testing* para evaluar los resultados el día antes de la entrega (Viernes 16 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

Ejercicio 1: Image Stitching

Esta primer parte consiste en analizar un set de imágenes provenientes de la misma escena y generar una imagen panorama. Para ello va a ser necesario aplicar los conocimientos vistos en el curso para detectar puntos de interés y calcular las transformaciones que llevan de una imagen a otra. Las imágenes de entrada se pueden ver en la Figura 1. El objetivo inicial es poder armar una imagen panorama con al menos tres de las imágenes de entrada.

Para conseguir dicho objetivo es recomendable seguir los siguientes pasos:

TRABAJO FINAL 2023

Figura 1: Cinco imágenes de la misma escena a alinear

1. Detectar puntos de interés y extraer descriptores para todas las imágenes (Se pueden utilizar las funciones provistas por OpenCV).
2. Implementar su propia técnica de *matcheo* robusto de puntos entre imágenes. Para ello es necesario utilizar los conocimientos vistos en el curso.
3. Implementa el cálculo robusto (ej: Utilizando RANSAC) de la transformación que mapea cada una de la imágenes con sus imágenes adyacentes.

La solución básica a este problema consiste calcular las transformaciones afín que mapean de una imagen a otra. En la Figura 2 se puede ver el resultado final esperado al alinear las 5 imágenes.

Figura 2: Imágenes Alineadas

Los objetivos de este ejercicio son:

1. Obtener una imagen panorama con al menos 3 de las imágenes provistas calculando una transformación afín.
2. **Bonus** Alinear las 5 imágenes provistas correctamente.

3. **Bonus** Calcular homografías en vez de transformaciones Afín. Se puede consultar el siguiente link para saber como calcular la homográfia

Este ejercicio está basado en un ejercicio propuesto por Joshep Redmond en la unviersidad de washington

Ejercicio 2: Detección de caras

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 CelebA[1](Large-scale CelebFaces Attributes) como imágenes positivas para entrenar el clasificador.

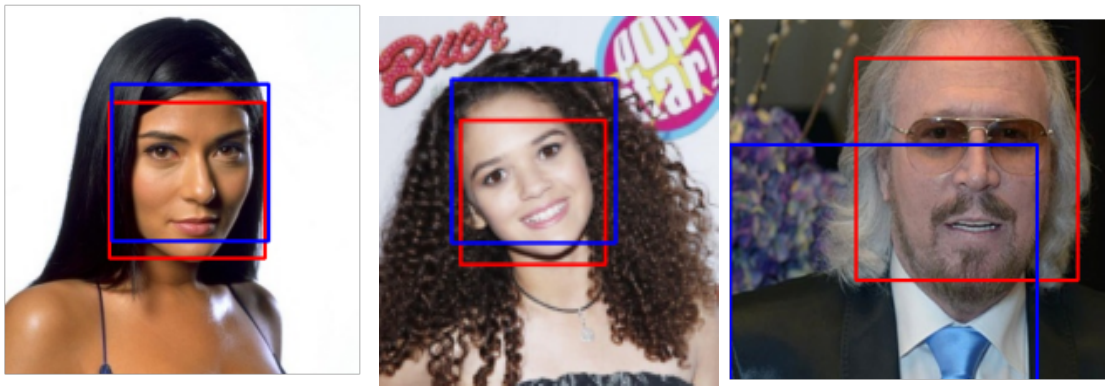
2. Un sub-set de imágenes con recortes random del *background* de las imágenes del mismo dataset para ser utilizados como ejemplos negativos de entrenamiento.

3. Otro sub-set de imágenes sin recortar de CelebA para utilizar como imágenes 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 (en rojo cara real y en 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.

[1] https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html



(a) Detección Correcta (b) Detección Correcta (c) Detección Incorrecta

Figura 3: Visualización de Detecciones

La implementación provista para hacer detección facial no es buena. La consigna consiste en usar los conocimientos provistos en clase para diseñar un mejor detector de caras. La implementación provista debería obtener un AP menor a 0.01 . Una buena solución a este problema que obtenga todos los puntos debe tener entre 0.20 AP y 0.45 AP. Una solución para obtener puntos **bonus** debe estar por encima de 0.45 AP.

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* a múltiples escalas para poder detectar caras de distintos tamaños.

Si realizan estos pasos es posible obtener resultados entre 0.20-0.45 AP. Para sobrepasar estos resultados es necesario:

- Afinar los parámetros de HoG, LBP y el SVM utilizado.
- Ajustar su algoritmo de sliding window para que trabaje a múltiples escalas.
- Utilizar otros algoritmos para extracción de features mejores que HoG o LBP

Si se realiza esto es posible obtener resultados por encima de 0.45 AP. Cualquiera de estas mejoras puede sumar puntos **(bonus)** si el AP sobrepasa el esperado.

Si se implementa su propio extractor de carácteristicas HoG o LBP sin utilizar *scikit-image* u otras librerías que calculen el descriptor y se utiliza para resolver el problema se puede obtener un punto **(bonus)**

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.

Ejercicio 3: Clasificación de Sonrisas

La clasificación de sonrisas en imágenes de caras consiste en un problema de clasificación con dos clases *Sonrisa/No Sonrisa*. Este problema se puede resolver utilizando un clasificador capaz realizar predicciones *one-class*. Para entrenar el clasificador en cuestión necesario contar con datos de la etiqueta de *Smile* de cada una de las imágenes de entrada.

En esta parte vamos a entrenar un clasificador de dos clases para clasificar si una cara esta sonriente o no, para ello es necesario:

- Cargar los datos de los archivos `training_labels.pkl` y `validation_labels.pkl` para entrenar y validar el clasificador respectivamente. Estos datos fueron extraidos de CelebA.
- Es necesario recortar las imágenes de validación con los bounding boxes provistos en el problema anterior para validar correctamente su solución.
- Para este problema no se provee una solución inicial, es esperable que los alumnos puedan crear un clasificador que clasifique caras entre caras sonrientes y no sonrientes y puedan validarlo correctamente.

Es necesario implementar una solución original basada en un clasificador clásico o en una solución de deep learning. Una solución básica de este problema se espera que tenga un porcentaje de *accuracy* 85 %. Teniendo en cuenta que se va a clasificar en 2 clases distintas, un sistema de clasificación de género con decisiones aleatorias debería obtener 1/2=50 % Accuracy.

Para obtener los resultados esperados se pueden seguir las siguientes opciones:

1. Implementar una red de convolución simple como la vista en el práctico y entrenar desde cero nuestra red de clasificación.
2. Utilizar una con pesos preentrenados para hacer transfer learning sobre la tarea que vamos a resolver.

3. Se puede implementar un clasificador basado en features HoG o LBP como en el ejercicio anterior.

Para este ejercicio se pide tener un algoritmo de clasificación con al menos 85 % de accuracy. Si se implementa una solución basada en deep learning y se sobrepasa el 91 % de accuracy se puede obtener un punto **(bonus)** en este problema. En este problema es necesario entrenar su propio clasificador y validar correctamente los resultados.

Extras

- Generar una aplicación que haga en conjunto detección y clasificación de sonrisas sobre cualquier imagen de caras.
- Implementar alguna aplicación interesante que utilice cualquiera de las tecnologías implementadas en este trabajo.

Presentación

Para la presentación se sugiere que tenga 9 slides. Una con el titulo del trabajo y nombres de los estudiantes. Se sugieren Dos slides para la cada uno de los ejercicios para: una describir la metodología usada y otra para reportar los resultados. 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 allá de los resultados. **Se espera que la presentación dure de 10 - 12 min**.

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