

Desarrollo de aplicaciones de reconocimiento facial con base en los resultados del análisis de tres familias de algoritmos

GERMÁN QUIROZ GONZÁLEZ

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
DIRECTORES DE TESIS:
M. EN C. PABLO MANRIQUE RAMÍREZ
DR. EUSEBIO RICARDEZ VÁZQUEZ

21 DE NOVIEMBRE DE 2022



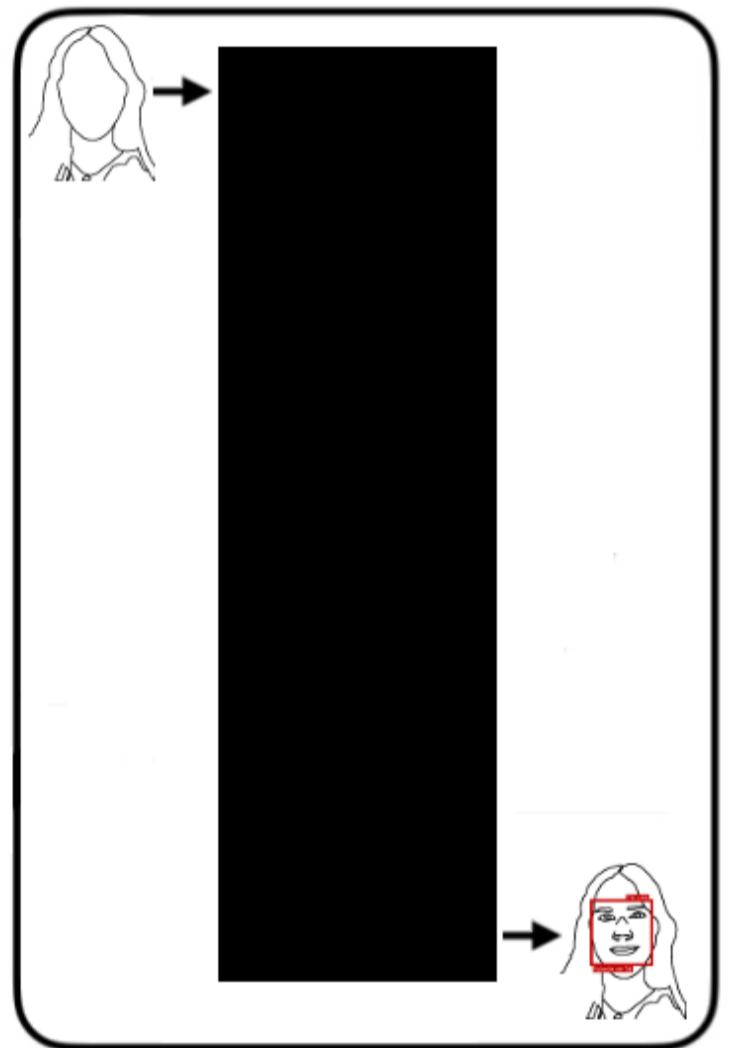
Buenos días, gracias por su asistencia.

Se presenta la tesis con el tema:

Desarrollo de aplicaciones de reconocimiento facial con base en los resultados del análisis de tres familias de algoritmos
por parte de German Quiroz, Egresado del CIC-IPN

Los directores de esta tesis son
el maestro en ciencias Pablo Manrique Ramírez y el doctor Eusebio
Ricardez Vazquez Ciudad de México, noviembre 22, 2022.

PLANTEAMIENTO DEL PROBLEMA



Planteamiento del problema

Se tiene la cara de una persona que se presenta ante algún artefacto cuyo funcionamiento está por detallarse. Como resultado se obtiene la probabilidad de que esa persona corresponda a una identidad previamente registrada.

JUSTIFICACIÓN

- Los sistemas de control de acceso comerciales son, en general, **cajas cerradas** que no pueden ser modificadas.
- En México no hay, a la fecha, un sistema de control de acceso elaborado **completamente** con herramientas de código abierto al alcance de todo el público.
- Tampoco existe una explicación de lo que hace **cada uno** de sus componentes ni una **guía para integrar** cada uno de ellos.
- El material cubierto en este trabajo puede ser usado para **otros emprendimientos** tales como en el área de entretenimiento¹ y **navegación autónoma** vía radar.²

¹Mediante visión por computadora.

²Principalmente los algoritmos de detección y clasificación. Heuel, Steffen, Rohling, Hermann. "Pedestrian Recognition Based on 24 GHz Radar Sensors in Ultra-Wideband Radio Technologies for Communications, Localization and Sensor Applications, edited by Reiner Thomä et al. London: IntechOpen, 2013. 10.5772/53007.

JUSTIFICACIÓN

- Los sistemas de control de acceso comerciales son, en general, cajas cerradas que no pueden ser modificadas.
- En México no hay, a la fecha, un sistema de control de acceso elaborado **completamente** con herramientas de código abierto al alcance de todo el público.
- Tampoco existe una explicación de lo que hace cada uno de sus componentes ni una guía para integrar cada uno de ellos.
- El material cubierto en este trabajo puede ser usado para otros emprendimientos tales como en el área de entretenimiento mediante vision por computadora y y **navegación autónoma** vía radar, principalmente los algoritmos de detección y clasificación.

HIPÓTESIS

El investigador puede controlar completamente su entorno de análisis cuando crea herramientas propias, incluyendo conjuntos de datos, y tiene elementos para elegir otros recursos disponibles al trabajar en reconocimiento facial por computadora. En este trabajo mostramos cómo hacerlo³.

Hipótesis

El investigador puede controlar completamente su entorno de análisis cuando crea herramientas propias, incluyendo conjuntos de datos, y tiene elementos para elegir otros recursos disponibles al trabajar en reconocimiento facial por computadora. En este trabajo mostramos cómo hacerlo.

³Una vez aprobada esta tesis,haremos disponible el material: código, imágenes, textos traducidos, requiriendo un periodo de curación del mismo, en <https://github.com/germaKonda>

OBJETIVO GENERAL

Utilizar tres técnicas que al funcionar unidas, permiten desarrollar aplicaciones que mejoran la velocidad de respuesta⁴ en reconocimiento facial, enfocándonos en equipos de bajo rendimiento.

Objetivo general
Utilizar tres técnicas que al funcionar unidas, permiten desarrollar aplicaciones que mejoran la velocidad de respuesta en reconocimiento facial, enfocándonos en equipos de bajo rendimiento.

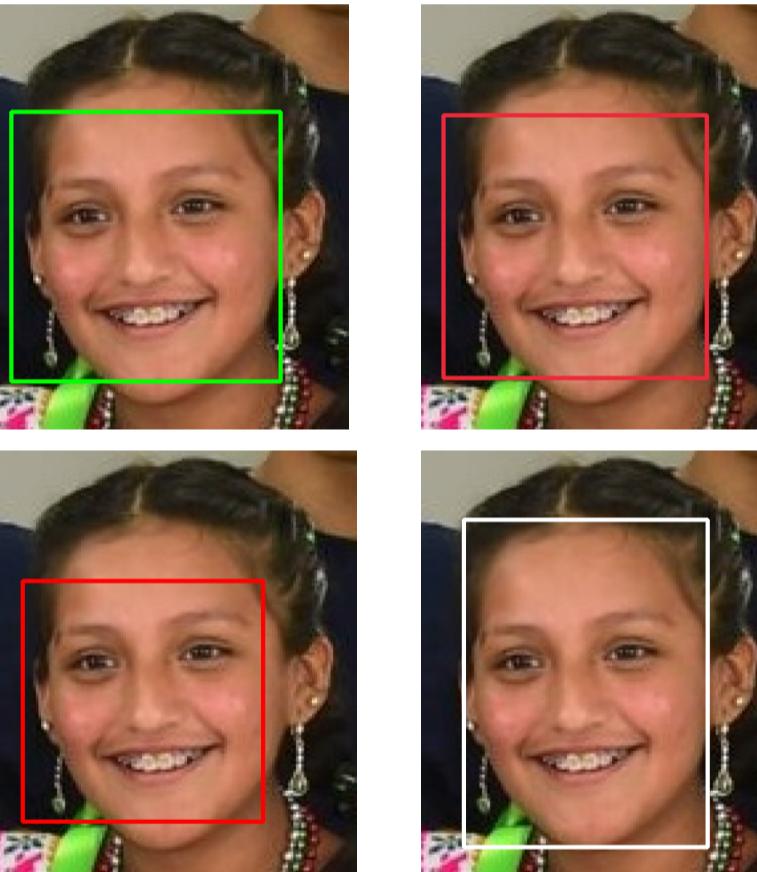
⁴Respecto a cualquier aplicación que realice menos de 6 estimaciones por segundo.

OBJETIVOS ESPECÍFICOS

Objetivos específicos

OBJETIVO 1

Mejor detector facial en eficiencia
y velocidad,

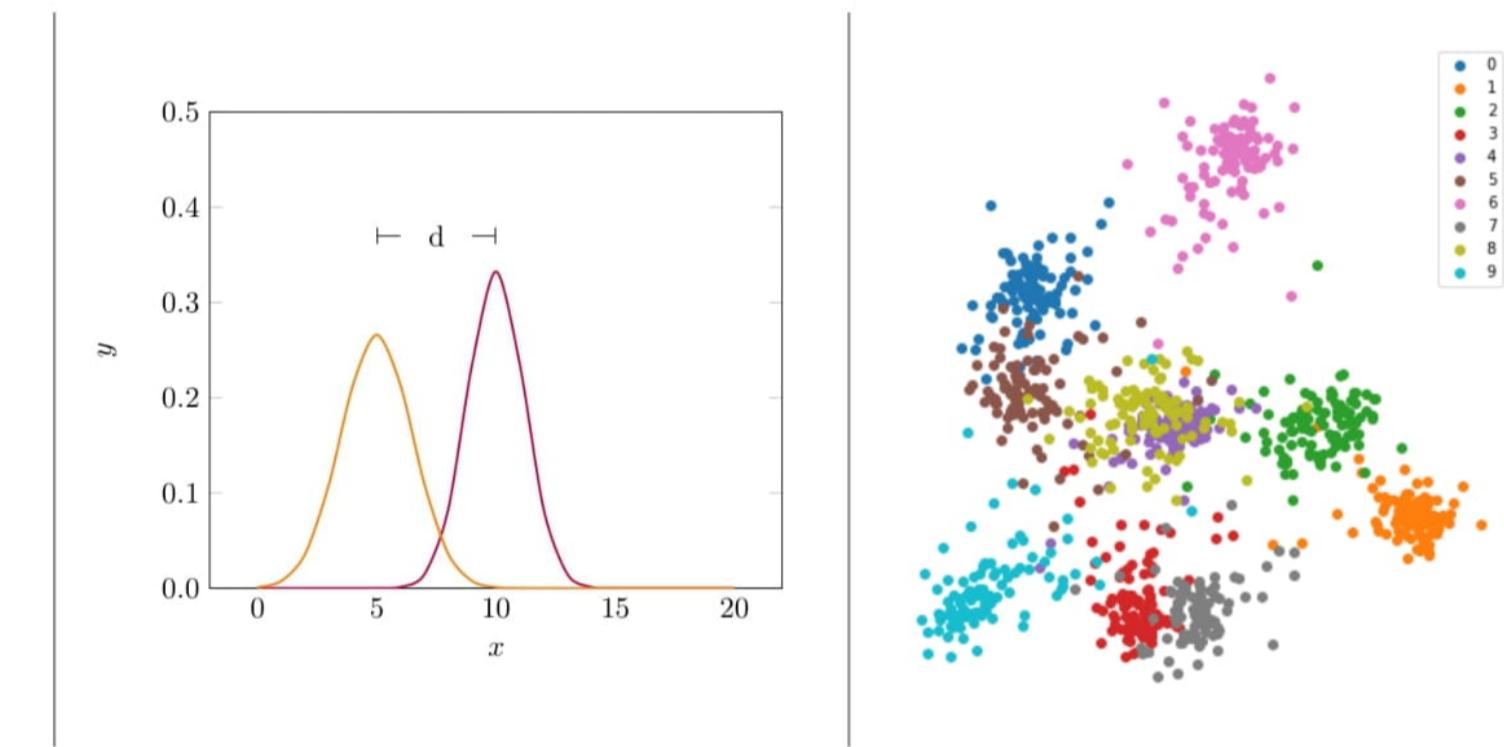
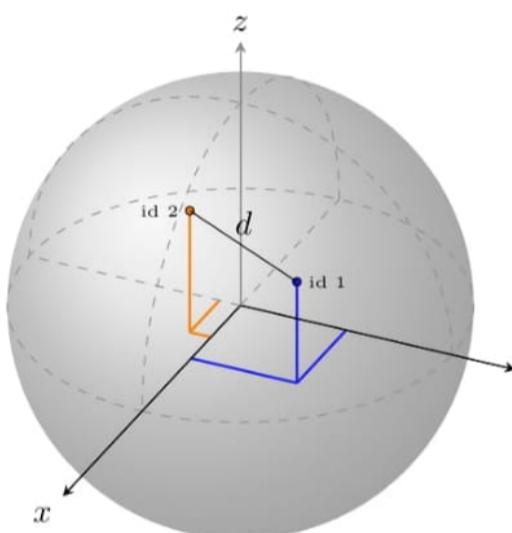


Objetivo 1

Encontrar el mejor **detector facial** en términos de eficiencia y velocidad, de los cuatro analizados aquí.

OBJETIVO 2

Implementar funciones de pérdida en una red neuronal

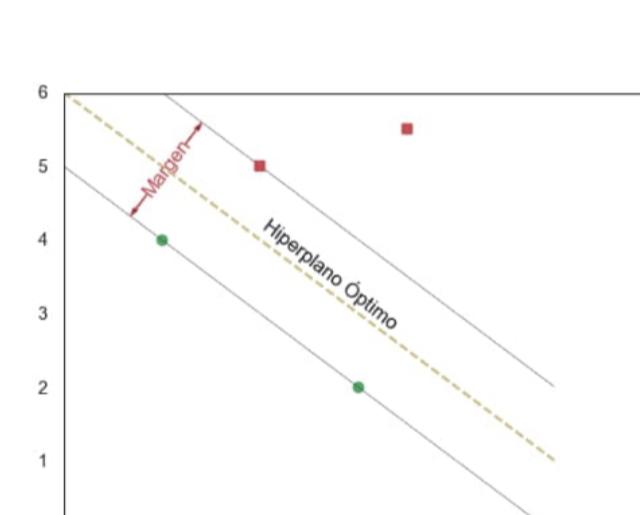
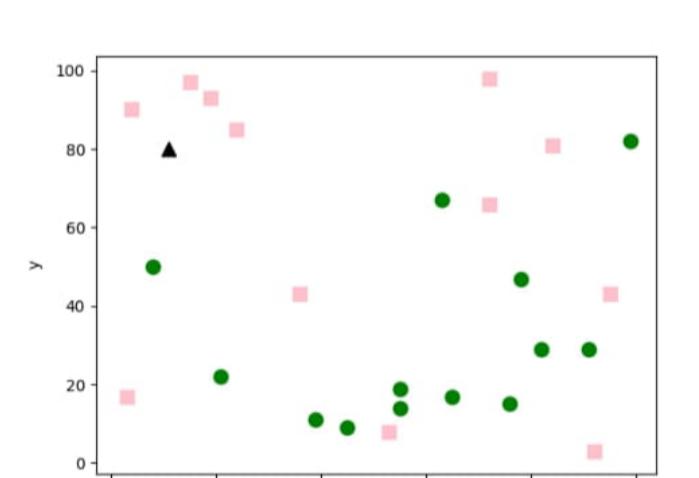
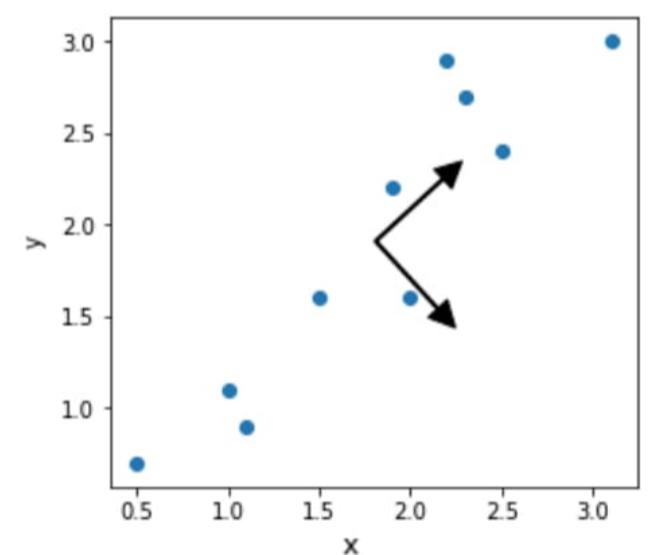


Objetivo 2

Implementar una red neuronal con tres diferentes **funciones de pérdida** y ver cuál es la que obtiene mejores resultados en el proceso de convergencia cuando se trabaja en imágenes conteniendo la cara de una persona.

OBJETIVO 3

Clasificador de representaciones vectoriales faciales.

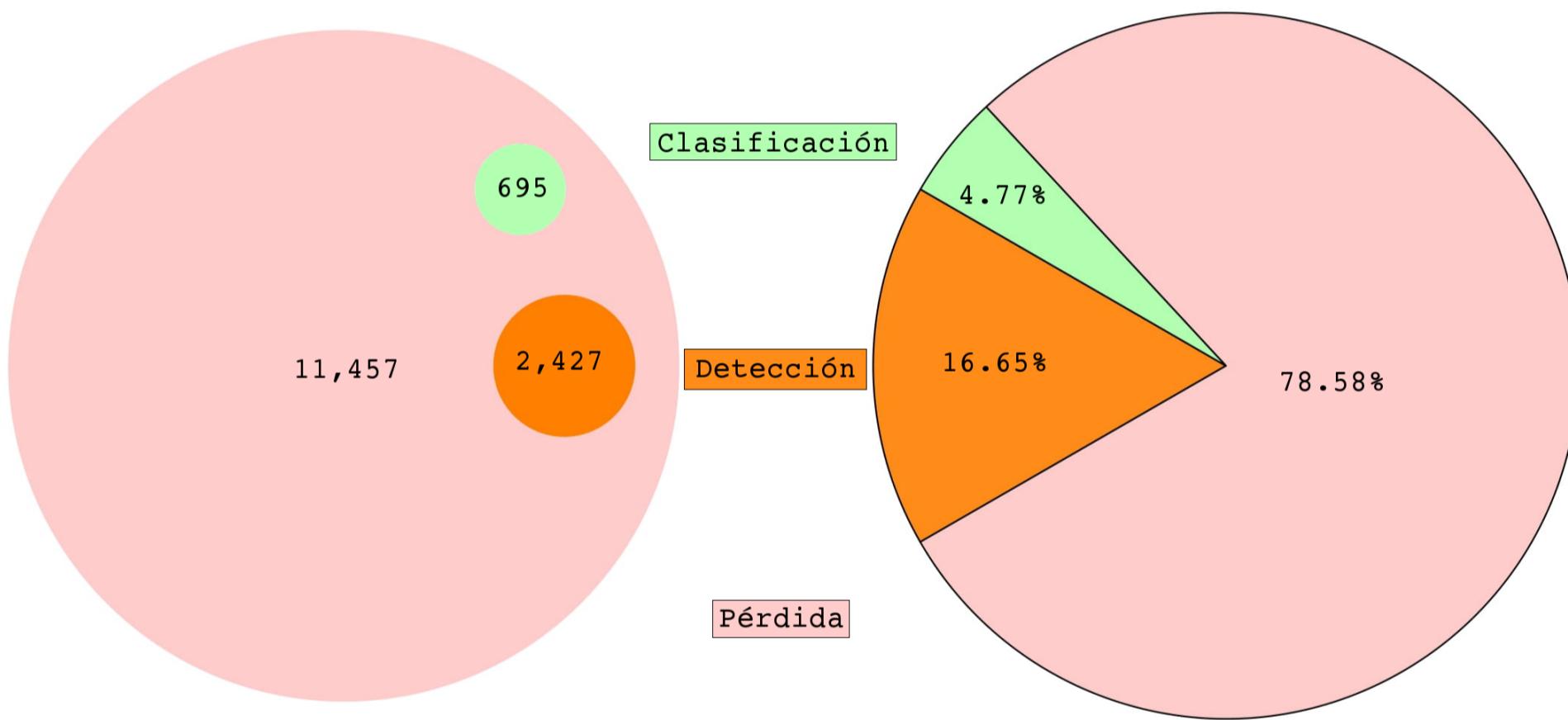


Objetivo 3

Descubrir el mejor algoritmo de **clasificación** de representaciones vectoriales faciales de los tres analizados aquí.

OBJETIVO 4

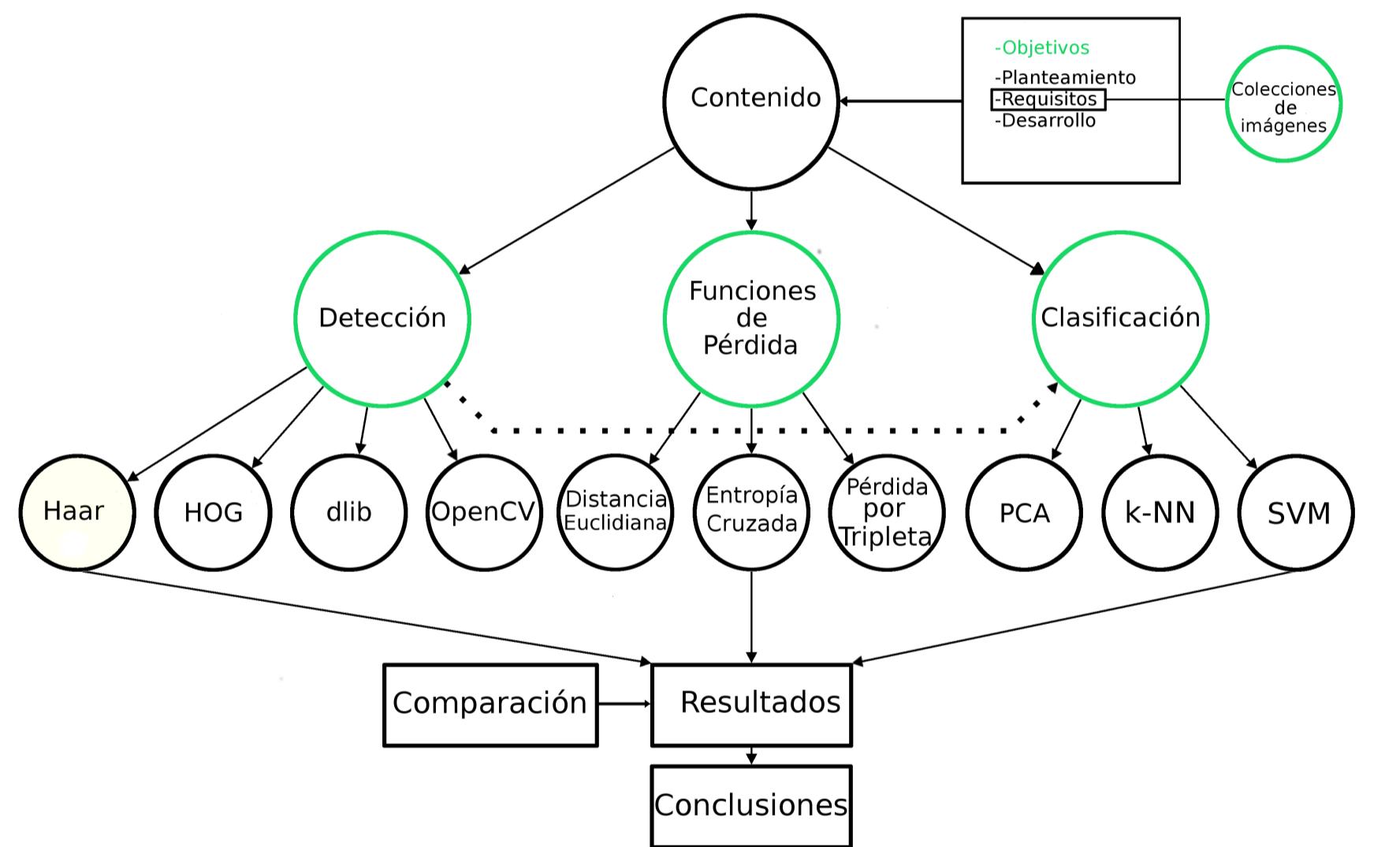
Hacer colecciones de imágenes y herramientas propias.



