**CURSO:** WS7A – INTELIGENCIA ARTIFICIAL

**CLASE:** SESION #21 (LABORATORIO)

**TEMA:** APRENDIZAJE Y EVALUACION DE REDES HOPFIELD EN PYTHON | DETECCION DE ROSTROS EN PYTHON

**PROFESOR/A:** Ing. PATRICIA REYES SILVA

En esta clase, veremos en la práctica:

- El uso de las librerías **neurodynex3.hopfield_network** y **OpenCV** para la implementación de redes neuronales recurrentes Hopfield.
- Off-Topic: Como detectar rostros en imágenes y videos utilizando las librerías de Python mediapipe y opencv.

## OBJETIVOS

(1) Implementar y probar en Python el algoritmo de aprendizaje y evaluación que desarrollan las redes Hopfield.
(2) Aprender a detectar rostros en imágenes y videos en Python.

## ACTIVIDADES

### (1) **Objetivo:** Implementación y evaluación de Redes Hopfield

Descargaremos los (3) tres notebooks Python contenidos en el archivo zip **lab09-HNN-RedesHopfield** y los editaremos en Jupyter de Anaconda o en Google Colab. El enlace de descarga está AQUÍ.

A continuación, ejecutar en el siguiente orden los Notebooks:

#### (1) HNN-Patrones-Aleatorios

Abrir el notebook HNN-Patrones-Aleatorios.ipynb.

Instalar previamente la librería neurodynex3: pip install neurodynex3

Utilizar los módulos hopfield_network.network, hopfield_network.pattern_tools y hopfield_network.plot_tools para aprender los elementos básicos de esta red.

Pasos a seguir:

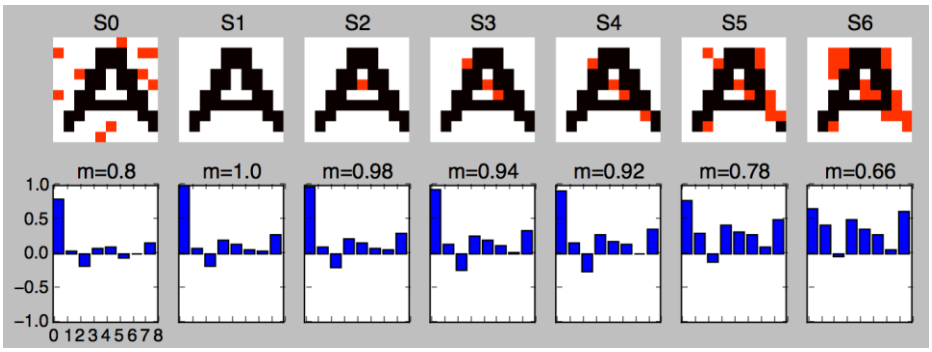
- Ejecutar el código.
- Leer los comentarios en línea y consultar la documentación de ser necesario.
- Los patrones y los píxeles alterados se eligen al azar. Por lo tanto, el resultado cambia cada vez que ejecuta este código.
- Ejecutar varias veces y cambiar algunos parámetros como **nr_patterns** y **nr_of_flips**.

## (2) HNN-Patrones-NO-Aleatorios

Abrir el notebook HNN-Patrones-NO-Aleatorios.ipynb.

En el ejercicio anterior utilizamos patrones aleatorios. Ahora tenemos una lista de patrones de entrada estructurados: las letras de la A a la Z.

Cada letra está representada en una cuadrícula de 10 por 10.




Se almacenan ocho letras (incluida la 'A') en una red Hopfield. La letra 'A' no se recupera.

Pasos a seguir:

- Ejecutar el código.
- Leer los comentarios en línea y buscar en el código las diferencias entre este código y el del notebook anterior HNN-Patrones-Aleatorios.ipynb.

## (3) HNN-Patrones-en-Imagenes

Abrir el notebook HNN-Patrones-en-Imagenes.ipynb.

Deberemos instalar OpenCV para el tratamiento de imágenes en Python. Según la plataforma de trabajo utilizada, ejecutar:

pip install opencv-python

conda install opencv

Comprobar también que la imagen deal-with-it-with-text3.jpg se encuentre en la misma carpeta.

Pasos a seguir:

- Ejecutar el código.
- Leer los comentarios en línea y entender como:
  - Cargar una imagen en OpenCV
  - Convertir la imagen a una escala de grises (2 colores)
  - Convertir la imagen a un arreglo de enteros (que serán los pixeles)
  - A partir del arreglo de enteros, convertir dicho arreglo a uno de tipo bipolar (1 para matices negros y -1 para blancos).
  - Obtener la matriz de pesos de la imagen (ahora en un arreglo bipolar).

