# ÍNDICE DE LAS PRÁCTICAS

## PRÁCTICA 4

---PETIA EXPLICATION----------

**First Part (A): Template Matching and Image Descriptors**

4.1) SSD and Normalized Cross-correlation for template matching

4.2) HOG image descriptor for object (person) detection

**Second Part (B): Image matching**

4.3) Recognition by correspondance, based on feature extraction (ORB)

---ANGELA EXPLICATION----------

**First Part (A):**

**1. 1.** Distancia euclidiana y distancia normalized cross-correlation con **Einstein** y su ojo.

**1. 2.** Cambiamos el contraste y vemos que la euclidiana falla, pero la cross se mantiene.

Se muestran máximos y mínimos de las distancias.

Se cambia la luz de los ojos de Einstein a oscura y ambas imágenes fallan.

**1.3** Ahora cambia el brillo del ojo y la distancia euclidiana no funciona y la otra sí.

Cambiamos también el grado del ojo y vemos que a medida que hay más rotación ninguna de las dos distancias funciona

**2.1.** Se obtiene el HOG (Histogram of Oriented Gradients) de una imagen tamplate de una **persona**.

**2.2. a.** Se obtiene el HOG para el resto de personas para la detección y se compara con el show_images (plot).

**2.2. b.** Visualizamos la parte de la imagen que es más parecida al template de la persona (rectángulo rojo) usando la distancia entre el template y los descriptores de las imágenes test. Nos piden el tamaño del HOG y nos hacen probar con diferentes parámetros.

Finalmente explicamos el HOG, con sus ventajas y sus desventajas.

**Second Part (B):**

**3.1.** Encontrar los puntos clave de una imagen del **Starbucks** (detectar la censure).

**3.2.** Encontrar las correspondencias entre la imagen modelo y la nueva imagen.

Lo haremos con toda la colección y hacemos que se muestren primero las más parecidas (sort).

**3.3.** Repetimos con diferente rotación y escala. Vemos que ORB object detector es invariante a rotación, iluminación y a escala un poco, mientras el HOG es variante a la rotación.

# PRÁCTICA 5

---PETIA EXPLICATION----------

## Image search using textures

### First Part (A):

1) Gaussian filters

2)Descriptors based on texture

### Second Part (B):

1) Distance between images and similarity search

---ANGELA EXPLICATION----------

### First Part (A):

**1.1.** Visualizamos el banco de filtros, cuantos son y su tamaño.
**1.2.** Visualizamos una **pizza** con resize y escala de grises.
**1.3.** Hacemos una función para mostrar n filtros del banco aplicados a la foto.
**1.4.** Es como el 1.3 pero con una foto de un **perro** y una **flor**.
**1.5.** Hacemos una función para extraer el vector de características.
**1.6. (2.2)** Probamos la función anterior para la imagen del perro y de la flor.

**2.1** Guardamos en un array las colecciones de pizzas, perros y flores con resize, mirando cuantas imágenes hay en cada una.
**2.2.** Implementamos una matriz que nos devuelva para cada imagen su vector de características.
**2.3.** Imprimir el tamaño del vector de características y las características de algunas de las imágenes.
**2.4.** Dadas tres imágenes se muestra el valor de los filtros con un plot de tres colores.
**2.5.** 2.4 con otras imágenes.
**2.6.** (2.5) Calculamos la distancia euclidiana entre la imagen y el resto de las imágenes del vector y mostramos la que tiene menos distancia.
**2.7.** (2.6) Repetimos 2.5 con las imágenes de los perros y las flores.

### Second Part (B):

**3.1** Cargar directorios de **pizzas**, **flores** y **perros** en un array con resize.

**3.2** Función para visualizar las n fotos más parecidas, según la distancia l2 norm y con un plot de las distancias.

**3.3** Lo mismo que el 3.2 pero con otra n.

**3.5** Función para medir la precisión con la que acierta (las pizzas p. e.) el algoritmo.

**3.6** Usamos todas las funciones con unos determinados parámetros.

**3.8** Función que devuelve las características basadas en el descriptor de color.

**3.9** Guardamos en la variable X_lm_rgb el descriptor de color de todas las imágenes.

**3.10** Calculamos la precisión de las imágenes retornadas con la información de color y hacemos un plot de las imágenes más cercanas a color.

**3.11** Calculamos la precisión de las imágenes retornadas con la información de la textura y hacemos el plot de las 4 imagenes.

# PRÁCTICA 6

---PETIA EXPLICATION----------

## Image search using textures

### First Part (A):

- Integral images and a classical use for fast harr-like feature computation.
- Use of Adaboost for classification.
- Decisions based on a user-defined threshold for balancing precision and recall.

### Second Part (B):

- Define an appropriate representation (descriptor objects)
  - Normally, reduce size of the data preserving the invariance and removing redundant dimensions.
- Train a classifier from a set of examples with their descriptors.
- Recognize a new face example using the learned model.

---ANGELA EXPLICATION----------

### First Part (A):

**1.1** Crear una imagen integral.

**1.2** Crear una imagen binaria y compararla con la imagen integral.

**1.3** Hacer test para comprobar que la imagen integral es correcta.

**1.4** Hacer la imagen integral de una imagen real.

**1.5** Explicar unas igualdades sobre la imagen integral.

**1.6** Comparar nuestra función hecha a mano con la función de Python para integrar imágenes.

**2.1** Obtener las Haar-like features.

**2.2** Función para visualizar las características Haar en imágenes de **caras**.

**3.1** Crear un array con las características de las caras y no caras y otro con etiquetas para las caras (1) y las no caras (0).

**3.2** Dividimos el dataset en train y en test.

**4.1** Entrenamos el clasificador Adaboost.

**4.2** Evaluamos la precisión del clasificador, probando con diferentes números de estimadores.

**4.3** Función para visualizar las características más importantes de una imagen.

**4.4** Evaluamos el clasificador aumentando los ejemplos de las caras.

### Second Part (B):

**1.1** Cargamos el dataset con las caras que tengan por lo menos 100 imágenes. Miramos sus características.

**1.2** Escogemos una imagen para cada clase, es decir, para cada **político**.

**1.3** Dividimos el dataset en train y test.

**2.1** Creamos el objeto PCA con 150 componentes y hacemos un plot de la varianza acumulada.

**2.2** Calcula la cara promedio con el PCA, haciendo la media.

**2.3** Calculamos las eigenfaces.

**2.4** Proyectamos (transform) en la base del PCA el conjunto de train y test.

**2.5** Recomponemos una cara mediante los componentes.

**3.1** Entrenamos el clasificador de Adaboost con las características del PCA.

**3.2** Mostramos la efectividad del clasificador explicando cada parámetro.

**3.3** Entrenamos el clasificador Adaboost sin el PCA y evaluamos su precisión.

**4.1** Predecimos el nombre de los políticos usando el Adaboost con y sin el PCA.