Objetivo 4

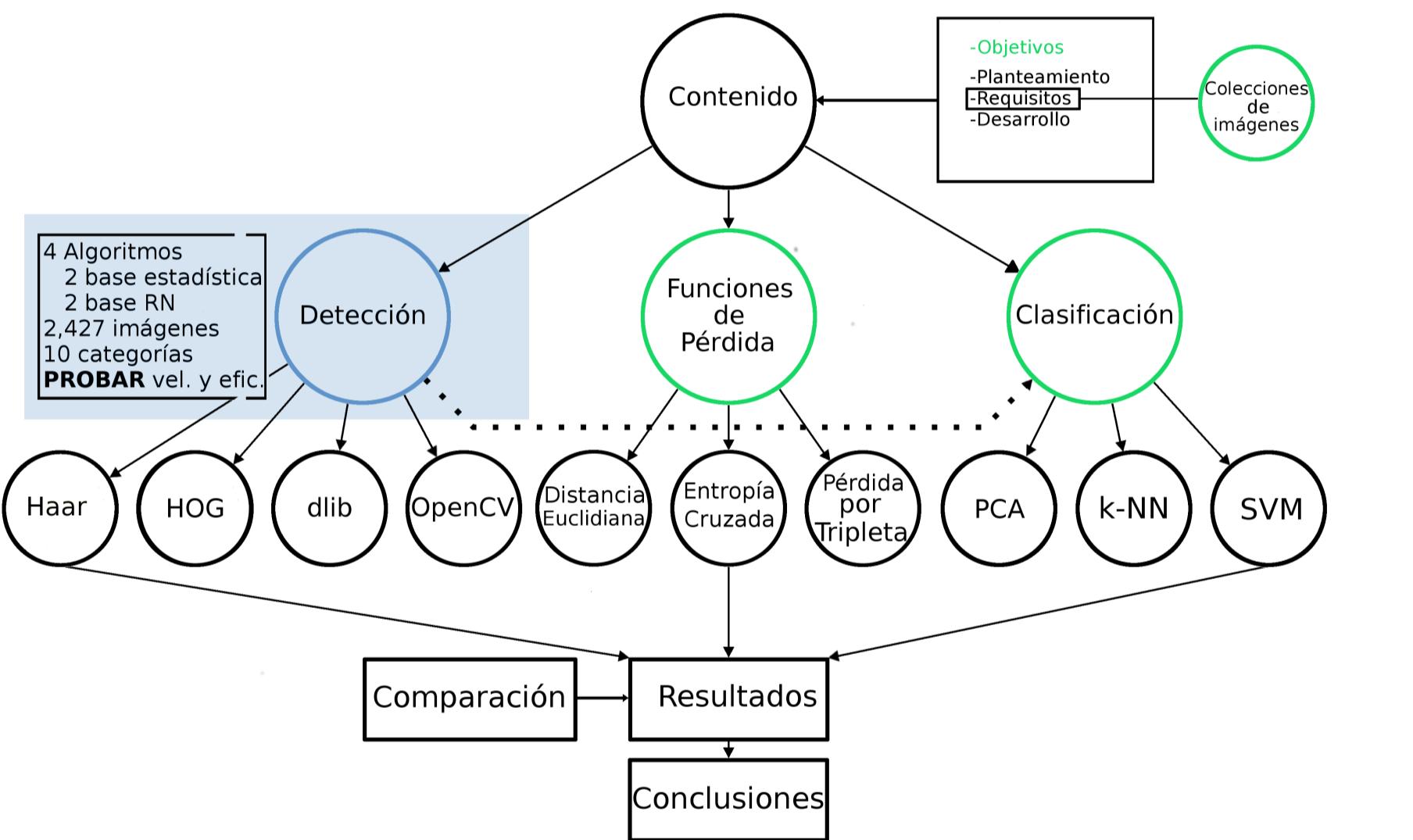
Mostrar que se pueden hacer **colecciones de imágenes y herramientas propias** para nuevas investigaciones y para reproducir resultados de otros estudios.

DIAGRAMA GENERAL DE LA PRESENTACIÓN



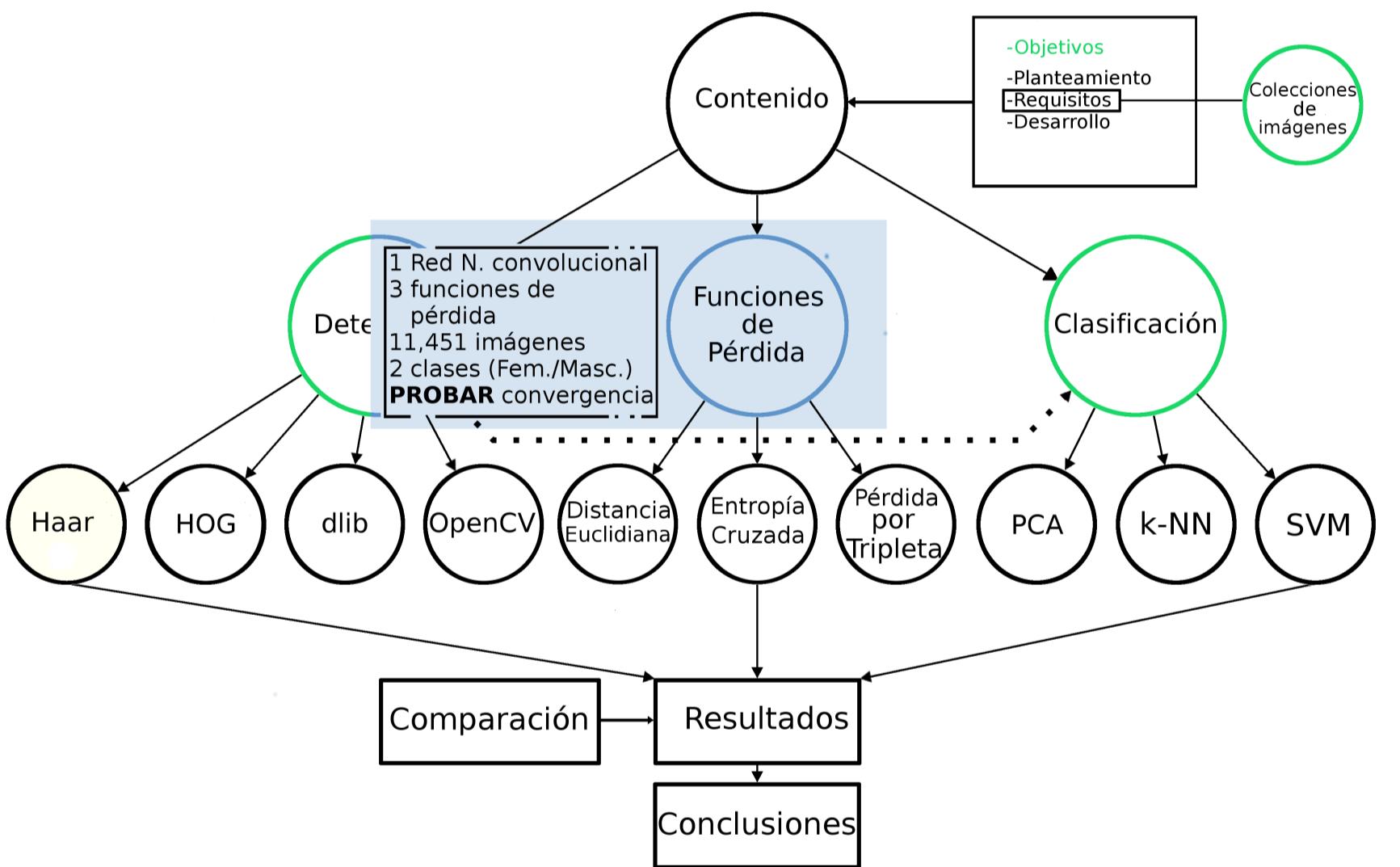
El diagrama presenta el contenido de este trabajo. Cada uno de los círculos verdes representa un objetivo.

DIAGRAMA — OBJETIVO I



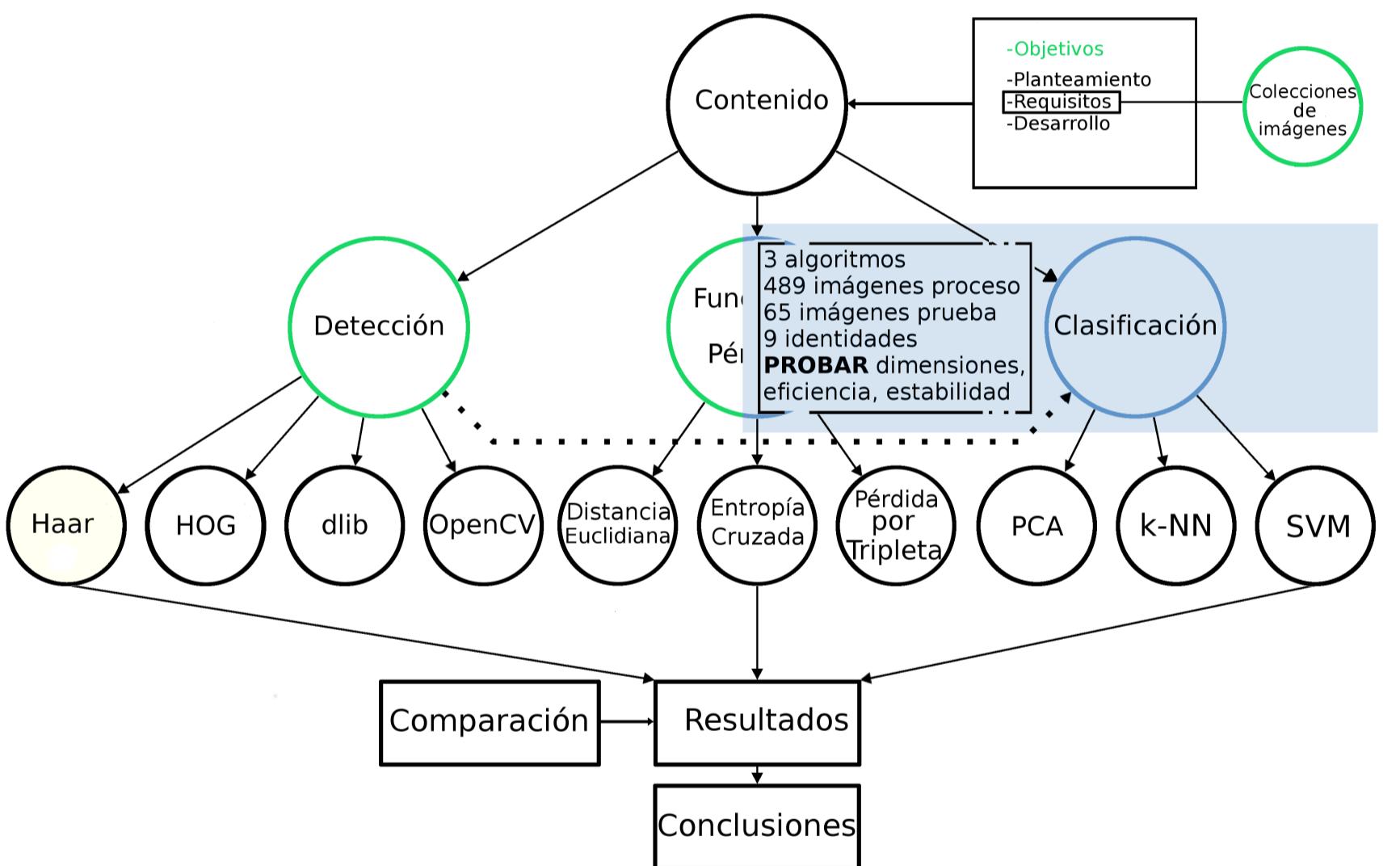
En el objetivo i, se quiere encontrar el mejor detector de caras con base en los factores de velocidad y eficiencia.

DIAGRAMA — OBJETIVO II



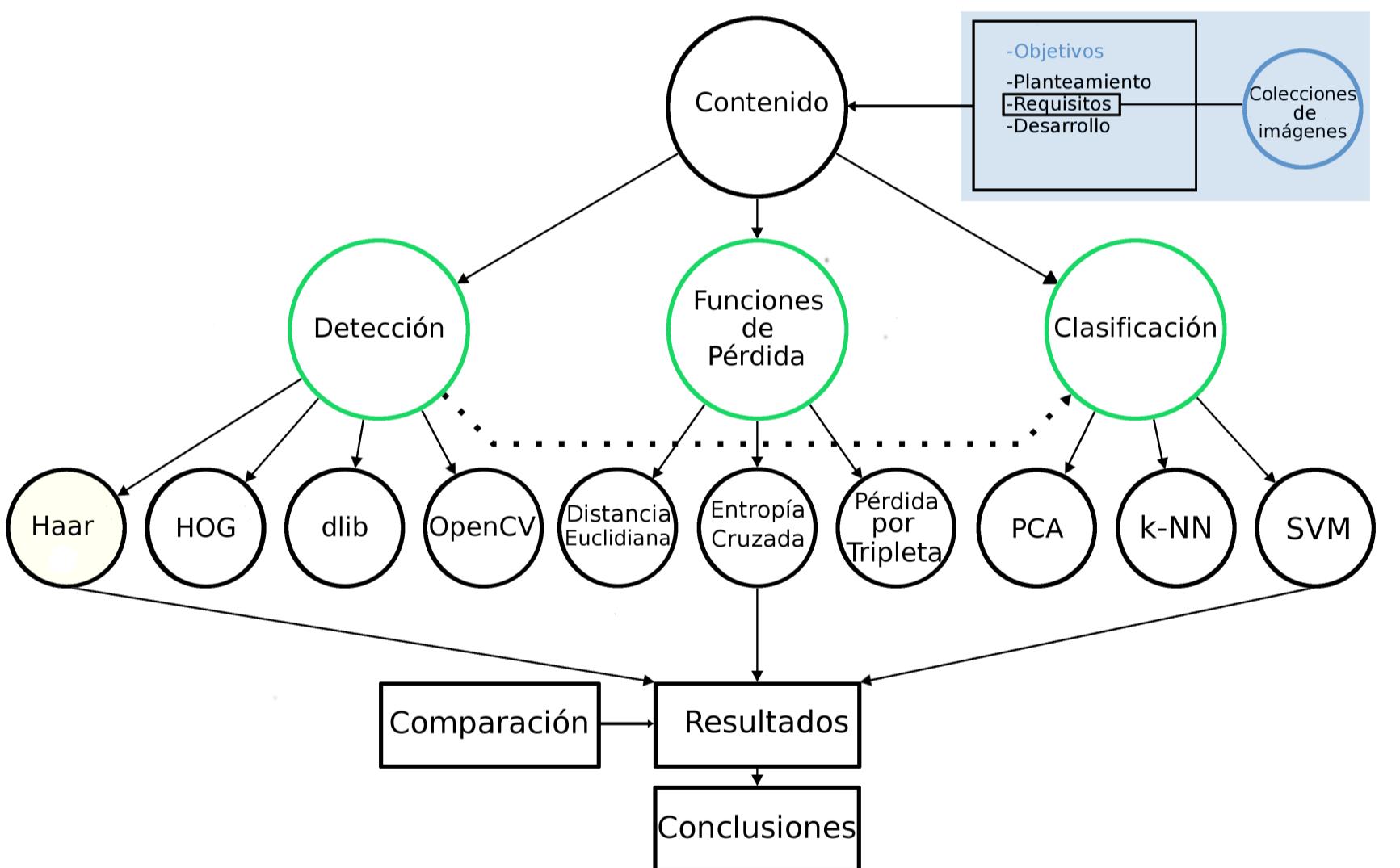
En el objetivo ii, que quiere encontrar aquella función de pérdida que mejor convergencia presente, en términos de menor pérdida.

DIAGRAMA — OBJETIVO III



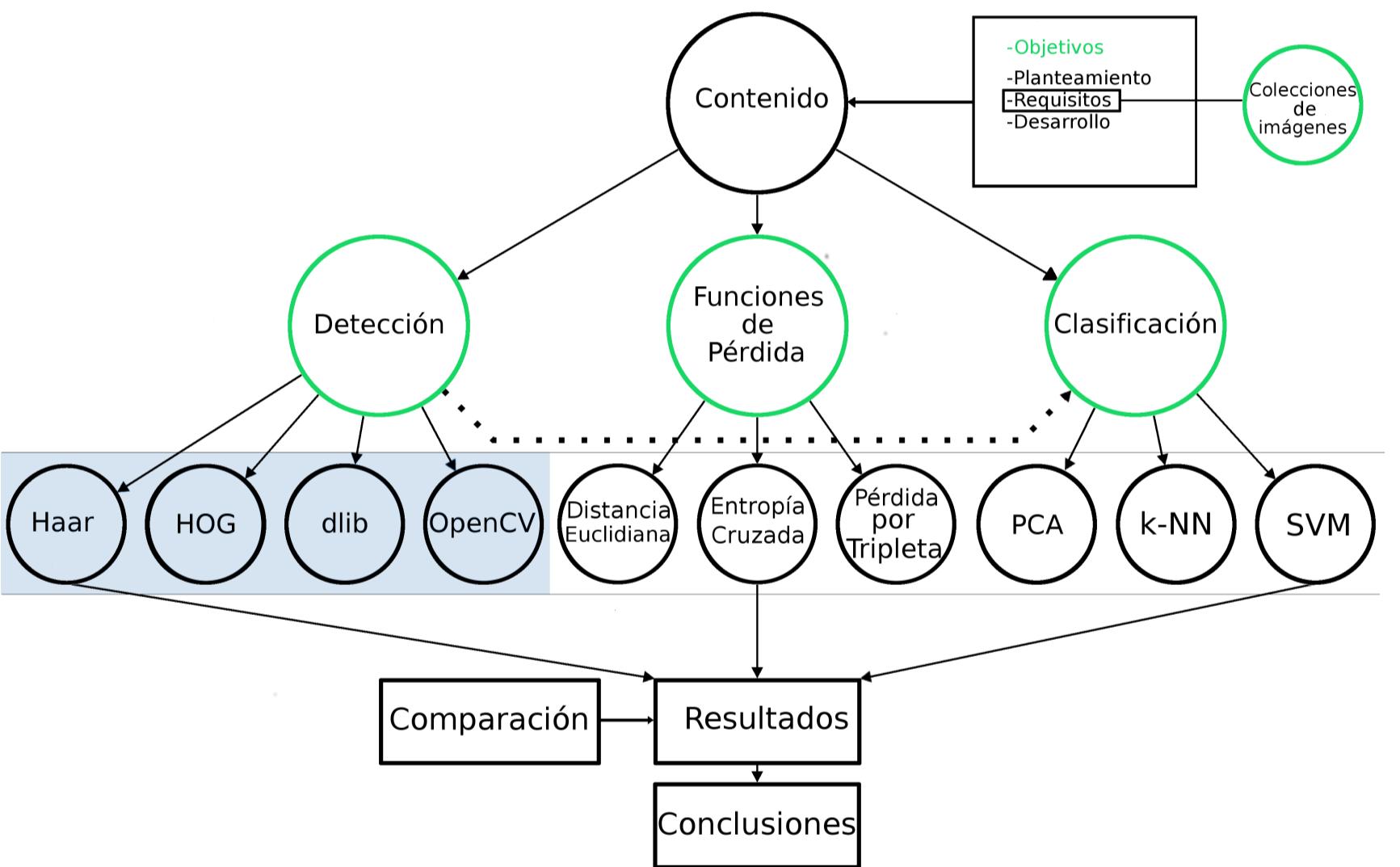
En el objetivo iii se trabaja en encontrar el mejor algoritmo de clasificación en términos de precisión, dimensiones espaciales y respuesta incremental.

DIAGRAMA — OBJETIVO IV



El objetivo iv es relevante para el resto del presente estudio debido a que al cumplir este objetivo se obtienen las colecciones de imágenes que se emplean en este trabajo.

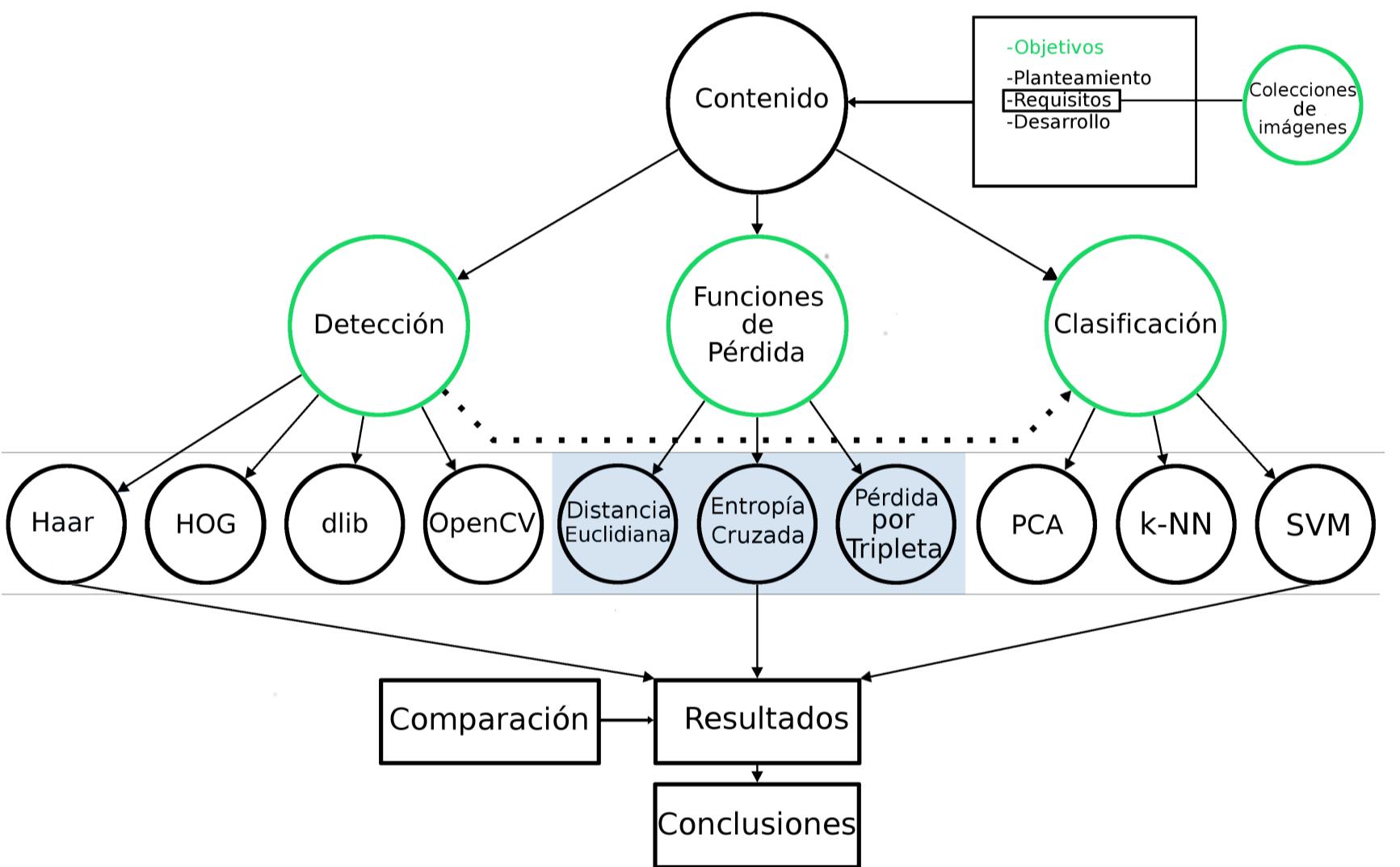
DIAGRAMA — OBJETIVO I — DETECTORES



Detectores

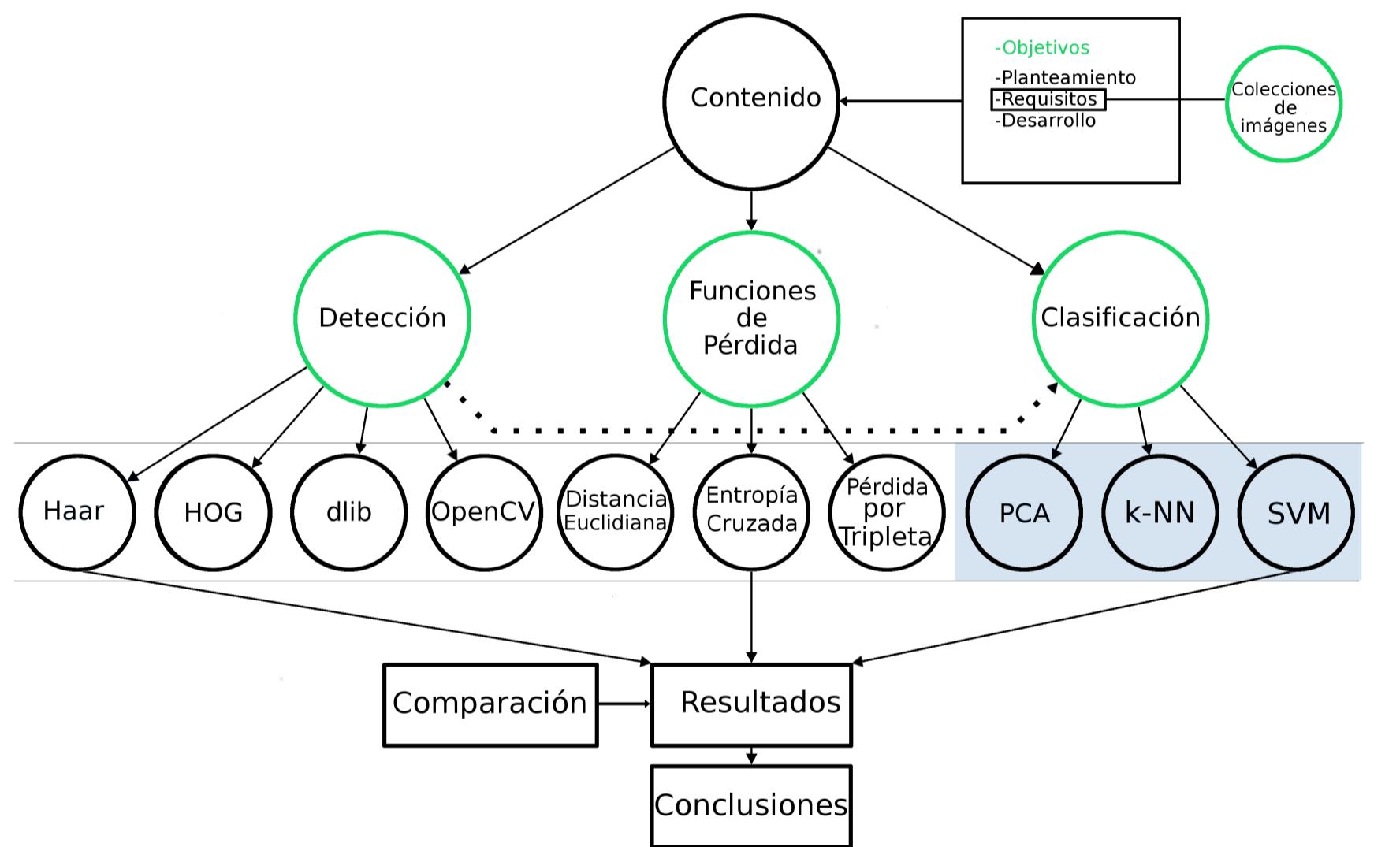
Para detección se estudian 4 algoritmos. Se empieza con cascada de clasificadores débiles e histogramas de gradientes orientados. Los otros dos algoritmos son detectores basados en redes neuronales como parte de las bibliotecas de desarrollo dlib y openCV.