- Generar una versión ruidosa de la imagen y mostrarla.
- Ejecutar la función **flow** que implementa la función de activación de la red Hopfield, actualiza energía y logra una como salida una imagen ruidosa evolucionada (la más parecida al patrón (imagen) inicial.

(2) **Objetivo:** Detección de rostros en imágenes y videos en Python

Descargaremos los (2) **dos códigos Python**, así como las carpetas **images** y **videos** contenidos en el archivo zip **lab10-Deteccion-Rostros,** los cuales editaremos en nuestro editor de preferencia. El enlace de descarga está AQUÍ.

Utilizaremos la librería mediapipe de Python para la detección de rostros. Esta librería detecta seis puntos clave en el rostro detectado y soporta la detección de múltiples rostros en una misma imagen.

Mediapipe está basado en BlazeFace (detector de rostros liviano y excelente rendimiento, que fue desarrollado para su uso en GPU móvil. Mayor detalle en https://arxiv.org/abs/1907.05047)

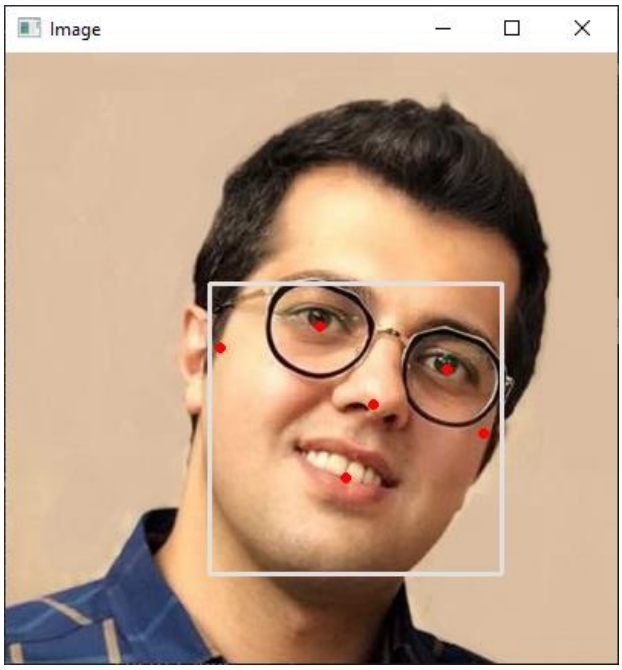
Los seis puntos clave en el rostro son:

- RIGHT_EYE
- LEFTT_EYE
- NOSE_TIP
- MOUTH_CENTER
- RIGHT_EAR_TREGION
- LEFT_EAR_TREGION

Pasos a seguir:

- Instalar la librería **mediapipe** vía: pip install mediapipe
- Ejecutar el código en la línea de comandos: Python mp-face-detection.py

  Obtenemos la siguiente imagen:

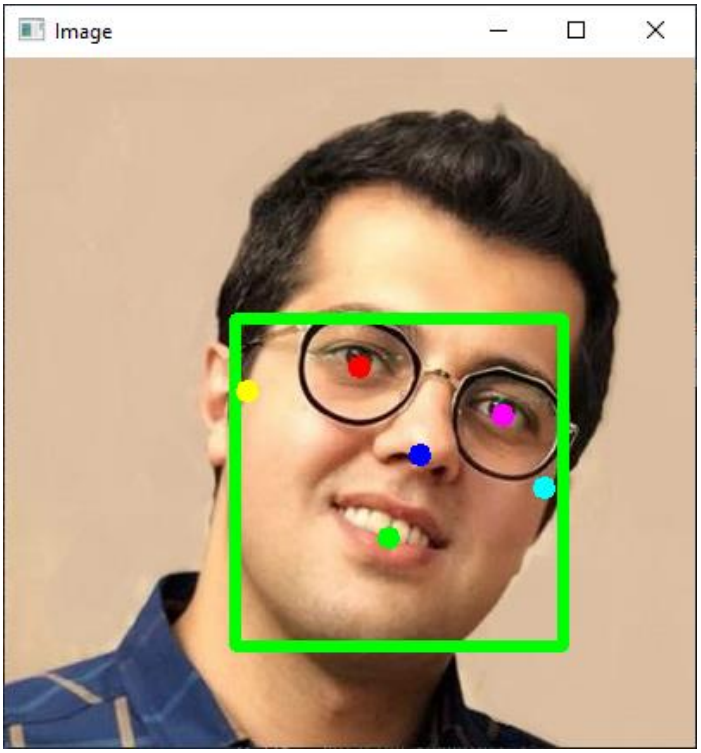



- Observamos que se marca el área del rostro encontrado en la imagen y se referencia cada uno de los puntos clave en el rostro con un punto de color rojo.