DIAGRAMA — OBJETIVO II FUNCIONES DE PÉRDIDA



Funciones de pérdida
Se analizan tres funciones de pérdida:
Distancia euclidiana, Entropía Cruzada y
Pérdida por tripleta.

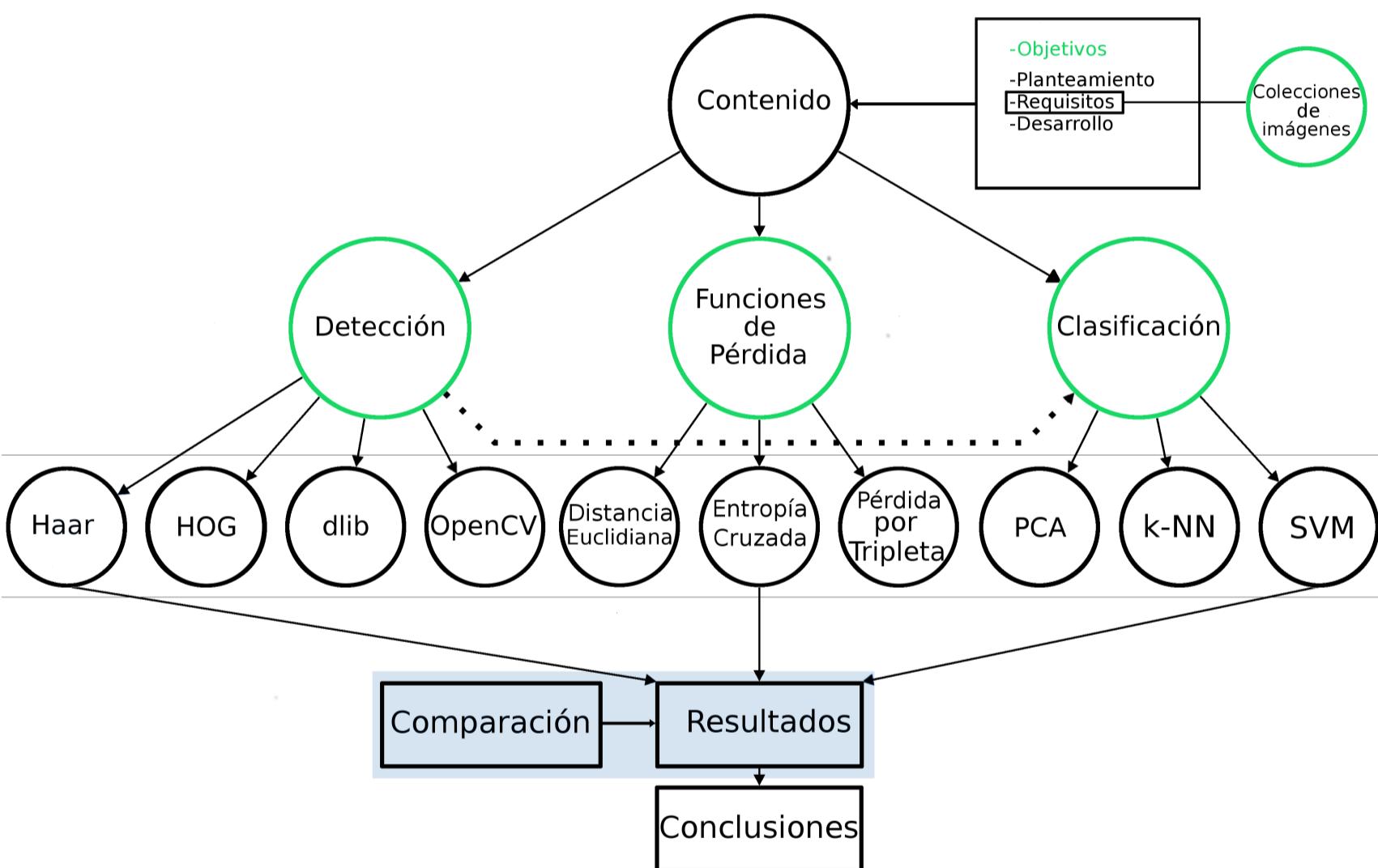
DIAGRAMA — OBJETIVO III — CLASIFICADORES



Clasificadores

Los tres algoritmos de clasificación a analizar son: PCA o análisis de componentes principales, k vecinos más cercanos y SVM o maquinas de sopote vectorial.

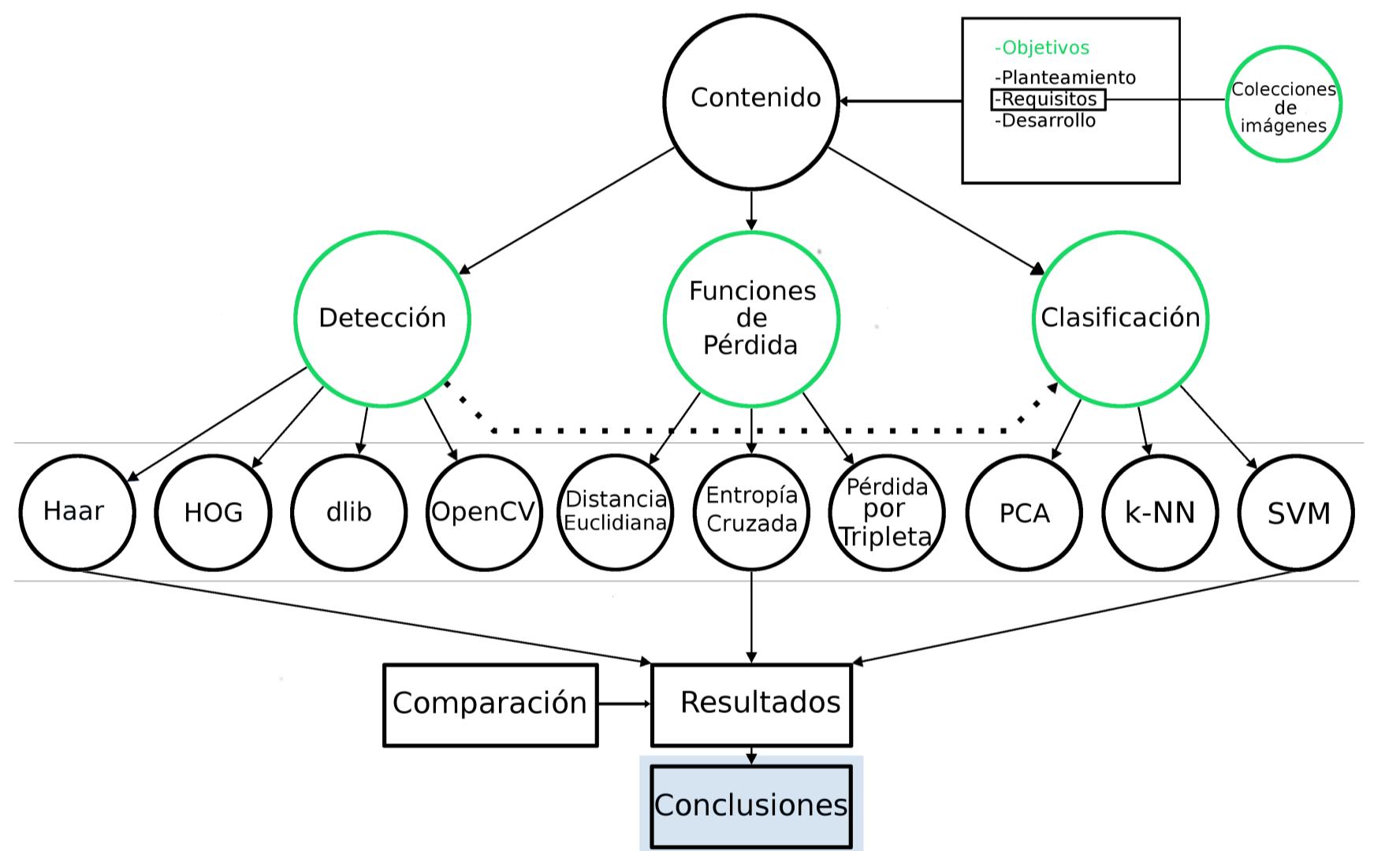
DIAGRAMA — RESULTADOS



Resultados

Posteriormente se presenta una sección de resultados del análisis de cada familia de algoritmos. Se muestra la interfaz de dos programas, uno de escritorio y otro de consola que sirven para hacer dos comparaciones. Una respecto a una solución académica y otra respecto a una aplicación comercial.

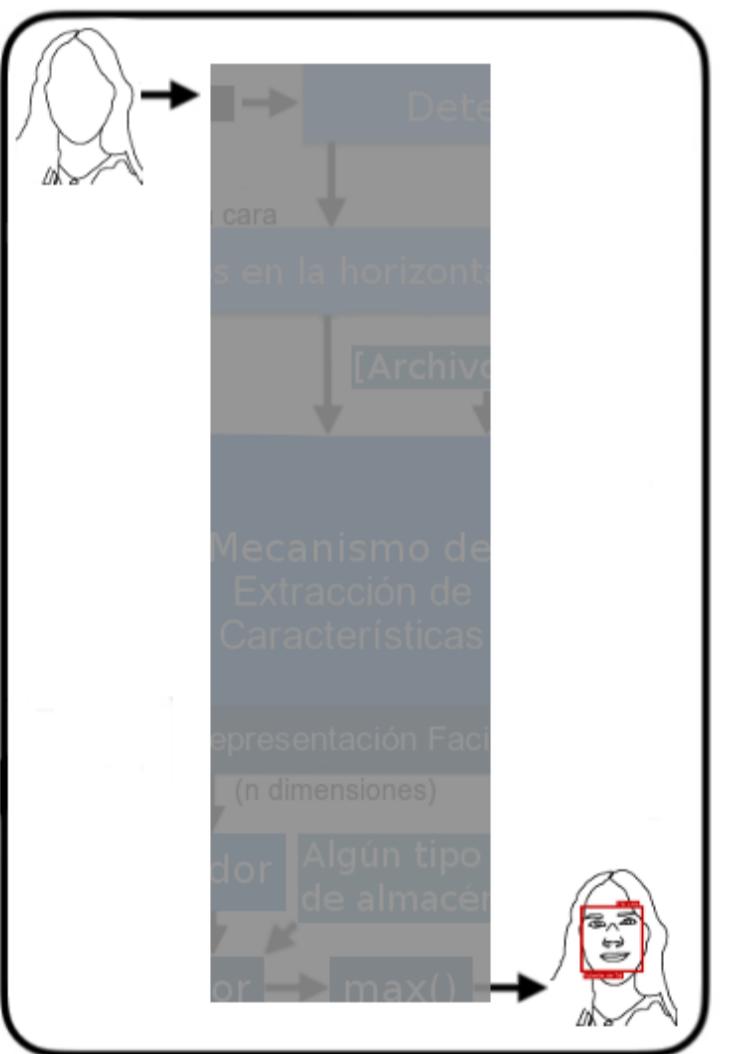
DIAGRAMA — CONCLUSIONES



Conclusiones

Al final se presentan las conclusiones y las aportaciones de este trabajo.

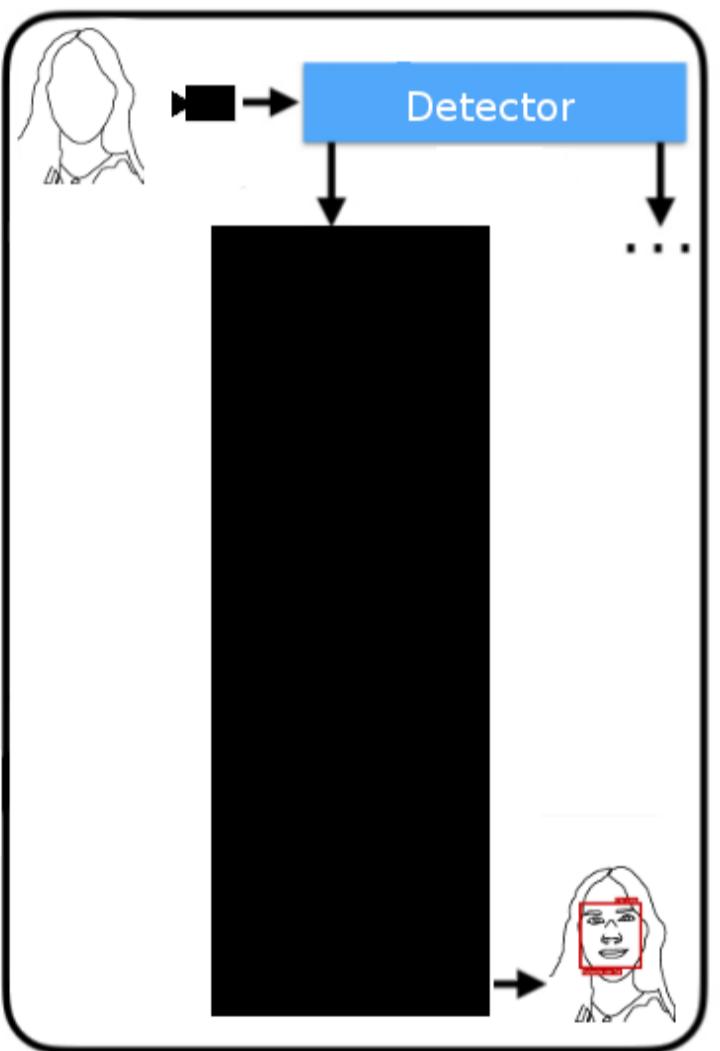
¿CÓMO RESOLVER EL PROBLEMA PLANTEADO?



¿Cómo resolver el problema planteado?

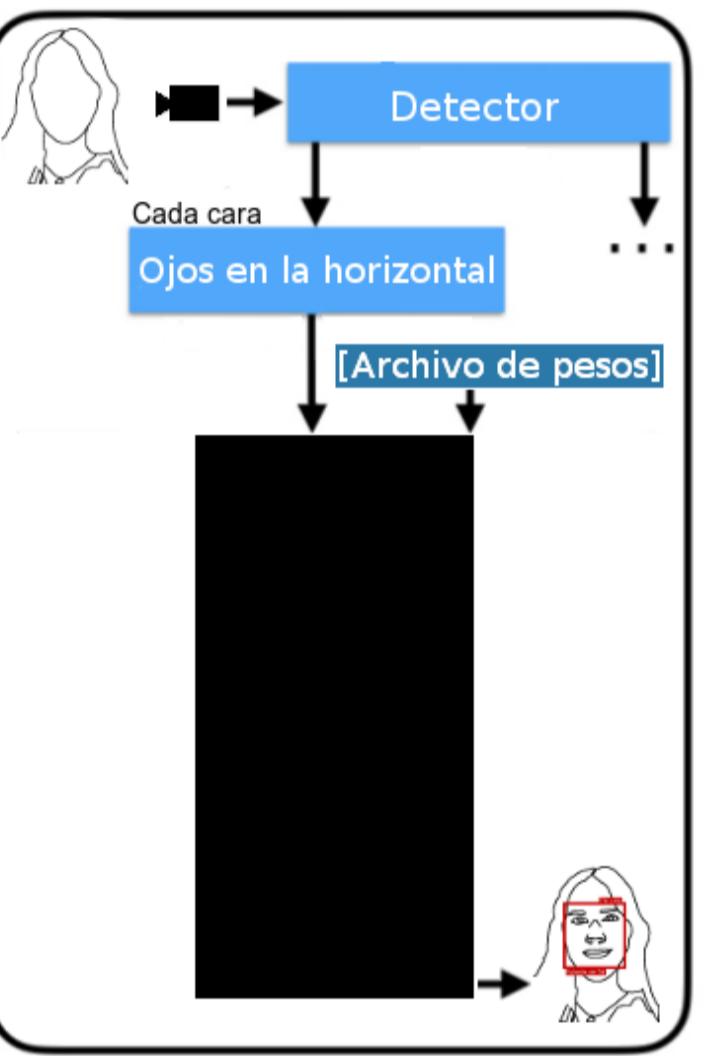
A continuación iremos develando los pasos necesarios para resolver el problema planteado.

DEFINICIÓN DEL PROBLEMA — DETECCIÓN



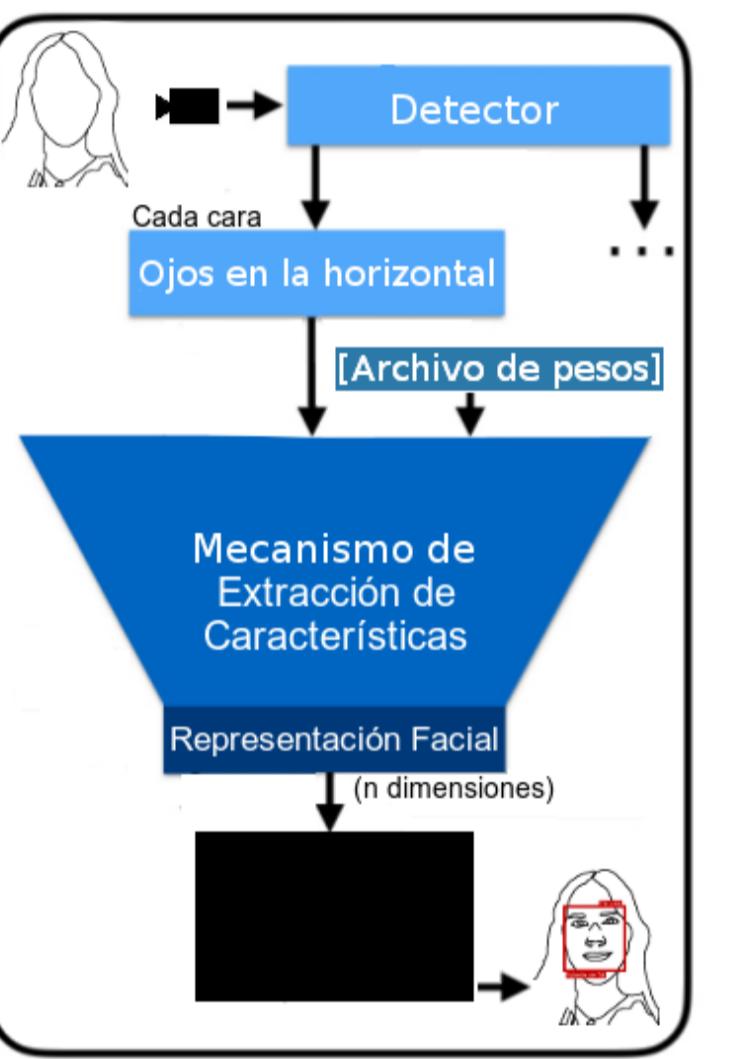
Primero, la señal de una cámara pasa a un algoritmo de detección como los que analizamos aquí. El algoritmo localiza un recuadro que cierra la cara...

DEFINICIÓN DEL PROBLEMA — CONDICIONES



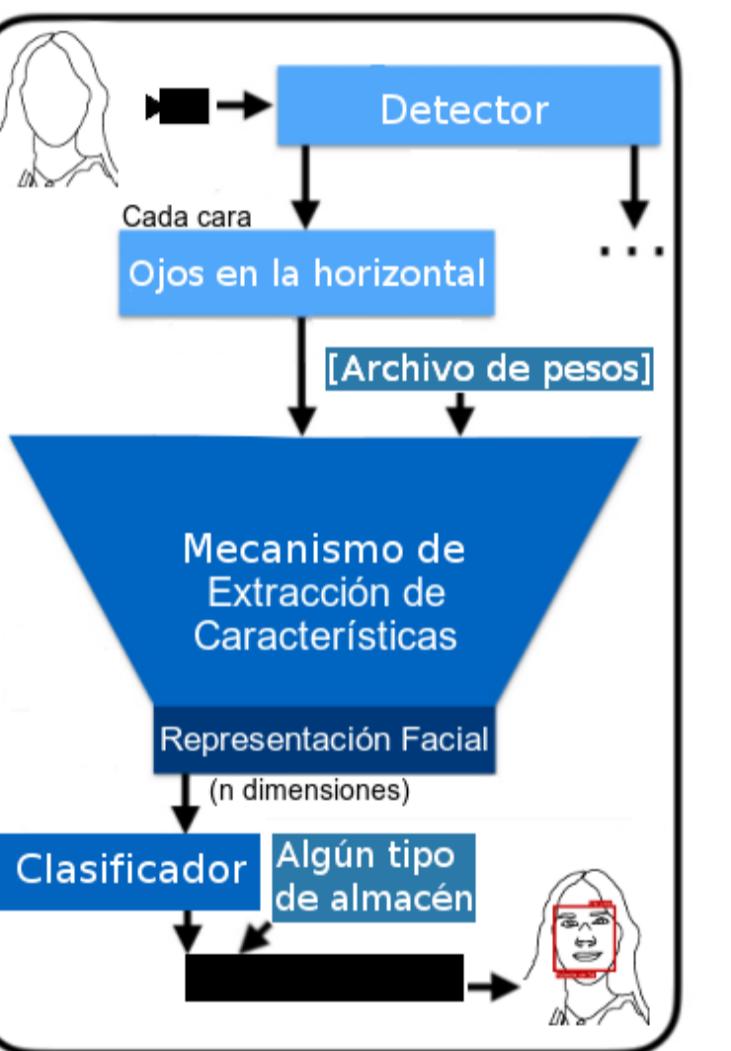
...y la pasa a una sección que opcionalmente puede establecer un criterio tal como que la cara mantenga los ojos en la horizontal.

DEFINICIÓN DEL PROBLEMA — EXTRACCIÓN



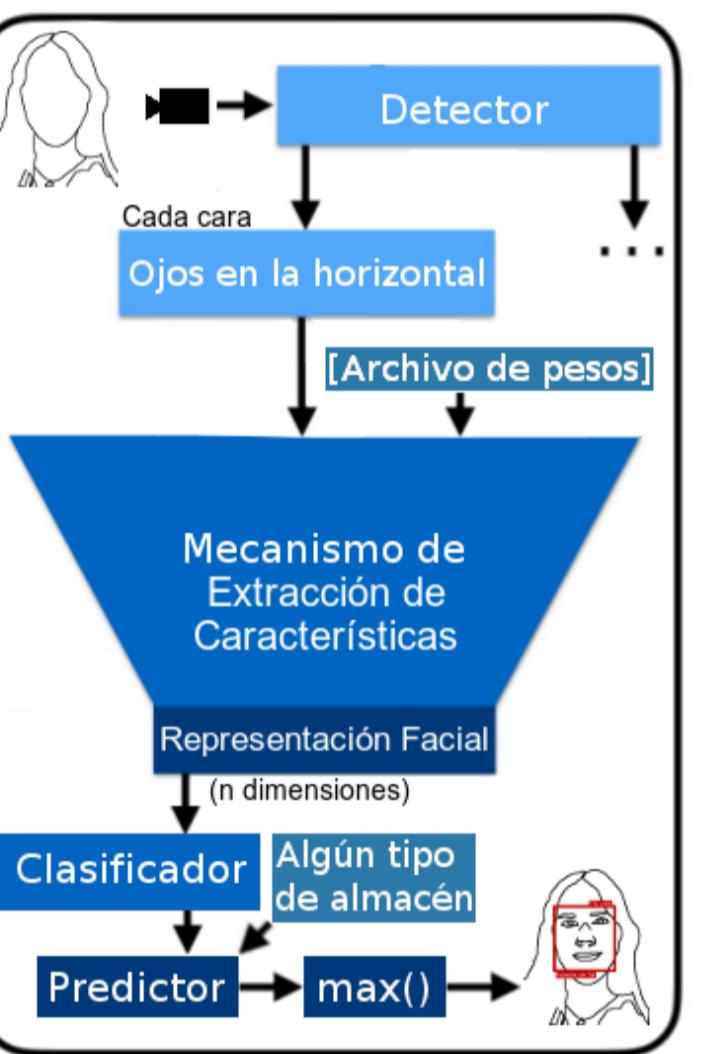
Este resultado pasa a un mecanismo de extracción de características, como alguno de los tres que aquí se analizan, y que opcionalmente puede auxiliarse de un archivo de pesos, como en el caso de redes neuronales. El resultado es una representación facial en forma de un vector de n dimensiones.

DEFINICIÓN DEL PROBLEMA — CLASIFICACIÓN



Dicho vector es alimentado a un algoritmo de clasificación y, junto con algún repositorio donde están almacenadas las representaciones de identidades previamente registradas...

DEFINICIÓN DEL PROBLEMA — CONCLUSIÓN



...se pasa a un módulo de estimación que entrega un arreglo de probabilidades. De estas se toma la más alta.
Se aprovecha para recordar que el reconocimiento facial no es un proceso determinístico sino probabilístico. De allí que tomemos el valor máximo de ese arreglo y lo presentemos como resultado más probable.

EQUIPO UTILIZADO

Se emplearon 4 computadoras en diferentes tareas

Vaio Centrino con 4GB RAM @Ubuntu 16.04
Procesó la mayor parte de este trabajo



Mac Book Pro core 2 duo 4GB RAM @Ubuntu 20.04
Ejecución de pruebas en lote



Mac Mini core i5 8GB RAM
Ejecución de red cuando la Vaio se atoró



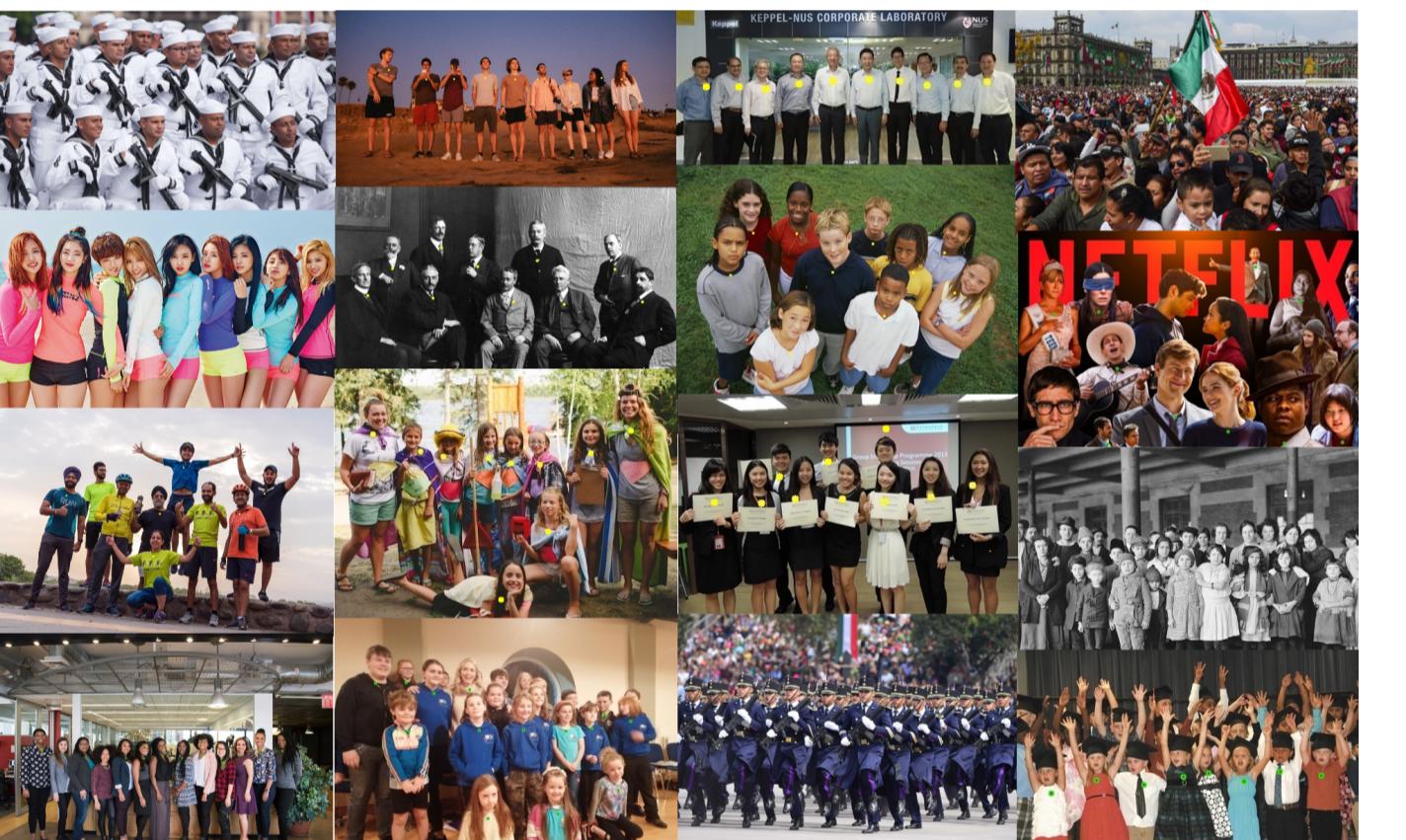
Raspberry Pi modelo B+
Probar la ejecución del reconocedor



Se emplearon estas cuatro computadoras.
La mas 'avanzada' de ellas es la que tiene el
procesador core i5 pero sólo se uso al
principio del proceso de convergencia con
las funciones de pérdida analizadas.

COLECCIONES DE IMÁGENES

Requerimos imágenes sobre las cuales detectar caras



Colecciones de imágenes

Para contar con las colecciones necesarias se recabó una cantidad de imágenes grupales, como las aquí mostradas. Algunas fueron extraídas de revistas en formato .pdf.

PROGRAMAS PARA CREAR COLECCIONES (1/2)

Agrupación manual, captura de flujo de video y recorte manual



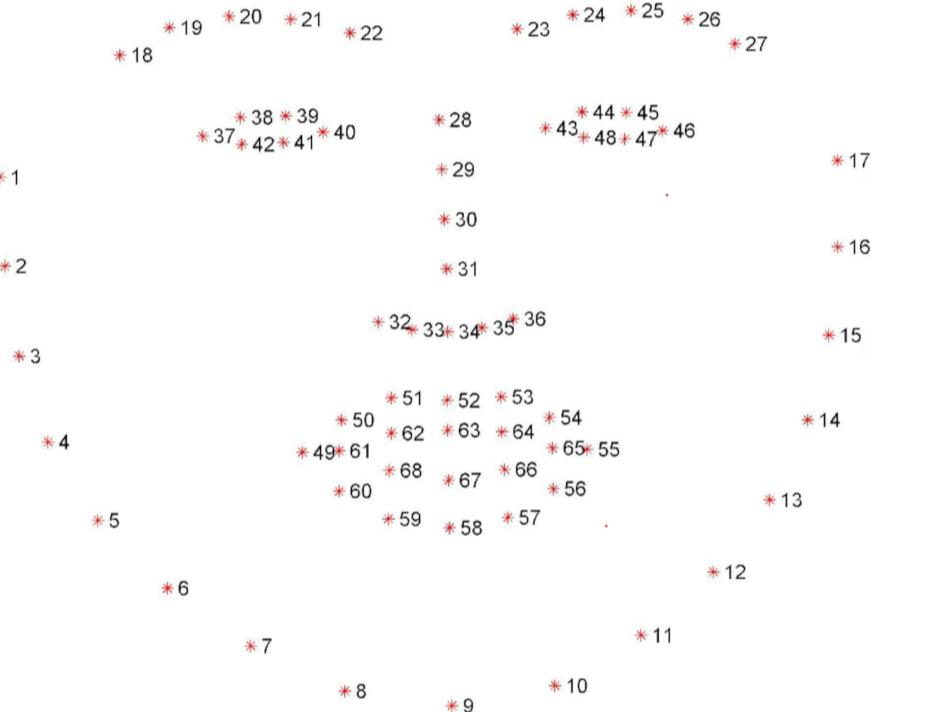
Se muestran **cuatro** de los programas codificados. Los dos **primeros** agrupan caras segun una categoria arbitraria de pose. Las imágenes se acomodan en carpetas que serviran para el resto del estudio.

El **tercer** programa captura cuadros a partir de un flujo de video y permite desechar imágenes de baja calidad. En este caso hay una cara de perfil eliminada.

El **programa** de abajo muestra una foto grupal sobre la que se enmarcó manualmente cada cara con el potencial de ser detectada. El recuadro rojo es un ejemplo de cara seleccionada y almacenada.

PROGRAMAS PARA CREAR COLECCIONES (2/2A)

Detección automática de caras inclinadas⁵

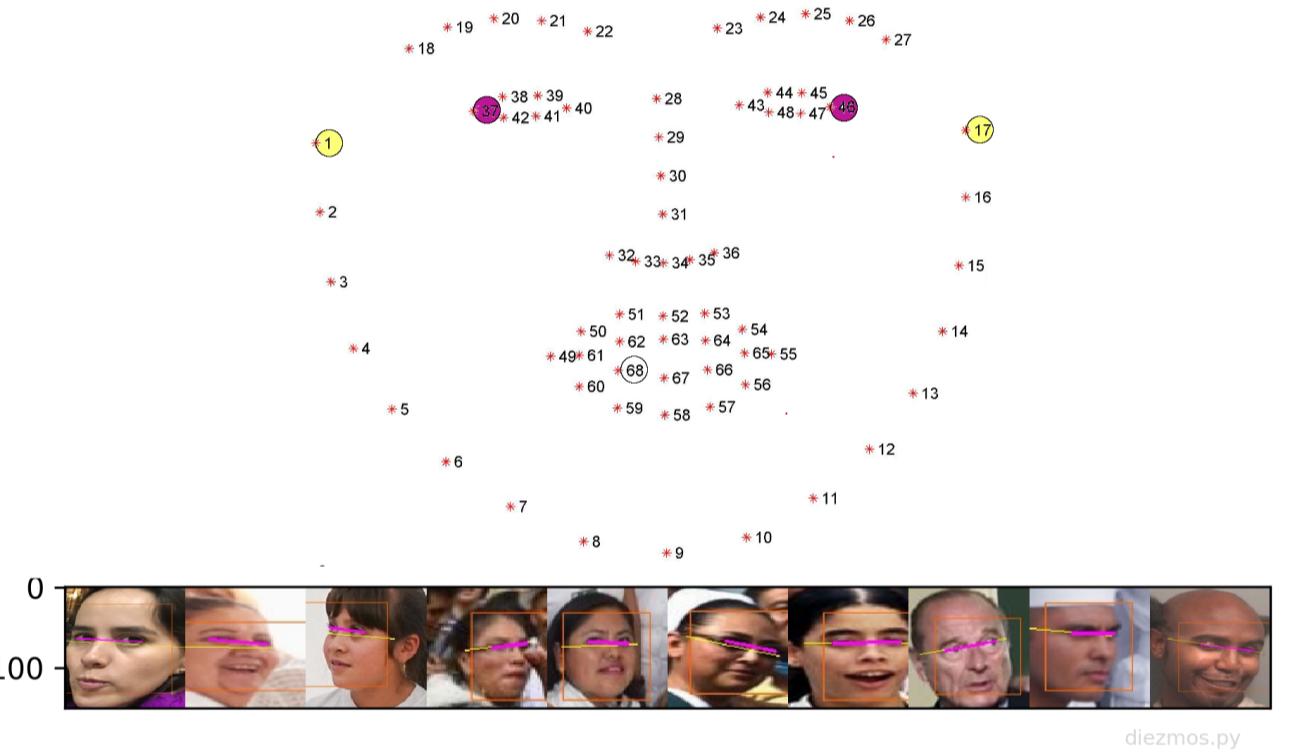


Otro programa de agrupamiento automático de imágenes utiliza el modelo ibug con 68 marcadores. El modelo es del colegio imperial de Inglaterra.

⁵Con ayuda del modelo i-bug: <https://ibug.doc.ic.ac.uk/> [ibug=ingelligent behaviour understanding group]

PROGRAMAS PARA CREAR COLECCIONES (2/2B)

Detección automática de caras inclinadas⁶



⁶Con ayuda del modelo i-bug: <https://ibug.doc.ic.ac.uk/> [ibug=ingelligent behaviour understanding group]

En el modelo se ven los pares de puntos 1-17 y 37-46. El mosaico de abajo muestra líneas magentas y líneas amarillas superpuestas a la caras. La geometría de estas líneas permite hacer la agrupación automática. Con diferentes puntos del modelo se pueden encimar máscaras a las caras como lo hace Snapchat

COLECCIONES CREADAS

Tres colecciones, 14.5k imágenes, 21 clases

Colección	Imágenes	Clases
Detección	2,427	10
Pérdida	11,451	2
Clasificación	622	9
Total	14,530	21

En la tabla observamos los resultados de procesar varios miles de imágenes grupales: se obtuvieron 14 mil 500 imágenes individuales y 21 clases. Se pueden ver las particiones correspondientes para cada familia de algoritmos.

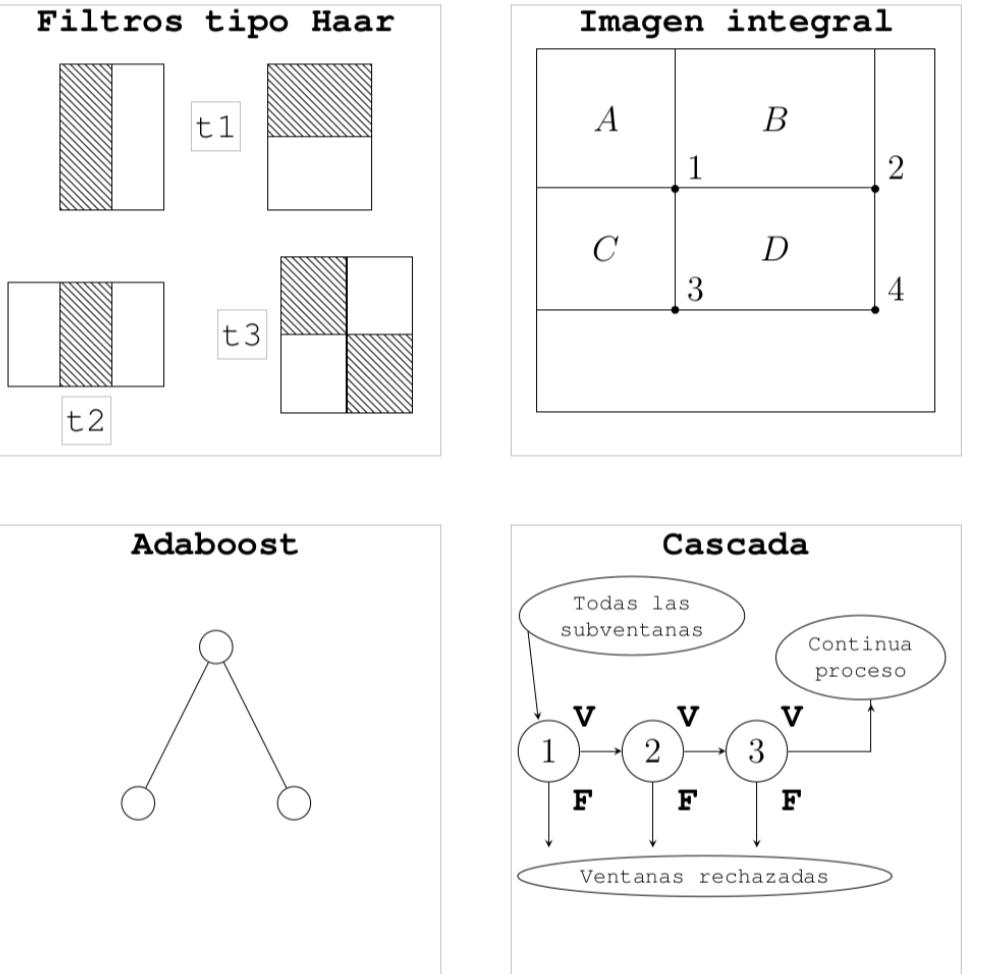
DETECCIÓN

Colección de imágenes para detección

4 algoritmos analizados
2,427 imágenes
10 Categorías de pose

Para detección se analizaron 4 algoritmos y se usaron 2,427 imágenes repartidas en 10 categorías de pose facial.

DETECCIÓN — HAAR — COMPONENTES BÁSICOS⁷



Estos son los cuatro componentes básicos que permiten la creación de un modelo de detección. No es una red neuronal sino un modelo de regresión.

⁷ Paul Viola and Michael Jones. *Rapid object detection using a boosted cascade of simple features*. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, volume 1, pages I–I. IEEE, 2001.

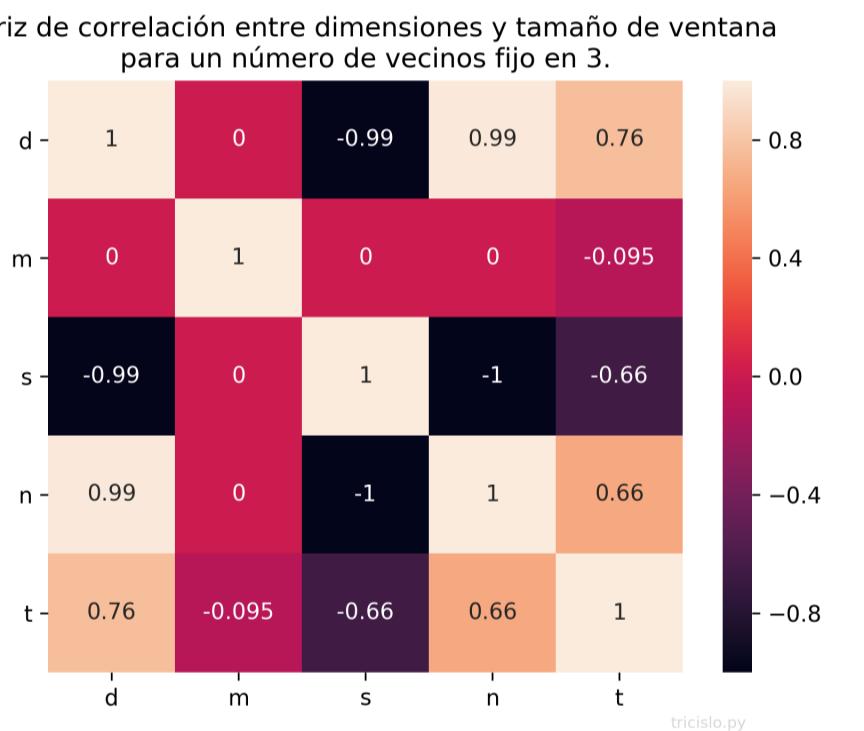
DETECCIÓN — HAAR

Función de OpenCV altamente configurable (7 parámetros)

```
rects = detector.detectMultiScale(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), scaleFactor = 1.1, minNeighbors = 3, minSize = (20, 20))
```

Dimensiones*	Ventana mínima	Sí detectadas	No detectadas	Tiempo (s)
TO	10	1,931	496	73
TO	15	1,931	496	48
TO	20	1,931	496	47
100	10	1,491	936	21
100	15	1,491	936	20
100	20	1,491	936	21
300	10	1,083	1,344	63
300	15	1,083	1,344	63
300	20	1,083	1,344	63
500	10	665	1,762	98
500	15	665	1,762	97
500	20	665	1,762	98

*TO = tamaño original.



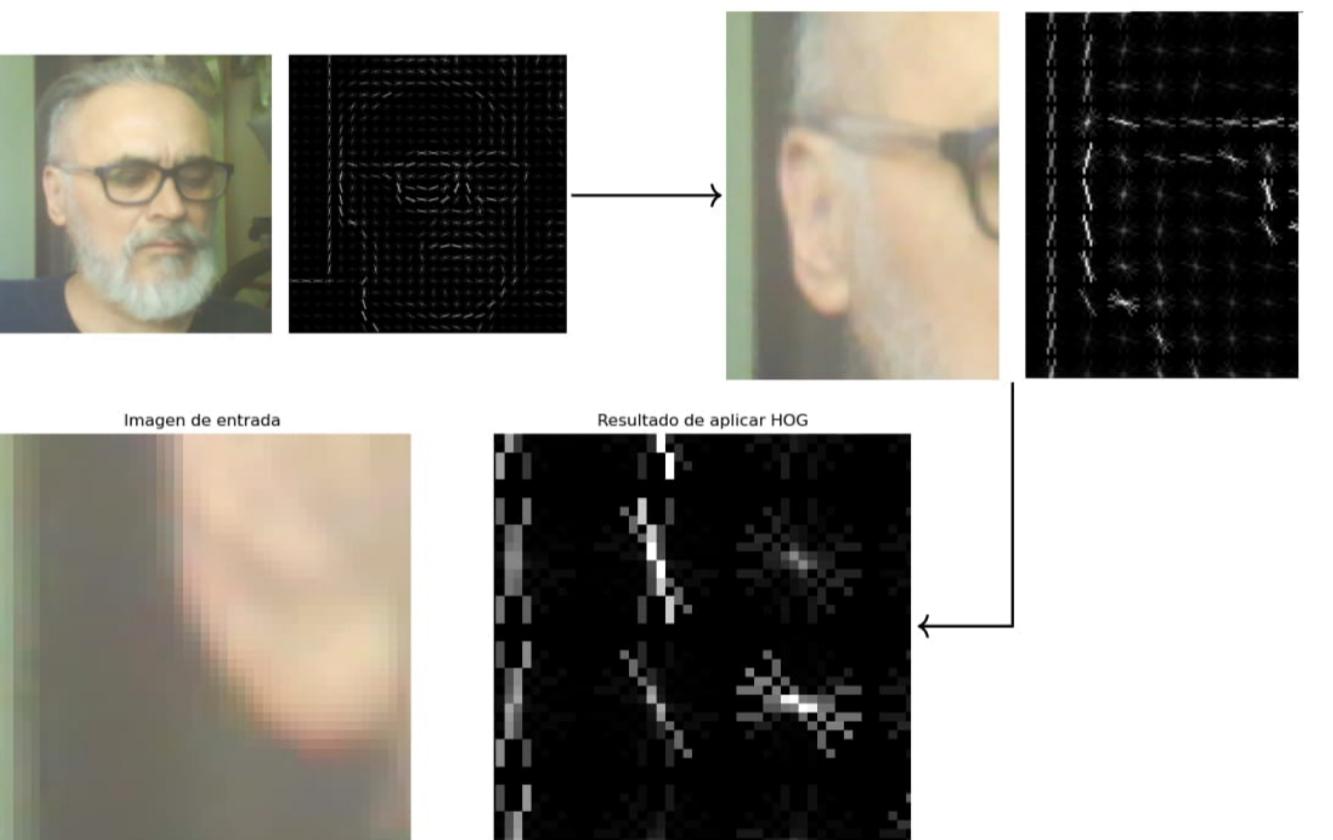
Detector Haar

En primer lugar vemos la función de detección con 5 de sus 7 parámetros. **La tabla** dice que para cada cambio de dimensión de la imagen y dimensión de ventana mínima no hay cambio en el número de detecciones, pero sí en el tiempo requerido. **Se ve** que el mejor resultado es cuando NO se cambian las dimensiones y se usa una ventana mínima de 20. La **gráfica** es una matriz de correlación entre dos variables (dimensiones y ventana mínima) y su efecto en la cantidad de detecciones y el tiempo de proceso.