- Adicionalmente, observamos la salida en la consola:

```
(base) E:\Patricia\developer\upc-WS7A\lab10-Deteccion-Rostros>python mp-face-detection.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Resultados: [label_id: 0
score: 0.92535496
location_data {
  format: RELATIVE_BOUNDING_BOX
  relative_bounding_box {
    xmin: 0.3349673
    ymin: 0.37886682
    width: 0.4761578
    height: 0.47607878
  }
  relative_keypoints {
    x: 0.51364064
    y: 0.4489303
  }
  relative_keypoints {
    x: 0.7213124
    y: 0.5191835
  }
  relative_keypoints {
    x: 0.6013636
    y: 0.57712555
  }
  relative_keypoints {
    x: 0.555817
    y: 0.6962379
  }
  relative_keypoints {
    x: 0.35035902
    y: 0.48320618
  }
  relative_keypoints {
    x: 0.78212583
    y: 0.62284315
  }
}
]
```

- La variable location_data es un vector que almacena cada recuadro de rostro detectado y en él, cada punto clave del rostro, del 0 al 5 (índice). A través de esta variable, podremos acceder a cada uno de estos puntos y referenciarlos.
- Ahora, Des comentar el código comprendido entre la línea 30 a la 69 y comentar lo comprendido entre la línea 71 y 82. Ejecutar nuevamente.



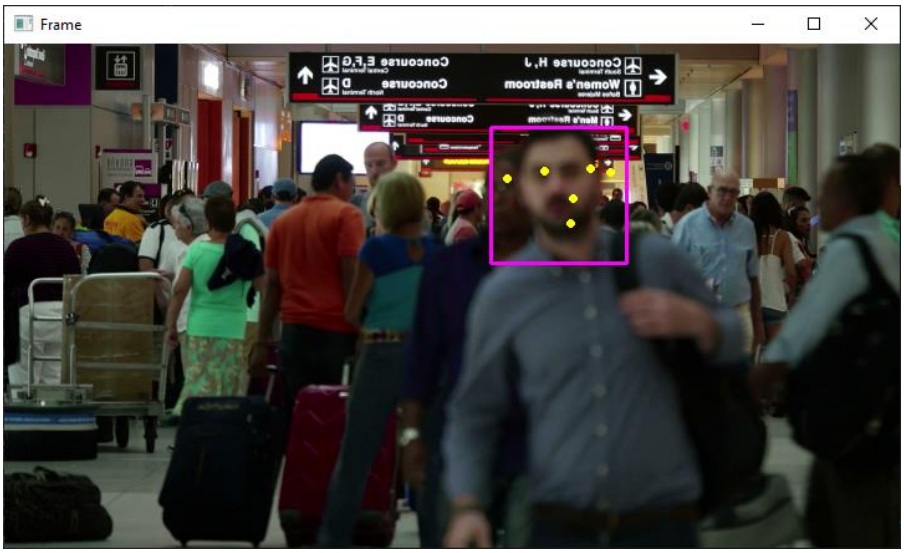

- Con el nuevo código, hemos podido acceder a cada uno de los puntos clave del rostro y modificado tanto el color como el grosor de cada uno de los puntos clave detectados.
- Ahora probar con algunas de las imágenes contenidas en la carpeta **images** (tanto para imágenes de rostros simples como múltiples rostros).

  El siguiente código es una variación del anterior, la diferencia está en que identificaremos rostros en un video en lugar que una imagen:

- Instalaremos la librería imutils (utilitarios para el tratamiento de imágenes) vía: pip install imutils
- Con la librería imutils podremos cambiar el tamaño del frame de nuestro video.
- Ejecutar el código: mp-face-detection-video.py
- Este código carga el video contenido en la carpeta video, sin embargo, pueden utilizar su webcam para capturar video y detectar su propio rostro mediante la siguiente instrucción (está comentada):

  cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

- Visualizaremos el siguiente video:




- Probar con su propia webcam y/o descargando videos de personas con sus rostros visibles.