DETECCIÓN — HOG — ANTECEDENTES

Patente (1986), Robert K. McConnell: *Method of and apparatus for pattern recognition.*

Artículo (2005), Dalal y Triggs: *Histograms of Oriented Gradients for Human Detection.*



```
from skimage.feature import hog
from skimage import data, exposure
import cv2

imagen = cv2.imread('cara384.png')
fd, hog_image = hog(imagen, orientations=8, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1), visualize=True, multichannel=True)
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
```

HOG Empezó con una patente y luego un artículo detonó su empleo. La imagen de abajo muestra la detección del cambio más brusco en intensidad el la parte baja de la oreja

DETECCIÓN – HOG

Función de dlib con sólo un parámetro configurable⁸

```
1 import dlib
2 from skimage import io
3 ...
4     imagen = io.imread('cara384.png')
5     detector = dlib.get_frontal_face_detector()
6     rects = detector(imagen, n) # n es el parámetro (entero entre 0 y 2)
```

Parámetro de escala	Detectadas	No detectadas	Tiempo (s)
0	1,813	614	23
1	1,987	440	93
2	1,975	456	353

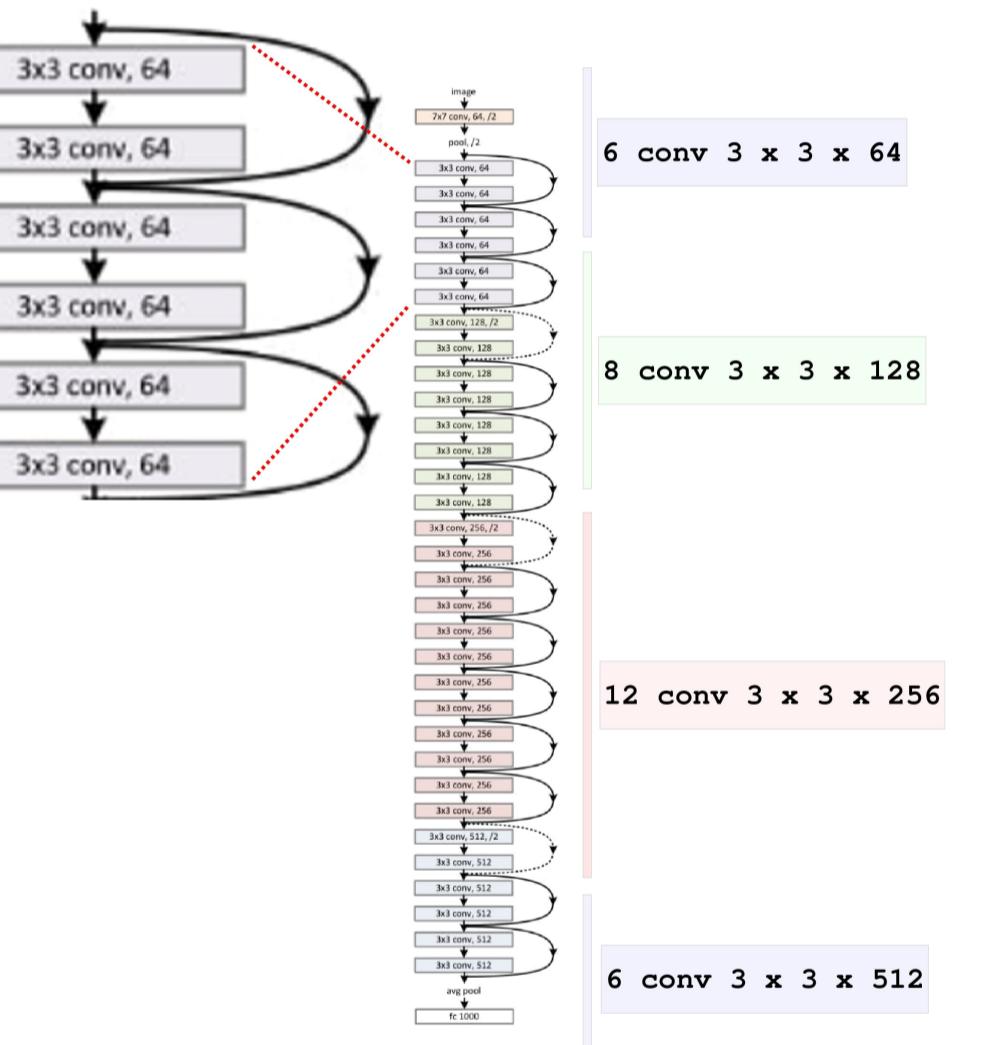
⁸El redimensionamiento es interno

En el caso de histograma de gradientes orientados, la función correspondiente de dlib sólo tiene un parámetro ajustable que, en la medida en que aumenta, al principio crece el número de detecciones ocupando un tiempo razonable, pero cuando se aumenta al siguiente nivel, las detecciones disminuyen y el tiempo aumenta 15 veces respecto al mejor tiempo.

DETECCIÓN — RN DLIB — MODELO DE RED

...Para aquellos interesados en los detalles del modelo, se trata de una red ResNet con 29 capas de convolución. Básicamente, es la red **ResNet-34** del artículo *Deep Residual Learning for Image Recognition* de He, Zhang, Ren, y Sun, con algunas capas menos y el número de filtros reducido a la mitad^a.

^a<http://blog.dlib.net/>



Esta es la red en la que se basa la implementación de dlib. Es una red residual resnet-34. El autor hizo modificaciones no bien especificadas.

DETECCIÓN — RN DLIB

Funciones de la red neuronal de **dlib**

Función	Propósito
detector=dlib.cnn_face_detection_model_v1 ("mmod_human_face_detector.dat")	Instanciar un objeto de detección
imagen = io.imread(archivo)	Función externa de lectura de imagen
Detecciones = detector(imagen, escala)	Objeto que regresa una tupla de datos

Arriba se ve un fragmento de código y abajo las tres funciones necesarias para producir el objeto de detecciones y al cual se le modifica el parámetro de escala o remuestreo interno.

DETECCIÓN — RN DLIB

Resultados de la ejecución de la red neuronal de **dlib**

Prueba	Remuestreo	Redimensionamiento ⁹	Detecciones	No. Detecciones	Tiempo (s)	Tiempo promedio (s) ¹⁰
1	0	no	1,891	536	298	0.1227
2	1	no	2,376	51	1,408	0.5801
3	2	no	2,379	48	5,847	2.4091
4	0	500	2,376	51	1,386	0.5710
5	1	500	2,376	51	14,282	5.8846

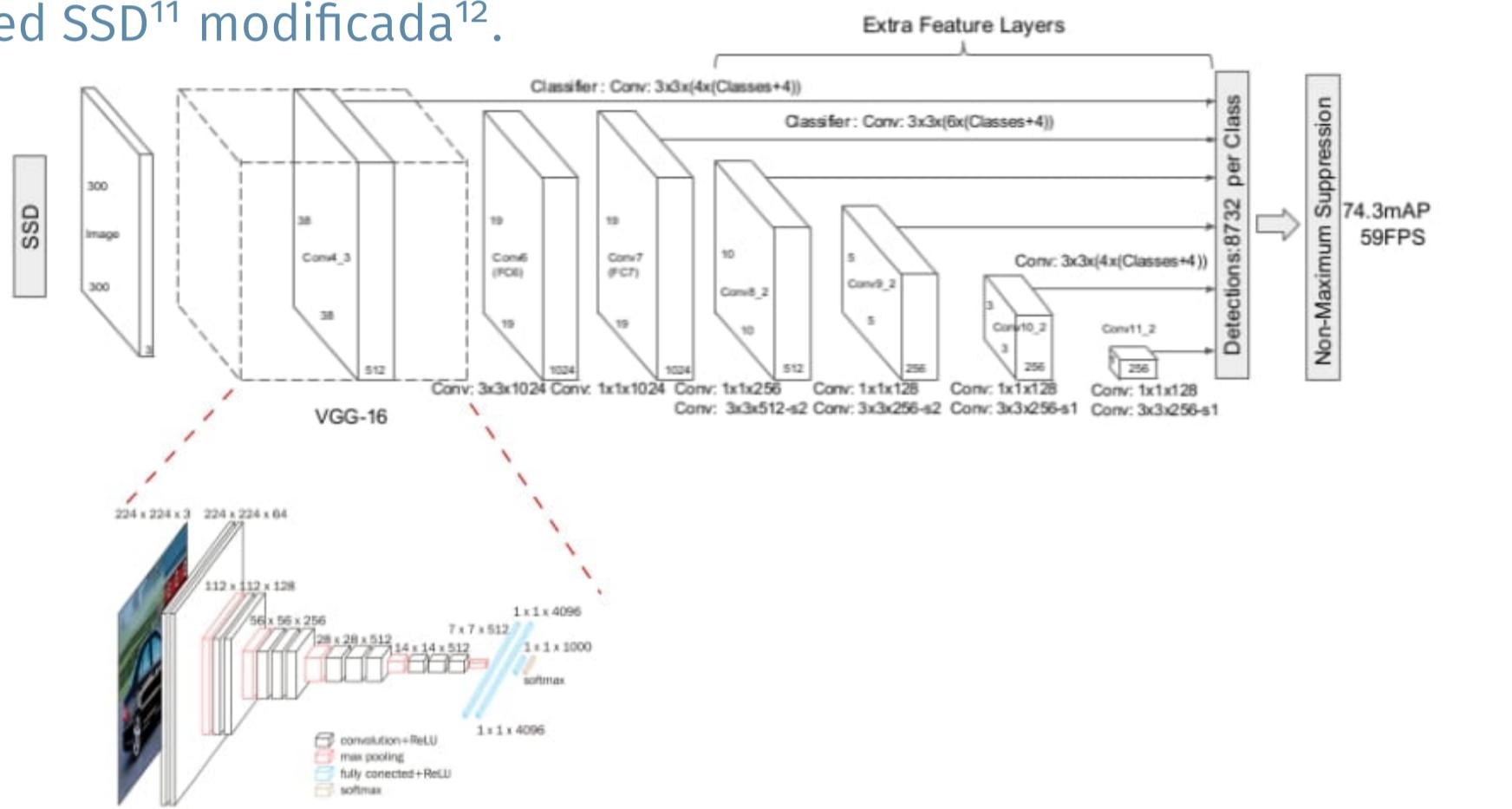
⁹píxeles de alto.

¹⁰tiempo entre número imágenes (2,427 siempre)

En la medida en que se aumenta el remuestreo, el número de detecciones aumenta, pero también aumenta el tiempo de procesamiento. **En su valor más bajo**, tarda 298 segundos en procesar 2,427 imágenes. Comparado con el mejor resultado de Haar, significa 6 veces más tiempo y menos detecciones.

DETECCIÓN — RN OPENCV

OpenCV Red SSD¹¹ modificada¹².



Este es el modelo de red en el que se basa.
A su vez, tiene una red vgg-016 incrustada.

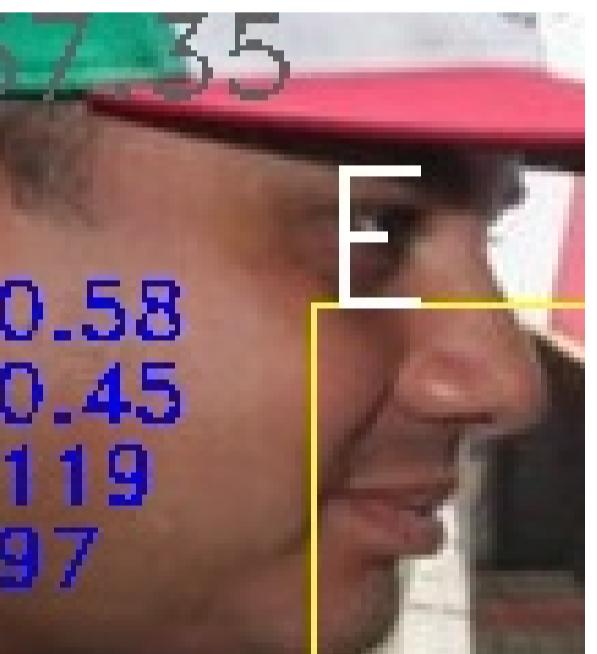
¹¹Wei Liu et al. SSD: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.

¹²https://github.com/opencv/opencv/blob/4.o.o-beta/samples/dnn/face_detector/how_to_train_face_detector.txt

DETECCIÓN — RN OPENCV

Red neuronal de **OpenCV**

Función	Propósito
1 cv2.dnn.readNetFromCaffe()	Cargar el modelo y los pesos.
2 cv2.dnn.blobFromImage()	Crear un objeto sobre el cual detectar.
3 detector.setInput(objeto)	Instanciar el objeto con la red.
4 dets = detector.forward()	Pasar la imagen (convertida en objeto) por la red y recibir una serie de datos.



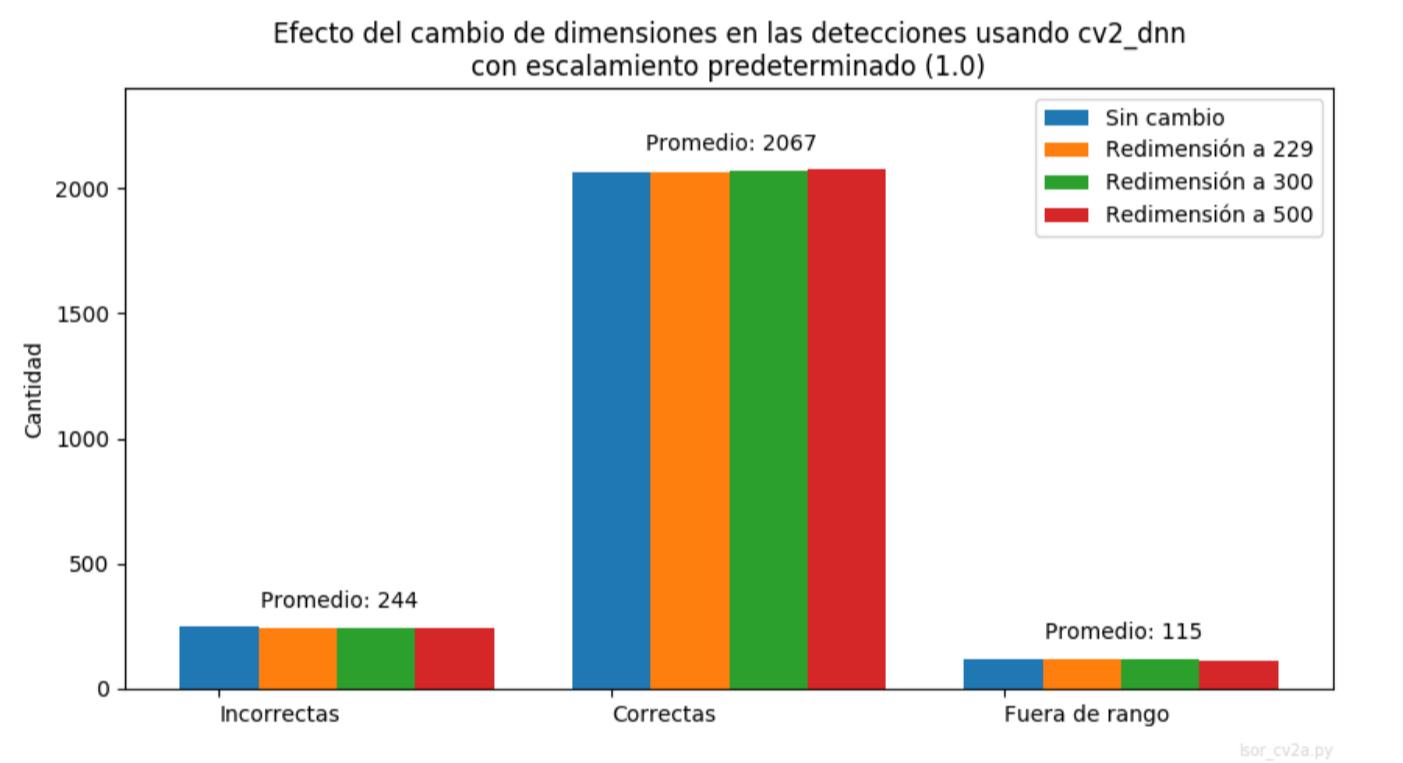
```
objeto = cv2.dnn.blobFromImage(imagen, scaleFactor=1.0, size,
mean, swapRB=true, crop=false)
```

OpenCV

Estas son las cuatro funciones usadas por la RN de OpenCV. Se encontró que uno de los valores regresados cuyos límites son de 0 a 100, se disparaba hasta el rango de miles. Esto indica un problema en las detecciones y la figura muestra uno de esos casos. Se enmarca con el inicio de la detección en el centro de la imagen pero sus extremos saliendo de los lados derecho

DETECCIÓN — RN OPENCV

Resultados de la red neuronal de **OpenCV**¹³



El detector se mostró muy parejo en los resultados, independientemente de las dimensiones de la imagen.

¹³Nota escrita durante el análisis:

No vale la pena hacer diagrama de tiempos si todas las redimensiones están en 4 minutos. Es difícil sacar un promedio por imagen debido a que el programa hace otras cosas aparte de detectar. Por eso simplemente se deja como una comparación en bloque.

DETECCIÓN — RESULTADOS

Resumen de resultados

Algoritmo	C ¹⁴	T ¹⁵	V ¹⁶
Haar	1,931	47	51.63
HOG	1.975 ¹⁷	353	6.87
dlib	2,376	1,408	1.72
OpenCV	2,067	269	9.02

¹⁴Número de caras detectadas.

¹⁵Tiempo en segundos

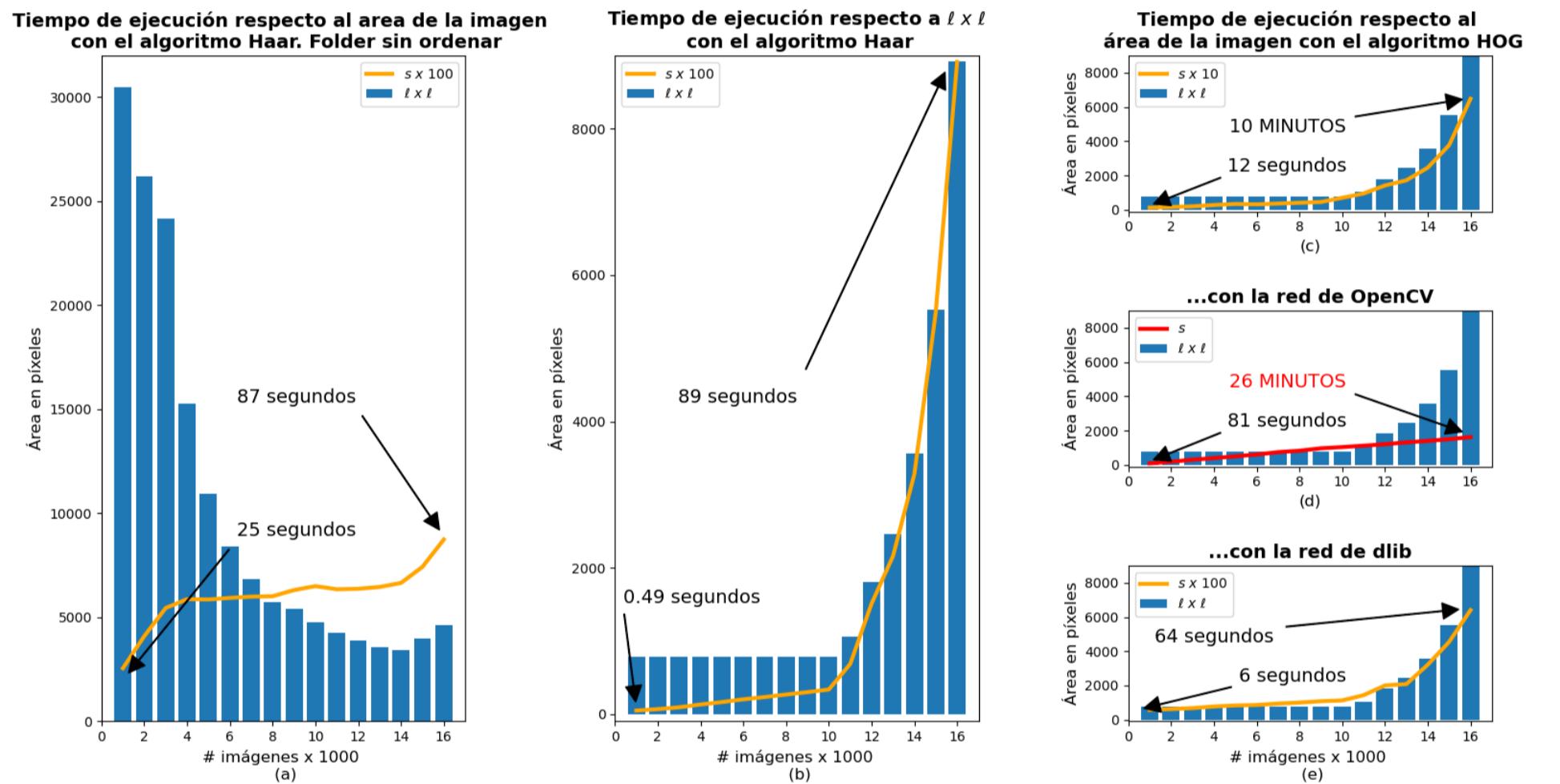
¹⁶Número de imágenes procesadas por segundo, de un total de 2,427

¹⁷Sólo 44 imágenes adicionales, a un costo de 7.5 veces más tiempo.

RESULTADOS de los algoritmos de detección
Se encontró que haar es más efectiva que las otras cuatro alternativas. **La más cercana detecta 44 caras adicionales, pero ocupa 7.55 veces más tiempo.**

ESTUDIO DE TIEMPOS DE PROCESO DE DETECCIÓN¹⁸

Estudio de tiempos de proceso de los algoritmos de detección respecto a la cantidad de imágenes procesadas



¹⁸Pruebas efectuadas en la computadora Vaio citada antes.

Se confirmó que los algoritmos de detección se comportan linealmente en el tiempo

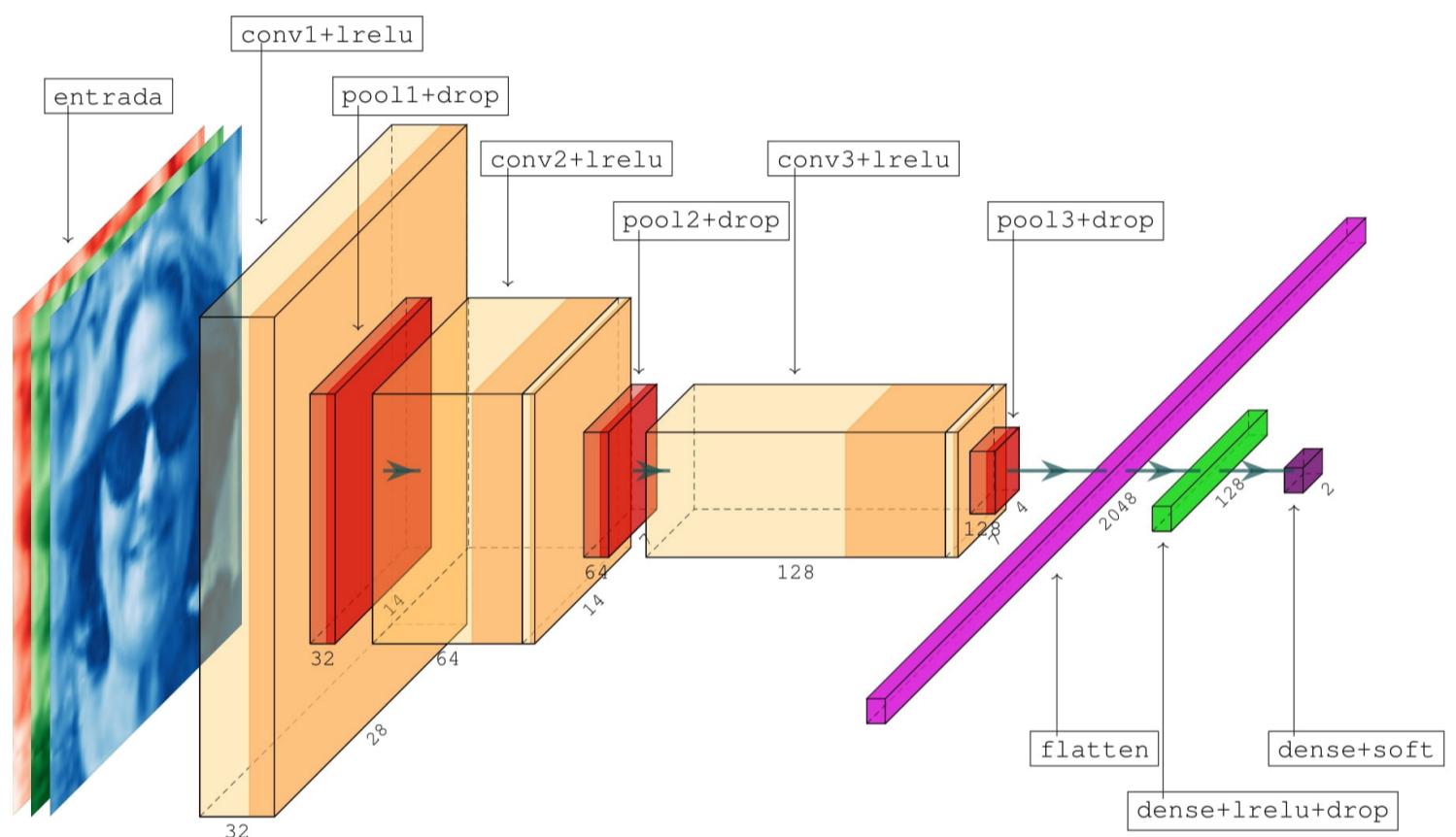
PÉRDIDA

Colección de imágenes para funciones de pérdida

-
- 1 red neuronal convolucional
 - 3 funciones de pérdida analizadas
 - 11,451 imágenes
 - 2 clases: mujeres y hombres
-

Para analizar funciones de pérdida se modificó una red neuronal se analizaron 3 funciones de pérdida y 11,451 imágenes repartidas en dos clases: mujeres y hombres.

PÉRDIDA — ARQUITECTURA¹⁹ DE LA RED SELECCIONADA²⁰



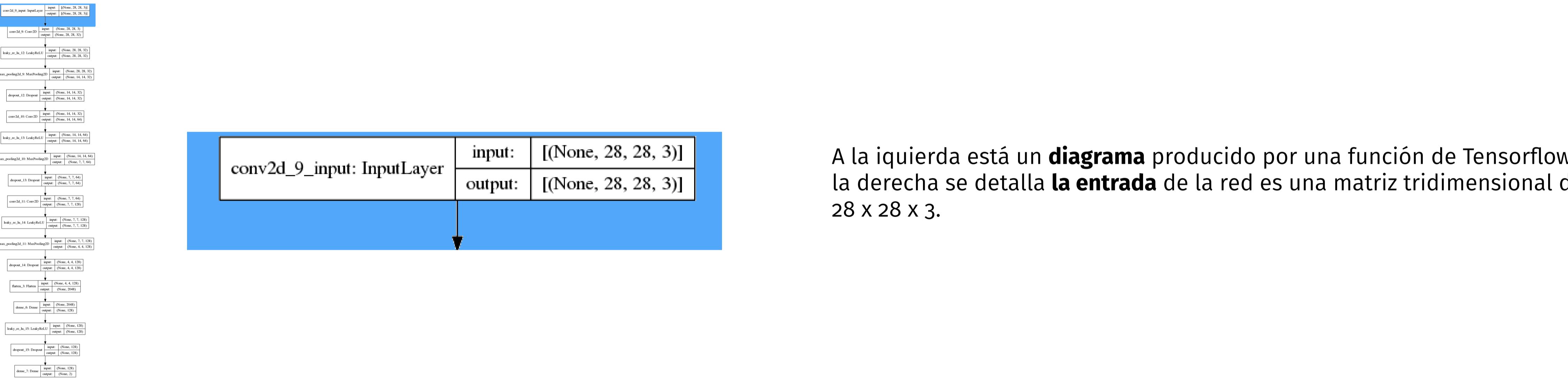
Esta es una representación tridimensional de la red emplead. Acabo de ver a una persona que se parece a esta cara.

¹⁹ Considerable modificación de la herramienta en: <https://github.com/Harislqbal88/PlotNeuralNet>. visitada: 2022-08-27.

²⁰ Abita Sharma. Convolutional neural networks in python with keras.

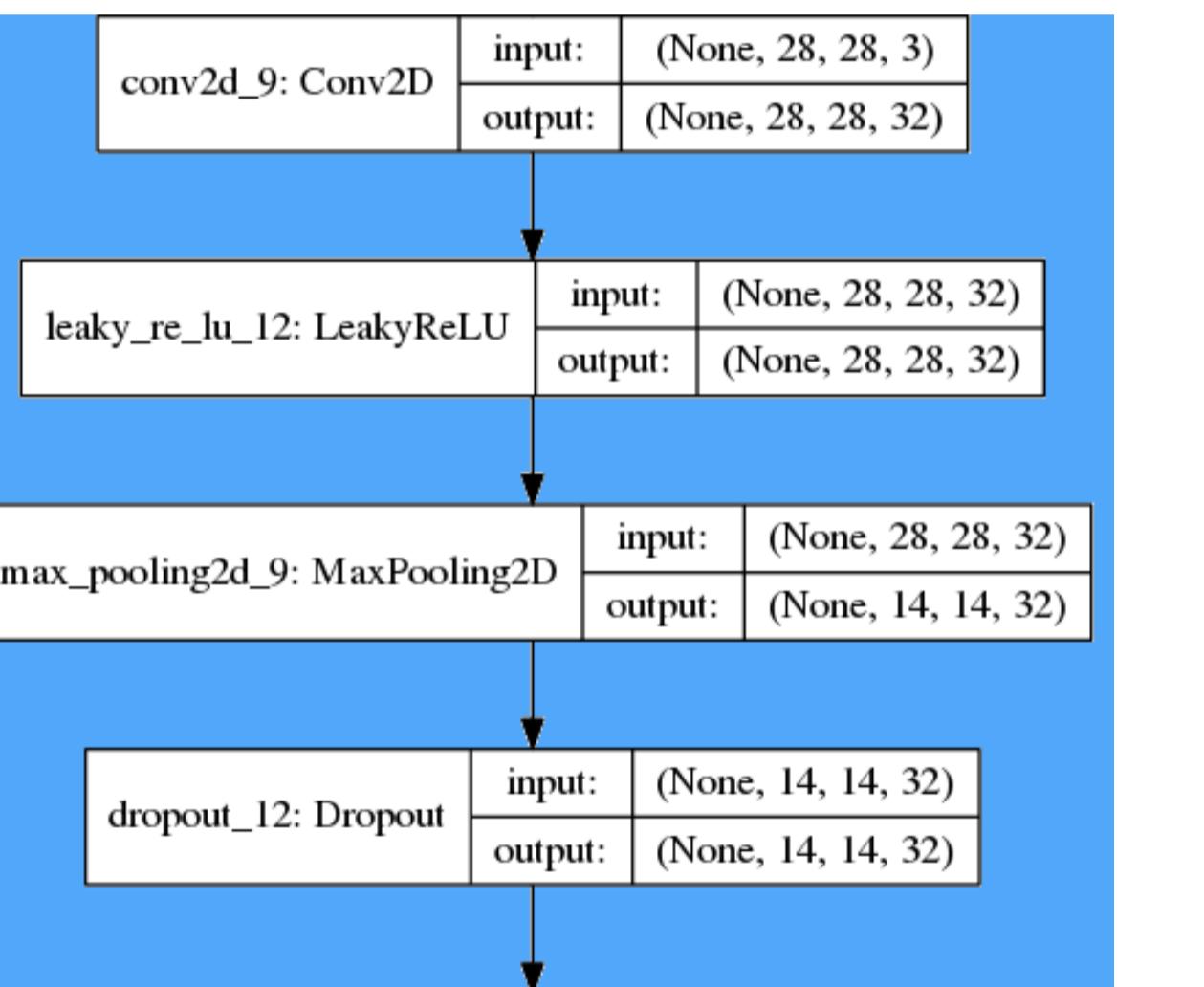
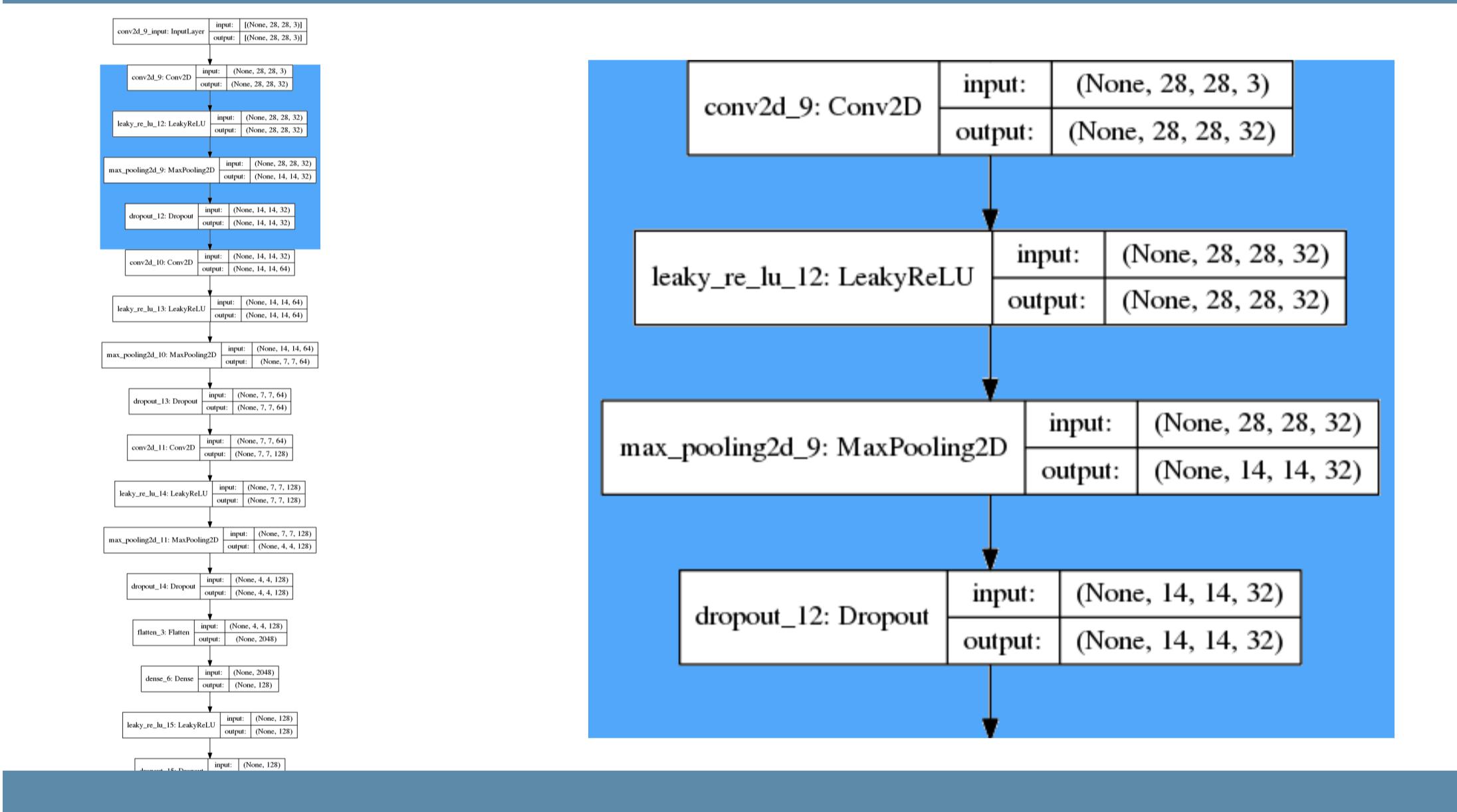
https://www.datacamp.com/community/tutorials/convolutional_neural_networks-python, 2017. visitada: 2021-06-23.

DESGLOSE DEL DIAGRAMA DE BLOQUES DE LA RED — ENTRADA



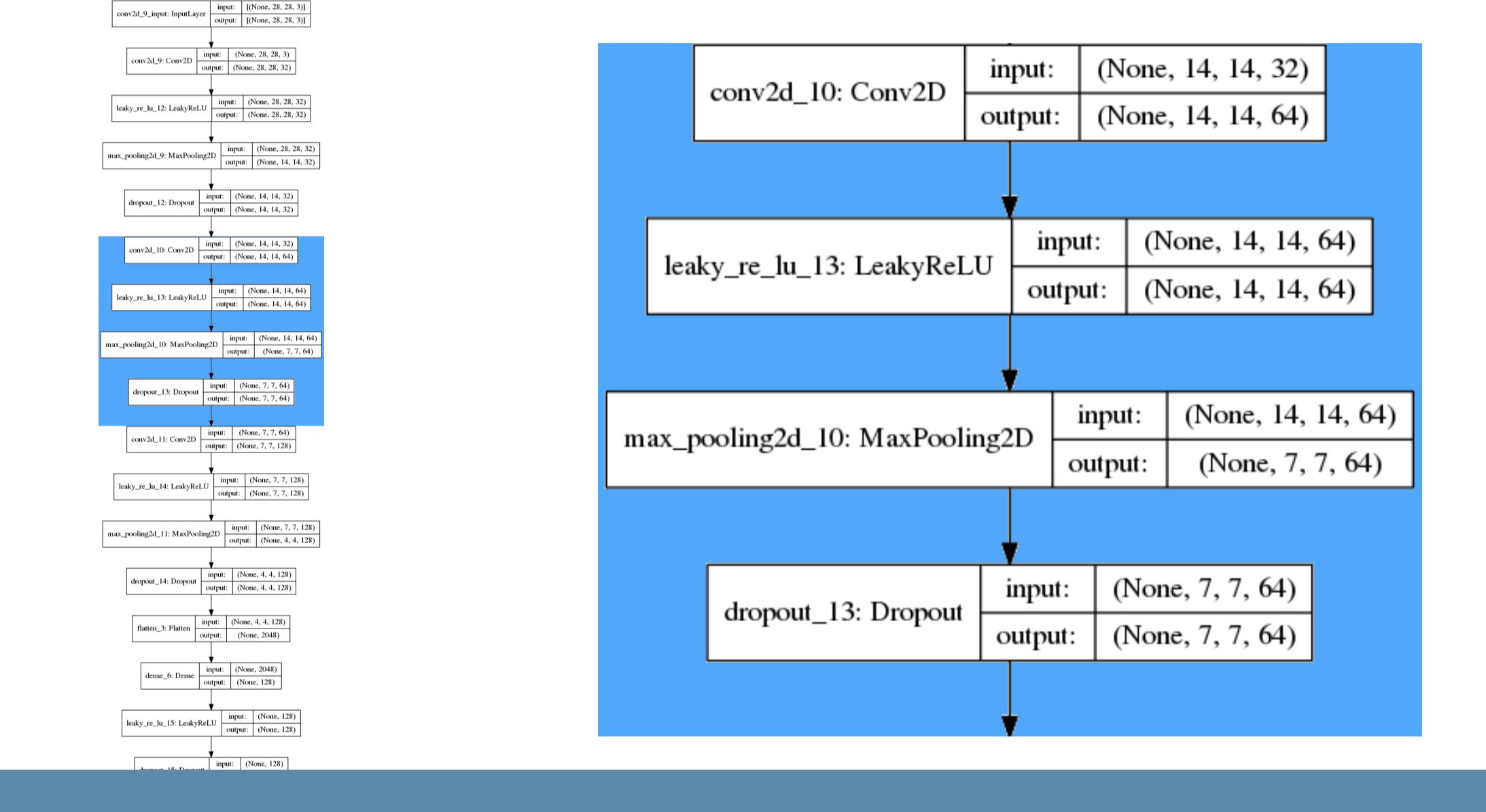
A la izquierda está un **diagrama** producido por una función de Tensorflow. A la derecha se detalla **la entrada** de la red es una matriz tridimensional de 28 x 28 x 3.

OTRAS REPRESENTACIONES DE LA RED — CONV_1



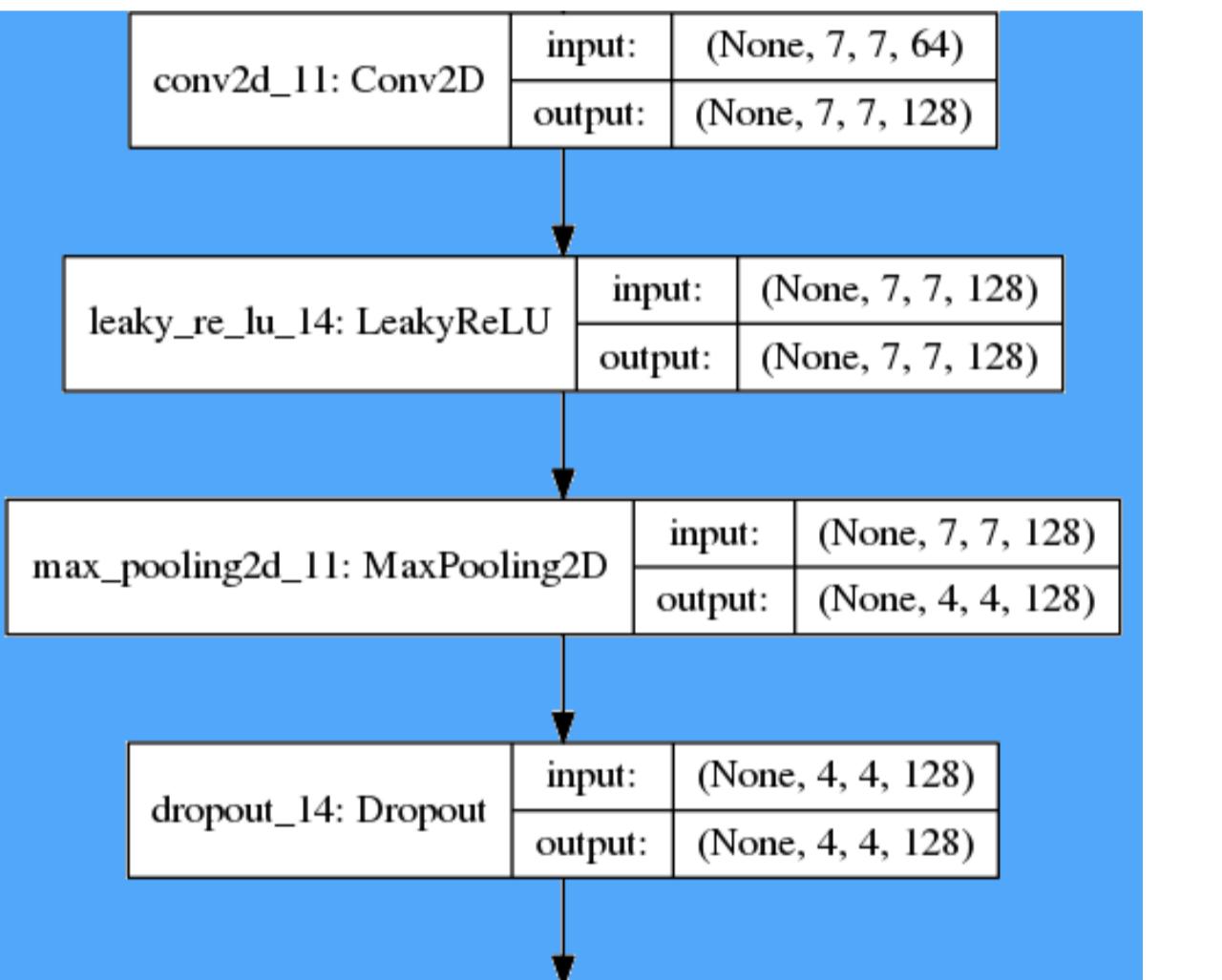
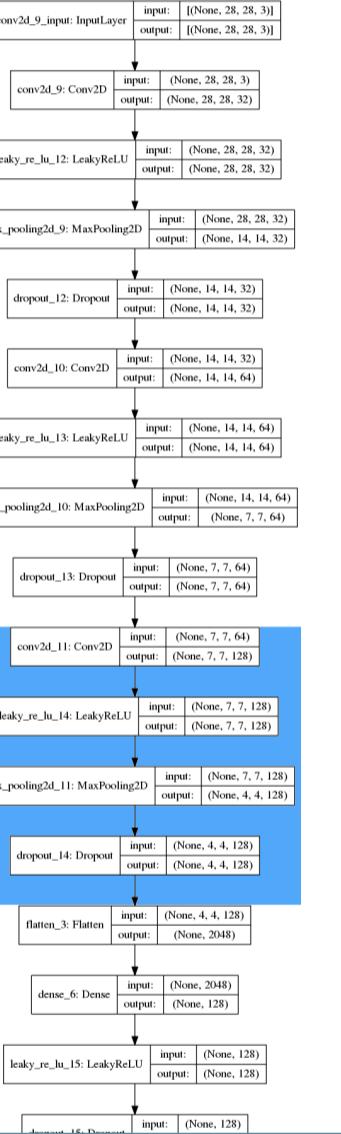
Esta pasa por tres bloques de convolucion función de activación extracción de máximos y abandono aleatorio Este es el primero.

OTRAS REPRESENTACIONES DE LA RED CONV_2



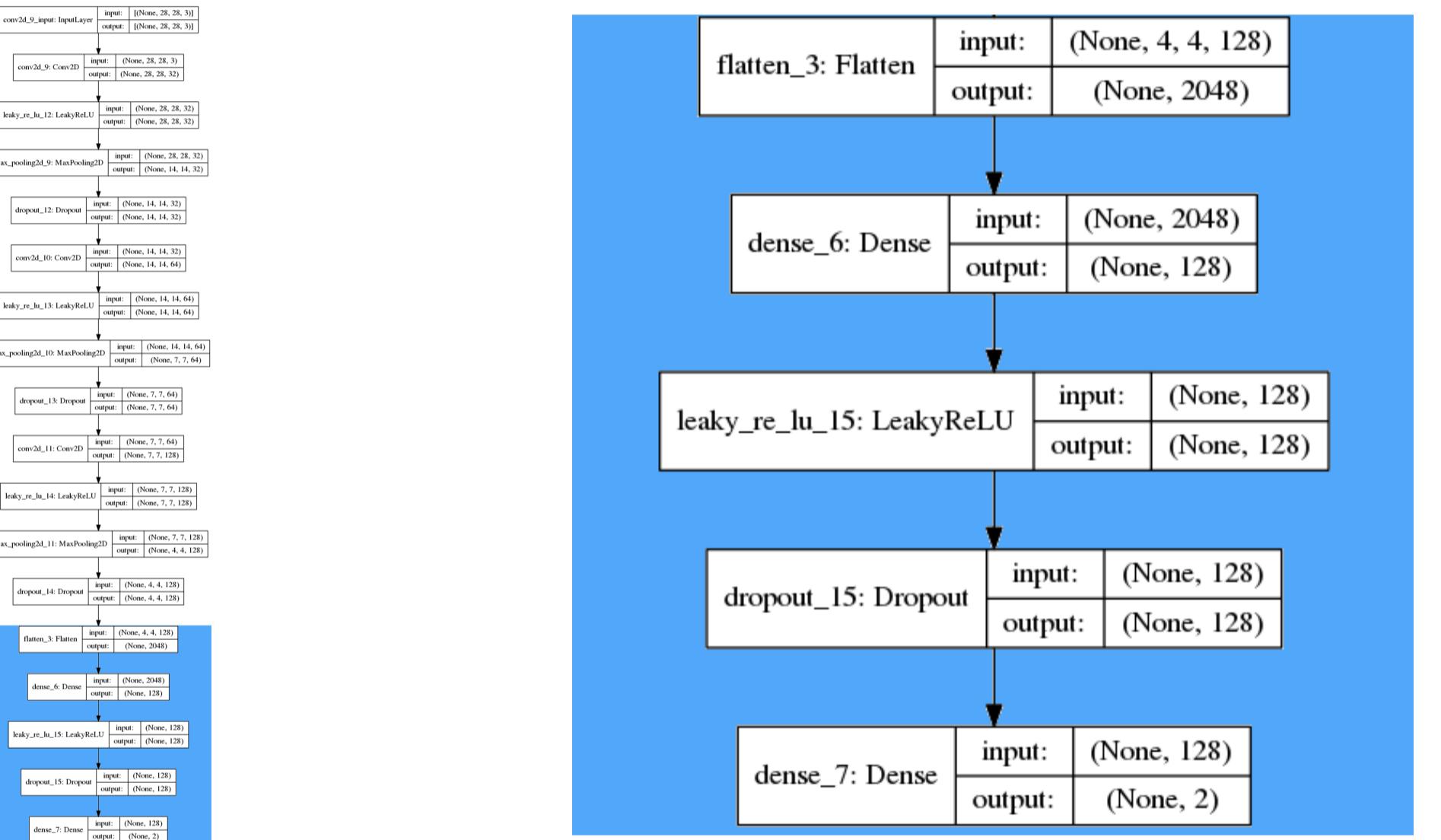
Esta el segundo bloque en el que vemos una reducción de los mapas de activación y un aumento en el número de canales como resultado de los pasos anteriores

OTRAS REPRESENTACIONES DE LA RED CONV_3



El **tercer** bloque ha reducido los mapas y nuevamente duplicó los canales

OTRAS REPRESENTACIONES DE LA RED — RESTO



Luego se **aplanan** los pesos y se **genera un vector** de 2,048 elementos ($4 \times 4 \times 128$). Se pasa por una **capa** completamente conectada con una entrada de 2,048 unidades y una salida de 128 unidades. A su vez se pasa por una **capa** de activación de la misma naturaleza de las anteriores y se obtiene **otro vector** de 128 elementos. Nuevamente se aplica una **capa de abandono** aleatorio y se conecta a otra **capa** completamente conectada en la que entran 128 unidades y salen dos, que corresponden al problema binario de este caso: la imagen de entrada es **de una mujer o de un hombre**. La ultima capa corresponde a softmax, que nos garantiza que la suma de los valores es 1 y por eso, el mayor, tiene la mas alta probabilidad. Cada unidad de esta capa de salida corresponde a una clase.

PÉRDIDA²¹

```
1 import tensorflow as tf
2 from tensorflow import keras ...
3 ruta='/home/usuario/folder/donde/estan/las/imagenes' # ruta a los datos
4 image_size = (28, 28) ... # parámetros de la red
5 X = tf.keras.preprocessing.image_dataset_from_directory( # función muy importante que ahorra
6                                         # tiempo en la preparación de los datos
7     ruta,
8     validation_split=0.2,      # parte la colección en 80% proceso y 20% validación
9     subset='training',        # indicación de que el subconjunto es para proceso
10    seed=1337,                # semilla de generación de lotes aleatorios
11    label_mode='categorical', # parámetro para armar las clases
12    image_size=image_size,    # dimensiones de la imagen
13    batch_size=batch_size,)   # tamaño del lote
14
15     modelo = keras.Sequential([capas])           'se define el modelo'
16
17
18 'SE COMPILA CON TRES PARAMETROS IMPORTANTES *****'
19 modelo.compile(loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),    '* pérdida'
20                  optimizer=keras.optimizers.Adam(),          '* optimizador'
21                  metrics=['accuracy'])                   '* desempeño'
22
23
24 hist=modelo.fit(train_ds, epochs=epocas, validation_data=Xv,) 'se ejecuta'
```

²¹Fragmento de código para definir, compilar y ejecutar una red. No se debe usar verbatim; se alteró por claridad.

Este fragmento de código destaca la función de compilación de la red en TF.

PÉRDIDA — DETALLE DE LA FUNCIÓN compile()

```
modelo.compile(loss={ tf.losses.mean_squared_error()
                      tf.losses.SparseCategoricalCrossentropy() , metrics=['accuracy']),
                  optimizer=keras.optimizers.
```

El detalle de la función muestra el lugar donde se modifican las funciones de pérdida aqui analizadas.

PÉRDIDA — ASPECTOS RELEVANTES

-
- Entrada original de 180 x 180 píxeles con solo 1,000 imágenes por clase
 - Inhibía por completo una computadora Centrino con 4GB RAM.
 - Se usó una Mac Mini core i5 con 8GB RAM.
 - Se bajó de 180 x 180 a 28 x 28 píxeles y se pasó de dos horas por ciclo a 5 minutos por ciclo.
 - Se regresó a la máquina original con más de 5,500 imágenes de 24 x 24 píxeles por clase.
-

De este segmento de estudio podemos destacar:

- Inicialmente se trabajó con imágenes de 180 x 180 píxeles y sólo 1,000 imágenes por clase
- Se vio que se inhibía por completo una computadora Centrino con 4GB de memoria
- Por ello se utilizó otra máquina equipada con procesador CORE i5 y 8GB de memoria.
- También se bajó la dimensión de las imágenes de 180 x 180 a 28 x 28 píxeles, con lo que se logró pasar de dos horas por ciclo a 5 minutos por ciclo.
- Se regresó a la máquina original con más de 5,500 imágenes por clase con dimensiones de 24 x 24 píxeles cada una.

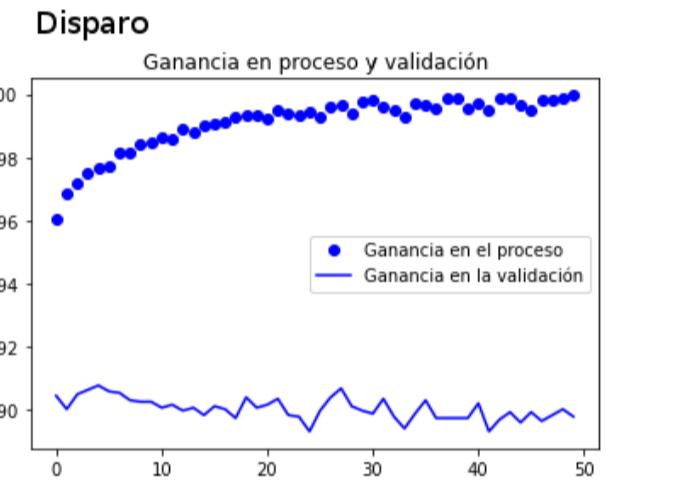
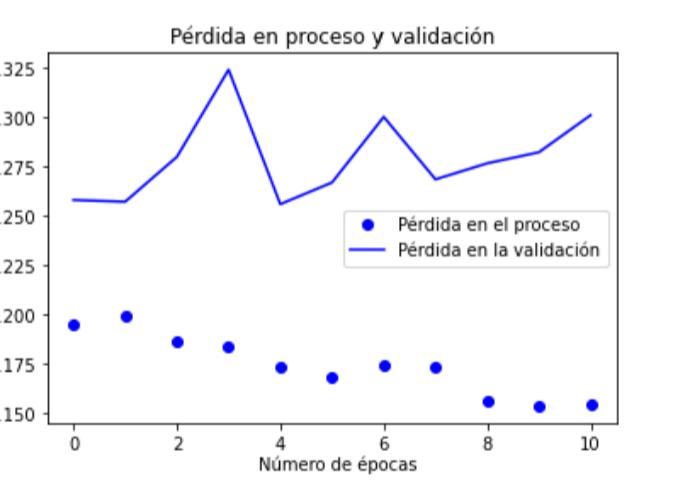
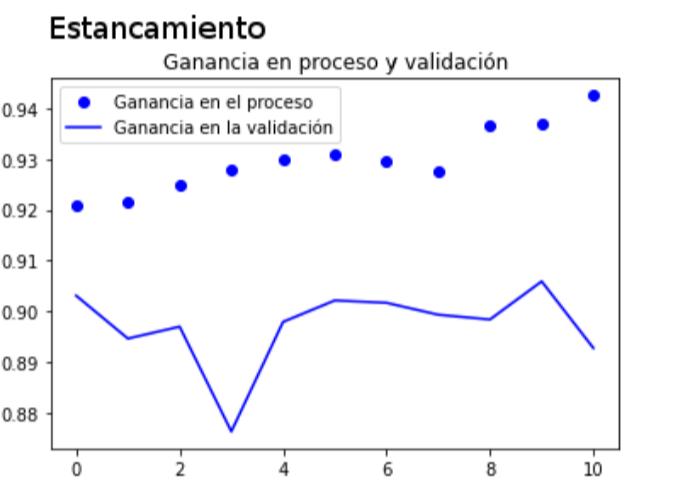
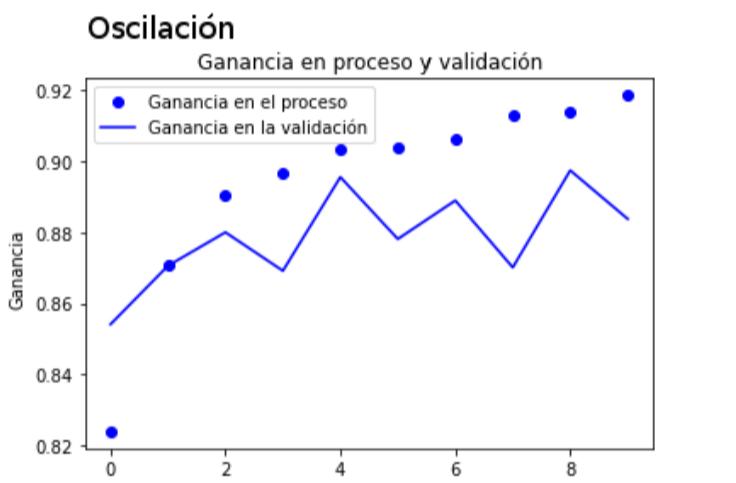
PÉRDIDA — COLECCIÓN DE IMÁGENES

Particiones de la colección de imágenes

Partición	Cantidad
Imágenes en total	11,451
Imágenes para procesar	10,756
Imágenes para validación	2,151
Imágenes para prueba	695
Imágenes clase 0	5,316+370
Imágenes clase 1	5,440+325

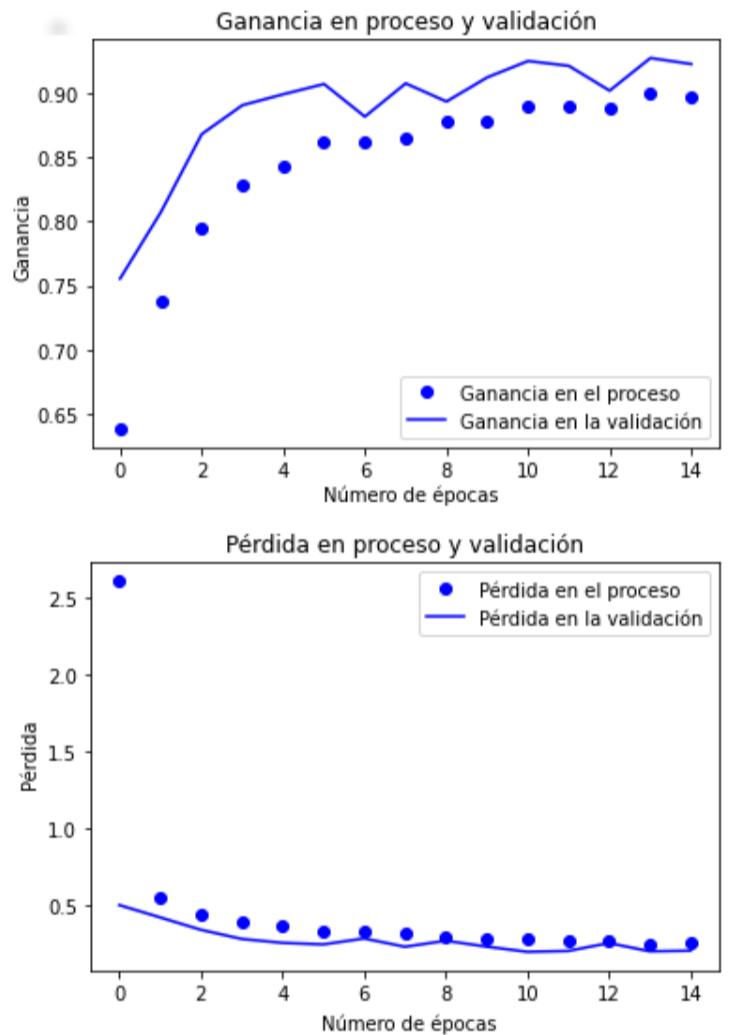
En la tabla se ven los detalles de la colección de imágenes de este apartado.

PÉRDIDA — INESTABILIDAD EN LA CONVERGENCIA



La imagen muestra tres casos en los que se recomienda detener la convergencia, cambiar de parámetros o abandonar el análisis de la función: oscilación, estancamiento y disparo

PÉRDIDA — RESULTADOS



Función	Pérdida	Ganancia
Euclíadiana ^a	0.4966	0.4968
Entropía ^b	0.2298	0.9031
Tripleta ^c	0.5273	0.4986

^aError medio absoluto

^bPérdida de entropía cruzada categórica dispersa

^cPérdida por tripleta

Resultados

La función de pérdida que mejor se comportó fue la de pérdida con entropía cruzada categórica dispersa, como se ve en la gráfica. Se observa una curva más suave que las otras, con una ligera oscilación, pero con la tendencia continua a crecer en ganancia y a decrecer en pérdida. Como comentario, la distancia euclíadiana en un hiperespacio es diferente a una 'distancia' entre distribuciones probabilísticas. En resumen, se usó una red de pocas capas, bajas dimensiones de imagen, sólo 15 ciclos y una alta ganancia.

CLASIFICACIÓN

Colección de imágenes para clasificación

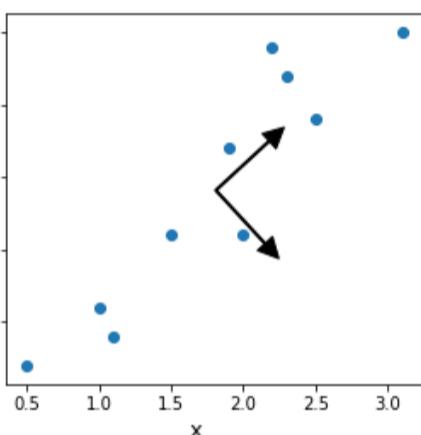
3 algoritmos analizados
489 imágenes de proceso
65 imágenes de prueba
9 identidades o clases

Para clasificación se analizaron 3 algoritmos, 489 imágenes de proceso y 65 imágenes de prueba para 9 identidades.

CLASIFICADORES – PCA – ASPECTOS TEORICOS

```
pca = f{media, varianza,  
desviación estándar,  
covarianza,  
autocorrelación,  
eigenvectores, eigenvalores}
```

```
1 import numpy as np  
2 from sklearn.decomposition import PCA  
3 ...  
4 X = np.array([datos])  
5 X = np.array([datos])  
6 pca = PCA(n_components=k)  
7 pca.fit(X)
```



PCA

El recuadro que se ve a la izquierda dice coloquialmente que PCA es una función de una familia de entidades estadísticas cuya representación se observa en la figura de la derecha. Abajo a izquierda se ve que todas esas entidades se encapsulan en dos funciones. Con ellas se pueden generar representaciones con tan sólo 22 dimensiones, como en el mosaico de abajo a la derecha. Se observa que aun así, las identidades son fácilmente reconocibles.

CLASIFICADORES – PCA

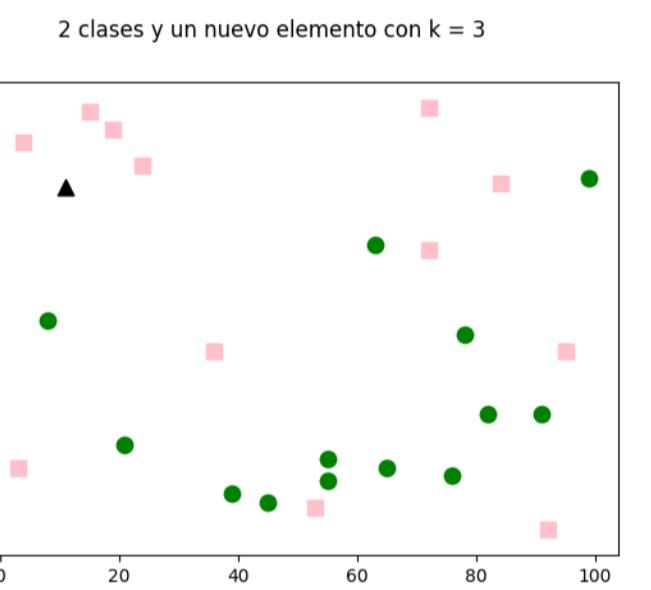
Resultados de PCA

Extractor de características	PCA	
Clasificador	k-NN	
Dimensiones	22	
Precisión	0.49230769230769234	
Repositorio	Memoria, reproceso	

Se ve que el extractor de características produjo un vector de 22 dimensiones. Este se paso a un clasificador k-nn debido a que PCA no es en sí un clasificador. La precisión fue inferior a 0.5. Los datos se cargan en memoria y se deben generar en cada ejecución. La imagen de la derecha muestra los resultados. En rojo los desaciertos y en verde los aciertos.

CLASIFICADORES — K -NN — ASPECTOS TEORICOS

```
1 from sklearn.neighbors import KNeighborsClassifier  
2 vecinos= KNeighborsClassifier(n_neighbors=3)  
3 vecinos.fit(x, y)  
  
1 from collections import Counter  
2 import numpy as np  
3 ...  
4 def estima_uno(x1,x_proc,y_proc,k):  
5     distancias=[]  
6     y=[]  
7     for i in range(len(x_proc)):  
8         distancia= ((x_proc[i,:]-x1)**2).sum()  
9         distancias.append([distancia,i])  
0     distancia=sorted(distancia)  
1  
2     for j in range(k):  
3         indice= distancias[j][1]  
4         y.append(y_proc[indice])  
5     return Counter(y).most_common(1)[0][0]
```



La primera parte muestra la implementación de la biblioteca scikit-learn y abajo una implementación más detallada del algoritmo. La figura corresponde a un programa que genera 25 muestras aleatorias de valores entre 0 y 100. Un nuevo elemento, también generado aleatoriamente, es clasificado como perteneciente a la clase de cuadrados rosas.

CLASIFICADORES — k -NN

Resultados de k -NN

Extractor de características	Ninguno	
Clasificador	k -NN	
Dimensiones	10,000	
Precisión	0.49230769230769234	
Repositorio	Memoria, reproceso	

Ahora se ve un ejemplo en el que no hubo reducción de dimensiones y se tomaron las imágenes tal cual, es decir, $100 \times 100 = 10,000$ elementos por imagen. Nuevamente se emplea k -NN, que usa distancia euclídea para ubicar elementos nuevos. Inesperadamente se encontraron los mismos resultados que reduciendo las dimensiones con PCA. Se hicieron nuevas pruebas comparando con $k = 1, 3, 5$ y 7 (siempre números impares) y se obtuvieron resultados iguales para el mismo k . Como en PCA, los datos se cargan en memoria y se deben generar en cada ejecución.

CLASIFICADORES – SVM – ASPECTOS TEORICOS

Decimos que el conjunto puede ser separado por un hiperplano de la forma:

$$(\vec{w} \cdot \vec{x}) + b = 0 \quad (1)$$

Luego de un proceso algebraico, podemos llegar a la siguiente expresión:

$$(1 - b - 1 + b)/\|\vec{w}\| = 2/\|\vec{w}\|, \quad (2)$$

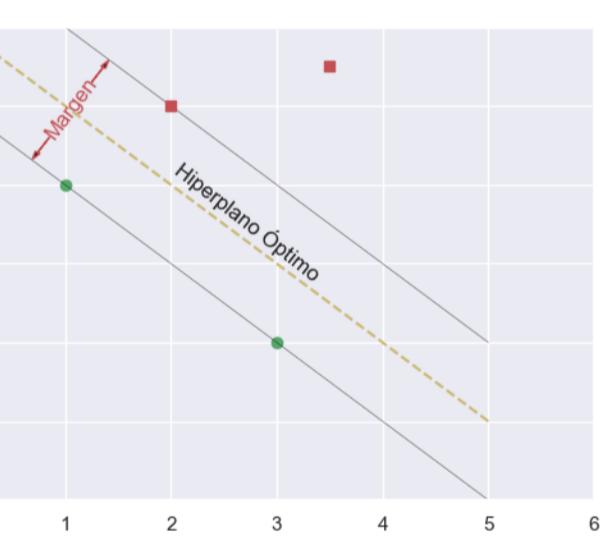
Los multiplicadores de Lagrange que nos ayudarán a optimizar la siguiente expresión:

$$\mathcal{L} = 1/2\|\vec{w}\|^2 - \sum \alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1] \quad (3)$$

Luego de resolver las restricciones de Lagrange llegamos a la siguiente solución:

$$\sum \alpha_i y_i \vec{x}_i \cdot \vec{w} + b \geq 0 \quad (4)$$

```
1 from sklearn.preprocessing import LabelEncoder
2 from sklearn.svm import SVC
3 ...
4 reconocedor = SVC(C=1.0, kernel="linear", probability=True)
5 reconocedor.fit(data["vectores"], identidades)
```



SVM

Arriba a la izquierda se observa un resumen de la formulación matemática del algoritmo y a la derecha el resultado gráfico del mismo. Se observan dos líneas sólidas y en medio una línea punteada. Las líneas sólidas son tocadas por puntos y se les da el nombre de vectores de soporte. La línea punteada es conocida como hiperplano óptimo y es equidistante a las líneas exteriores. La separación entre las líneas exteriores se conoce como margen máximo y encontrarlo es el objetivo del algoritmo. Abajo a la izquierda se ve la implementación de la biblioteca scikit-learn.

CLASIFICADORES

Resultados de SVM

Extractor de características	Openface*
Clasificador	SVM
Dimensiones	128
Precisión	0.7384615384615385
Repositorio	Disco, incremental



* Brandon Amos, et al. Openface:A general-purpose face recognition library with mobile applications. CMU School of Computer Science, 6(2), 2016.

SVM fue diseñado para clasificar desde su origen. Trata de encontrar el hiperplano óptimo en clases linealmente separables. Cuando no es así, se utiliza un mecanismo llamado kernel, que básicamente aumenta una dimensión al conjunto de datos y con esto se logra la separación buscada. Se usó un extractor de características basado en OpenFace el cual produjo un vector de 128 enteros. Este se alimentó a un clasificador SVN (implementado en scikit pero que prácticamente da los mismos resultados que la versión de OpenCV). El resultado se muestra en la tabla. ANTES, usando parte de las imágenes de enrolamiento originales, se lograron resultados arriba de 90 %, pero se consideró que había un sobreajuste en el mecanismo de generación de vectores y se decidió cambiar por otras imágenes de prueba tomadas en distintas condiciones. El repositorio se mantiene en disco y se hizo que fuera incremental.

CLASIFICADORES

Resumen de resultados por algoritmo

Algoritmo	Precisión
PCA	('Accuracy score is', 0.49230769230769234)
k-NN	('Accuracy score is', 0.49230769230769234)
SVM	('Accuracy score is', 0.7384615384615385)

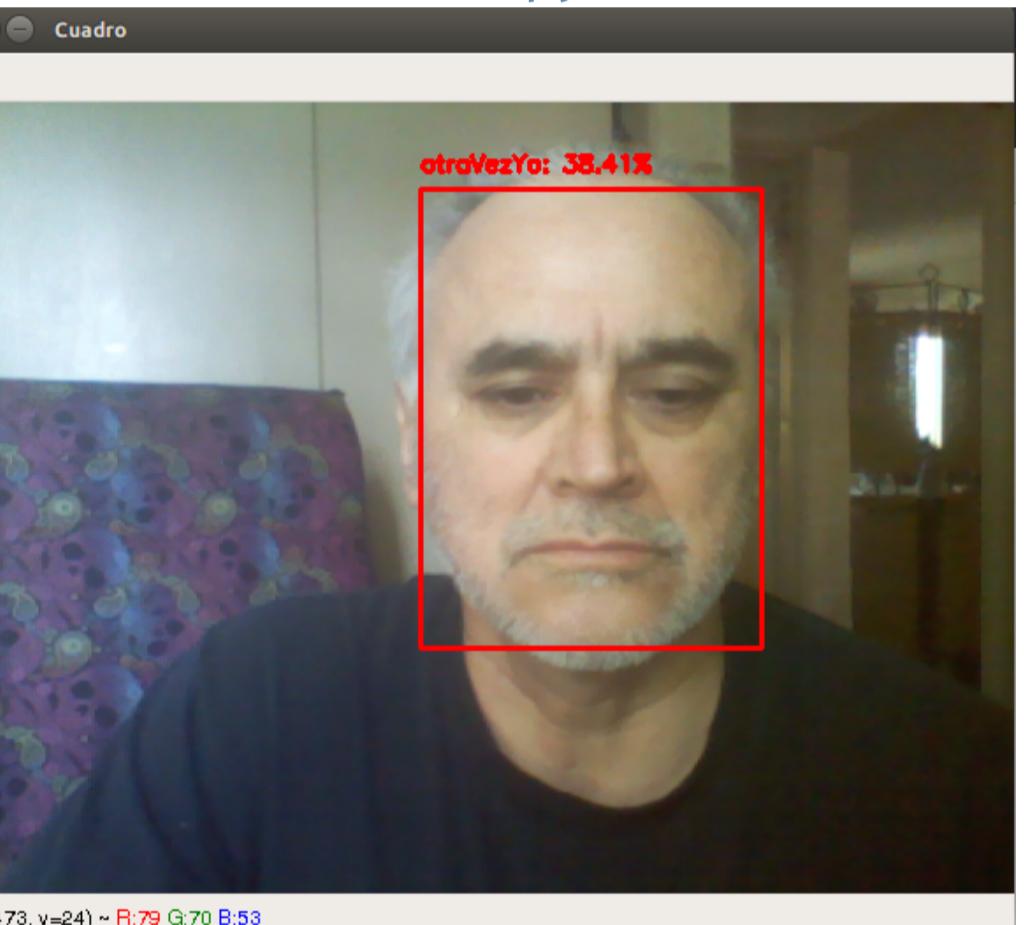
La tabla resume los resultados. SVM es mejor

RESULTADOS – INTERFAZ GRÁFICA Y DE CONSOLA

Programa Amaxayac



python reconocevCon.py



Se hizo una aplicación de escritorio que abarca un proceso completo entre el enrolamiento de la persona, hasta el registro de incidencias (entradas, salidas y rechazos). Una aplicación de consola entra en un ciclo permanente de detección, estimación de la identidad, reporte de la etimación, y registro de incidencia. Con estos resultados se pudieron hacer las dos comparaciones siguientes.

RESULTADOS – COMPARACIÓN CON SOLUCIÓN ACADÉMICA

Concepto	UCM ²²	Amaxayac
Microcontrolador	Raspberry Pi	Raspberry Pi B+
Lenguaje de programación	Python	Python
Base de datos	ERP ²³	Archivo Pickle
Aplicación de escritorio	Sí	Sí
Servidor dedicado	Sí	<u>Cualquier PC</u>
Reconocimiento en la terminal	No	<u>Sí</u>
Consulta en la terminal	Sí	No
Enrolamiento en la terminal	Sí	Sí
Edición conjunto de imágenes capturadas	No específica	<u>Sí</u>

²²Universidad Carnegie Mellon, Pennsylvania, EEUU

²³Siglas en inglés que se refieren a un 'Sistema de planificación de recursos empresariales.'

Con la solución académica se pueden ver varios indicadores similares pero se destaca que en este trabajo se hace el reconocimiento en la terminal, evitando el viaje de datos por la red.

RESULTADOS – COMPARACIÓN CON SOLUCIÓN COMERCIAL

Concepto	Comercial	Amaxayac
Número de identidades	10 mil	150
Tiempo de respuesta	No más de 5s en positivo verdadero	De instantáneo a un segundo para estimar frecuencia. 4 en promedio.
Falsos positivos	No reporta. No permite el acceso.	No reporta falsos positivos
Falsos negativos	Frecuentes con occlusiones diversas.	No
Antifalsificación	Sí	Muchos tipos de cara
Regionalización	Muchos tipos de cara	Red neuronal.
Tecnología	No indica	MySQL para incidencias , archivos binarios para identidades.
<u>Base de datos</u>	MySQL. Reportes excel. No directo.	Menos de 10 min. Permite edición.
Tiempo de enrolamiento	Menos de 10 min.	Menos de 10 min. Permite edición.
Nivel de iluminación	Iluminación infrarroja.	Medio. Sensible al exceso.
Recuadro	Sí	Sí

Se está lejos de una solución comercial tan sólo mirando la cantidad de identidades que aquella puede manejar. Lo máximo probado en este estudio es de 150 identidades.

CONCLUSIONES

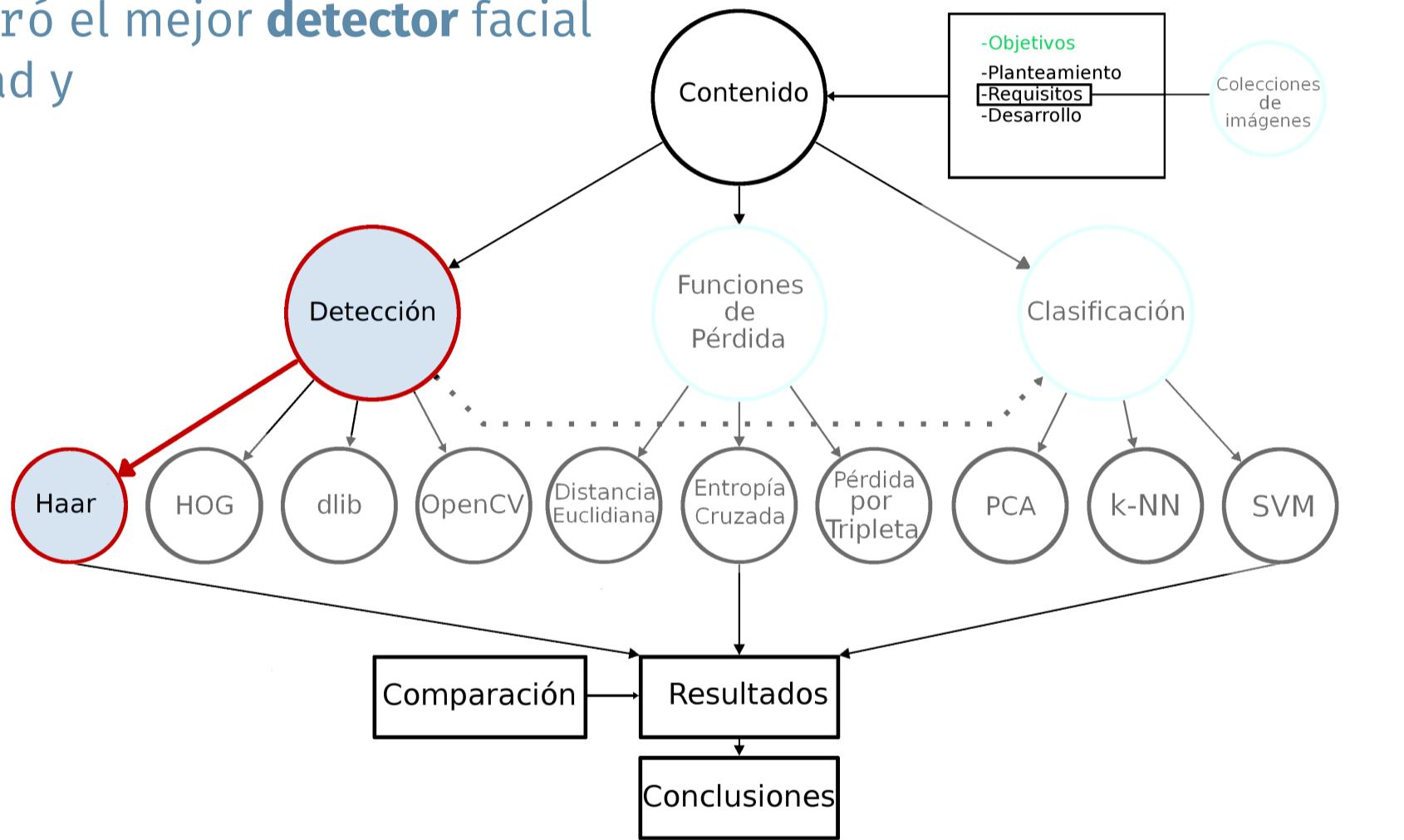
En relación con los Objetivos específicos, **en este trabajo** se logró encontrar que:

Objetivo		
1	Haar es el mejor detector	1,931 detecciones en 47 segundos; 51.63 imágenes por segundo
2	Entropía cruzada es la mejor función de pérdida	Pérdida: 0.2298, ganancia: 0.9031
3	SVM es el mejor clasificador	('Accuracy score is', 0.7384615384615385)
4	Se crearon las colecciones de imágenes propias	Tres colecciones, 14.5k imágenes, 21 clases

Se puede ver el resultado de cada familia.

DIAGRAMA DE CONCLUSIONES (1/4) – OBJETIVO I, CUMPLIDO

Se encontró el mejor **detector** facial en velocidad y exactitud.

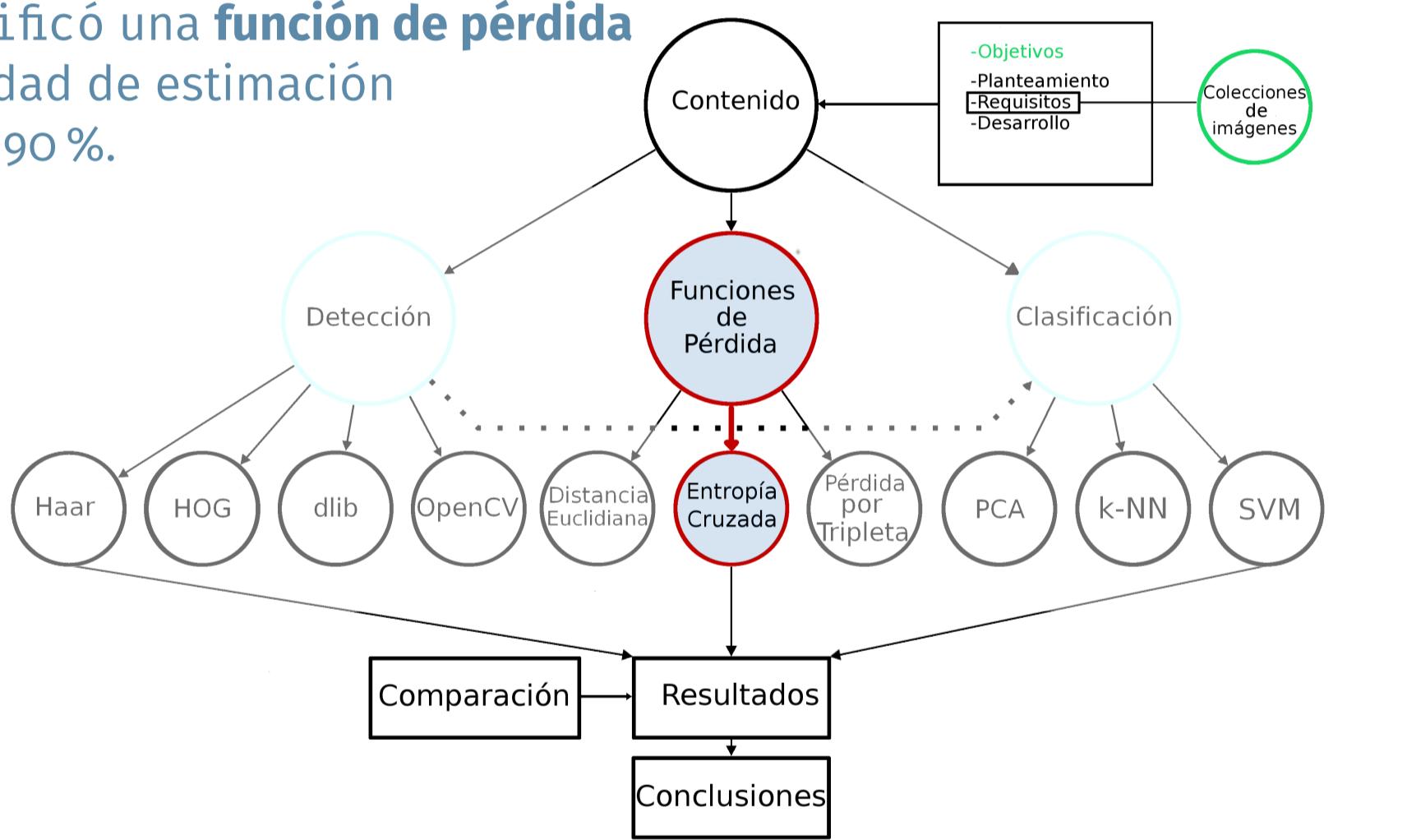


Conclusiones

Se cumplió objetivo i) al encontrar el mejor detector facial en velocidad y exactitud:
Haar

DIAGRAMA DE CONCLUSIONES (2/4) – OBJETIVO II, CUMPLIDO

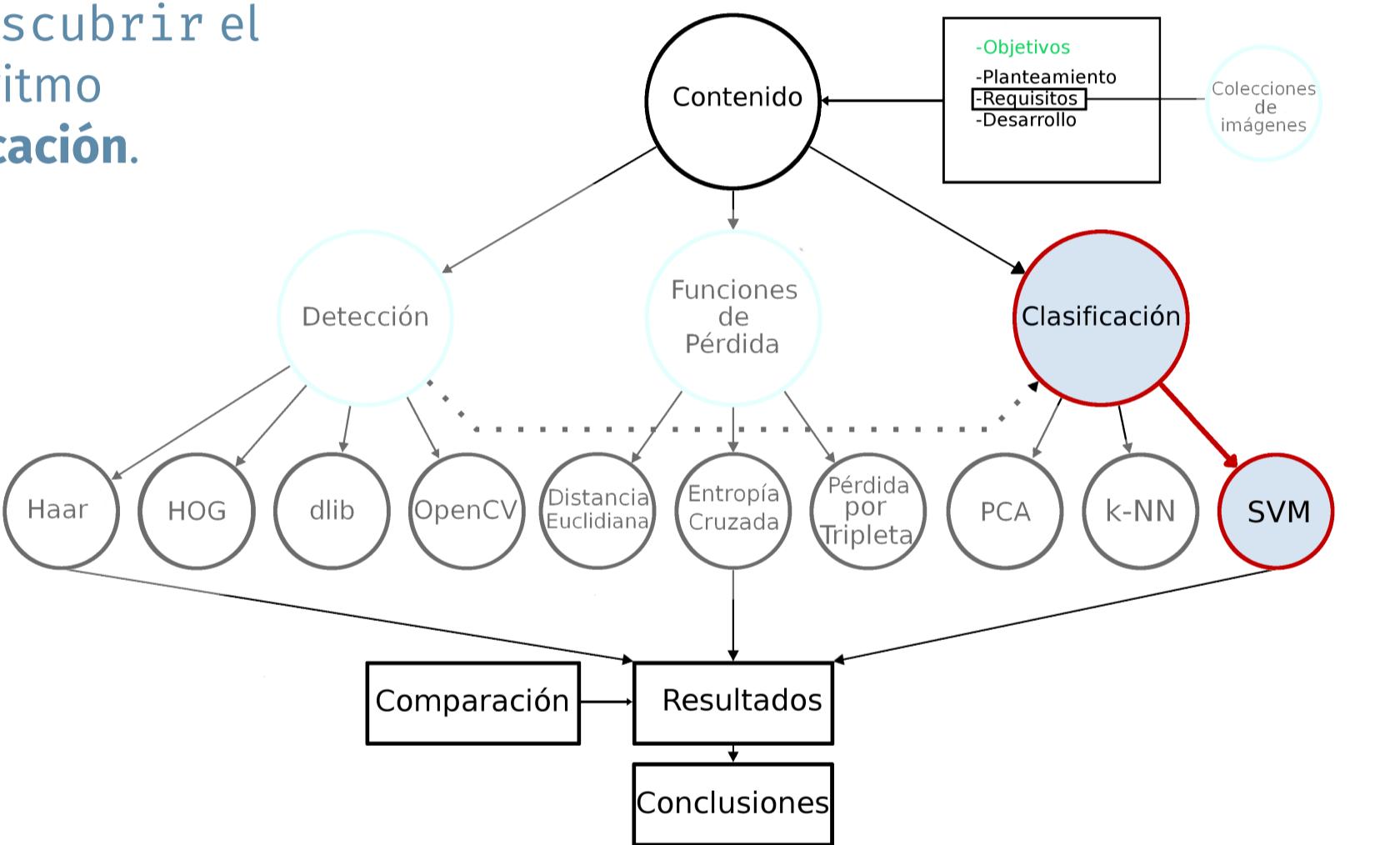
Se identificó una **función de pérdida** con capacidad de estimación mayor que 90 %.



Se cumplió el objetivo ii) al encontrar la mejor función de pérdida.

DIAGRAMA DE CONCLUSIONES (3/4) – OBJETIVO III, CUMPLIDO

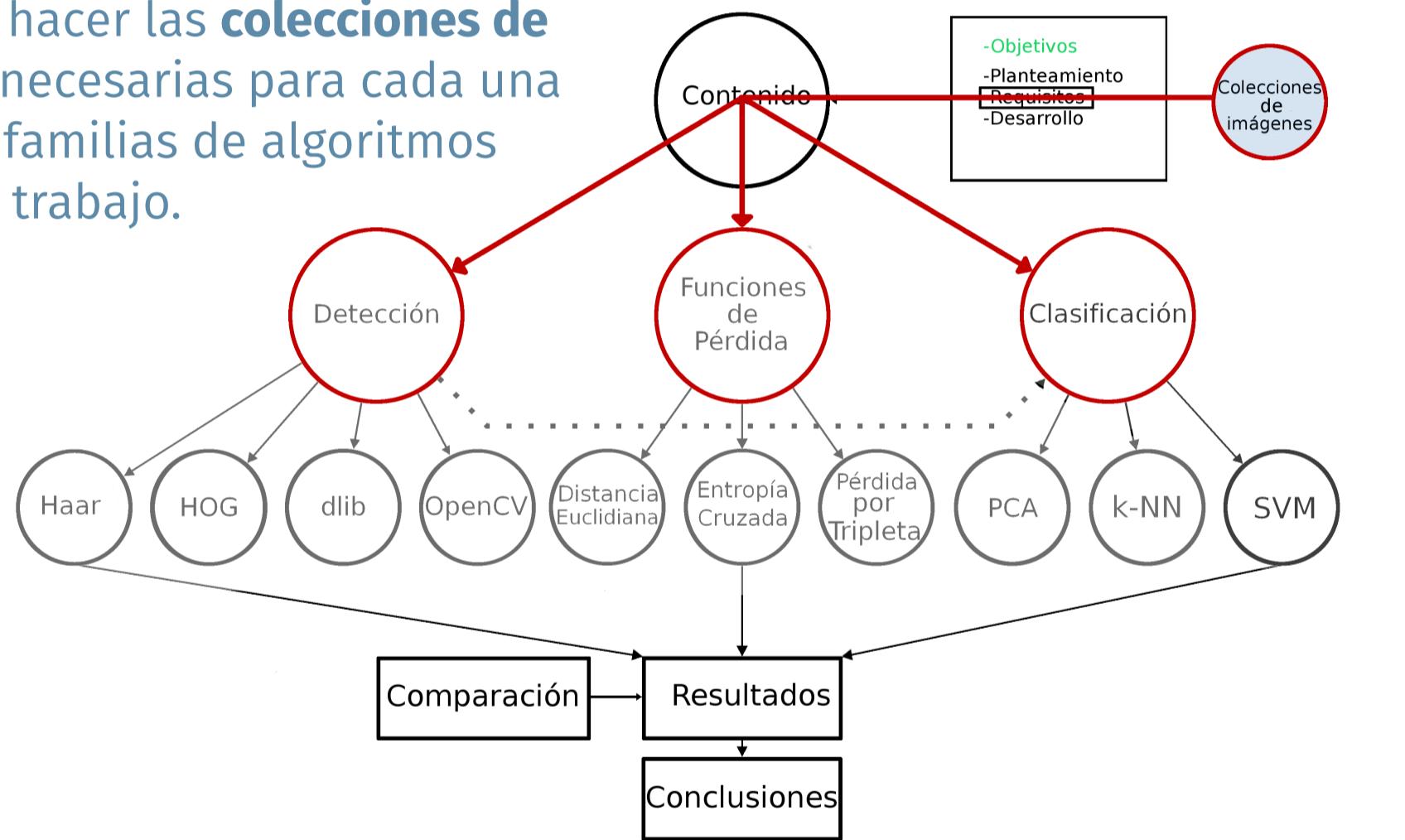
Se pudo descubrir el mejor algoritmo de **clasificación**.



Se cumplió el objetivo iii), al encontrar el mejor algoritmo de clasificación

DIAGRAMA DE CONCLUSIONES (4/4) – OBJETIVO IV, CUMPLIDO

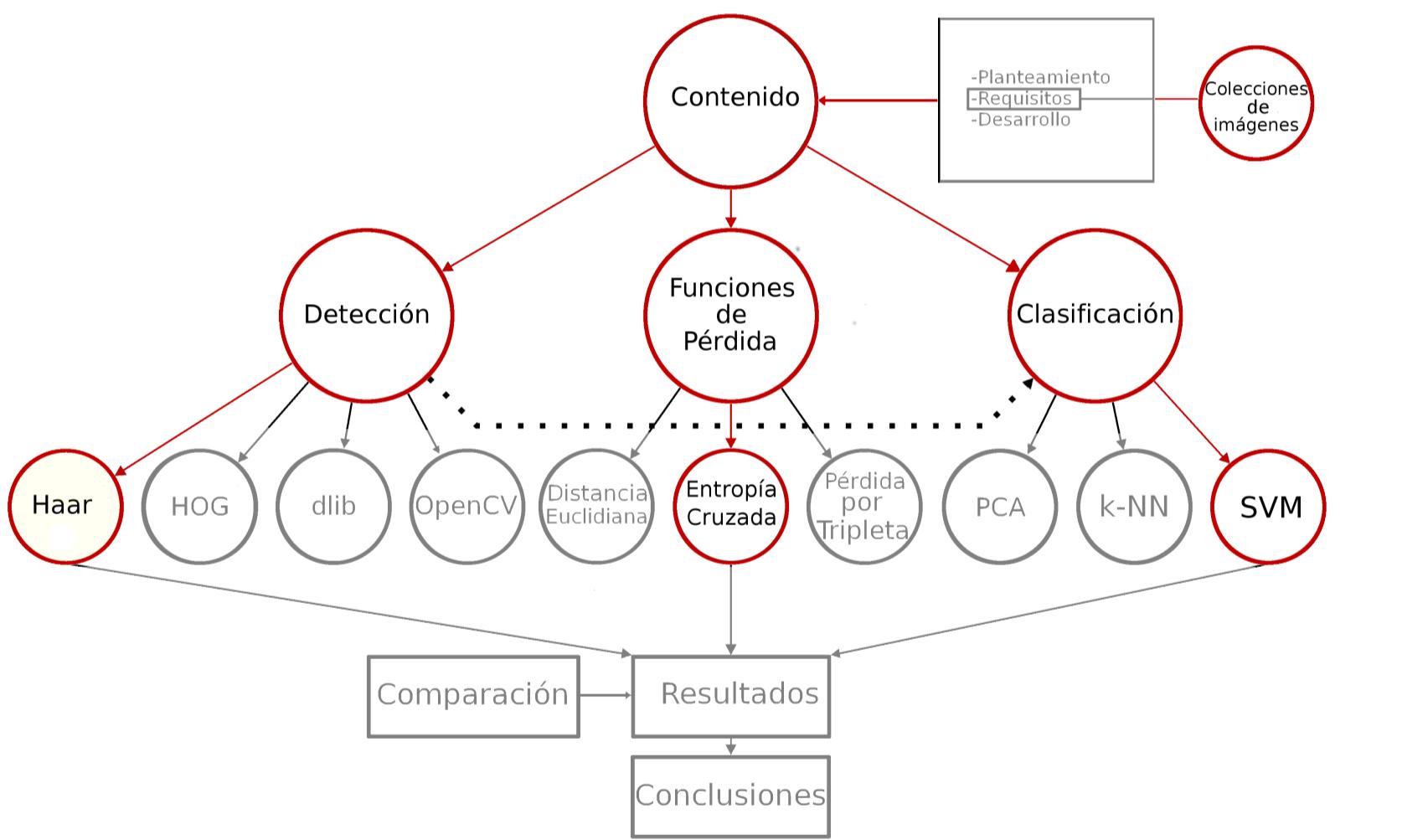
Se logró hacer las **colecciones de imágenes**, necesarias para cada una de las tres familias de algoritmos de nuestro trabajo.



Se cumplió el Objetivo iv) al crear tres colecciones de imágenes con las que se trabajó en cada uno de los objetivos anteriores.

DIAGRAMA DE CONCLUSIONES — CIERRE

Todos los Objetivos se cumplieron



Todos los Objetivos se cumplieron.

APORTACIONES

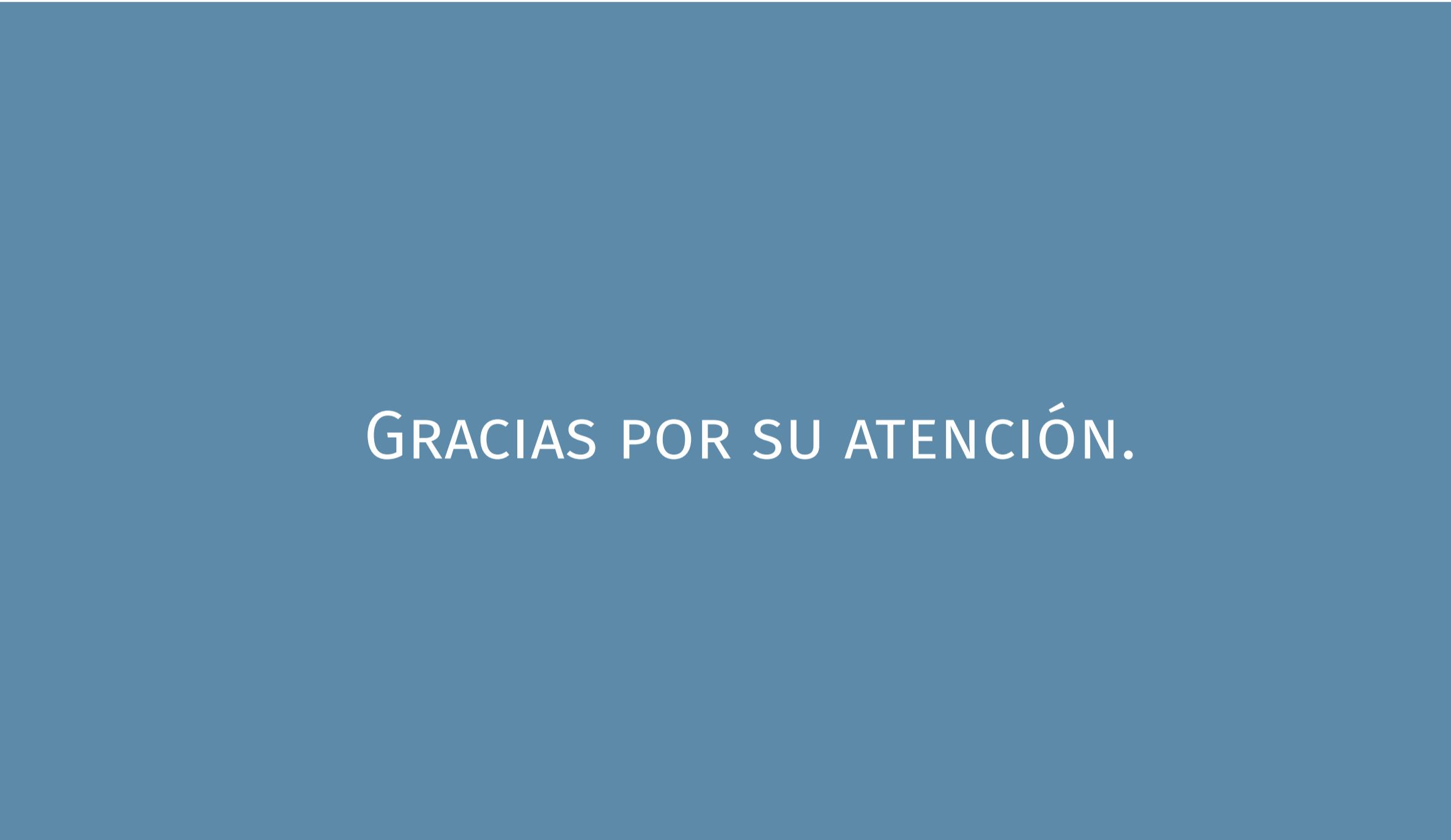
- Se **mostró** el proceso de creación de colecciones de imágenes propias.
- Quedaron **descubiertos** varios elementos no documentados en las funciones empleadas.
- Se **mostró de manera indirecta**, el gran poder de representación que se logra con PCA
- Se **hicieron** herramientas para el análisis de algoritmos usando datos propios.
- Con el material de este trabajo se pueden **escribir** varios artículos.
- Se **hace disponible** al público todo el material empleado en este trabajo.
- Se **logró implementar** una solución en una computadora de placa reducida Raspberry Pi modelo B+

- Se **mostró** el proceso de creación de colecciones de imágenes propias.
- Quedaron **descubiertos** varios elementos no documentados en las funciones empleadas.
- Se **mostró de manera indirecta**, el gran poder de representación que se logra con PCA
- Se **hicieron** herramientas para el análisis de algoritmos usando datos propios.
- Se deja material para **escribir** varios artículos.
- Se **hace disponible** al público todo el material empleado en este trabajo.
- Se **logró implementar** una solución en una computadora de placa reducida Raspberry Pi modelo B+

TRABAJO FUTURO

- Trabajar más con el algoritmo SVM.
- Mejorar la calidad de las imágenes.
- Enfocarse en la región de interés.

- Trabajar más con el algoritmo SVM.
- Mejorar la calidad de las imágenes.
- Enfocarse en la región de interés.



GRACIAS POR SU ATENCIÓN.

Gracias por su atención.