



INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

T E S I S

**"DESARROLLO DE APLICACIONES DE
RECONOCIMIENTO FACIAL CON BASE EN LOS
RESULTADOS DEL ANÁLISIS DE TRES FAMILIAS DE
ALGORITMOS"**

QUE PARA OBTENER EL GRADO DE:

**MAESTRÍA EN CIENCIAS EN INGENIERÍA DE
CÓMPUTO**

PRESENTA:

Ing. Germán Quiroz González

DIRECTORES DE TESIS:

**M. EN C. PABLO MANRIQUE RAMÍREZ
DR. EUSEBIO RICARDEZ VÁZQUEZ**



Ciudad de México

Septiembre 2022



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

SIP-13
REP 2017

**ACTA DE REGISTRO DE TEMA DE TESIS
Y DESIGNACIÓN DE DIRECTOR DE TESIS**

Ciudad de México, a 10 de agosto del 2022

El Colegio de Profesores de Posgrado del **Centro de Investigación en Computación** en su Sesión

(Unidad Académica)

Ordinaria No. 6 celebrada el día 30 del mes junio de 2022, conoció la solicitud presentada por el (la) alumno (a):

Apellido Paterno:	QUIROZ	Apellido Materno:	GONZÁLEZ	Nombre (s):	GERMÁN
-------------------	--------	-------------------	----------	-------------	--------

Número de registro: B 0 1 1 3 7 2

del Programa Académico de Posgrado: **Maestría en Ciencias en Ingeniería de Cómputo**

Referente al registro de su tema de tesis; acordando lo siguiente:

1.- Se designa al aspirante el tema de tesis titulado:

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

Objetivo general del trabajo de tesis:

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.

2.- Se designa como Director de Tesis a los profesores:

Director: **M. en C. Pablo Manrique Ramírez**

2° Director: **Dr. Eusebio Ricardez Vázquez**

No aplica: ☐

3.- El Trabajo de investigación base para el desarrollo de la tesis será elaborado por el alumno en:

Centro de Investigación en Computación

que cuenta con los recursos e infraestructura necesarios.

4.- El interesado deberá asistir a los seminarios desarrollados en el área de adscripción del trabajo desde la fecha en que se suscribe la presente, hasta la aprobación de la versión completa de la tesis por parte de la Comisión Revisora correspondiente.

Director(a) de Tesis

M. en C. Pablo Manrique Ramírez

Aspirante

C. Germán Quiroz González

2° Director de Tesis

Dr. Eusebio Ricardez Vázquez

Presidente del Colegio

Dr. Francisco Hiram Calvo Castro



INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México siendo las 11:00 horas del día 24 del mes de agosto del 2022 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Posgrado de Centro de Investigación en Computación para examinar la tesis titulada:

"Desarrollo de aplicaciones de reconocimiento facial con base en los resultados del análisis de tres familias de algoritmos" del (la) alumno (a):

Apellido Paterno:	QUIROZ	Apellido Materno:	GONZÁLEZ	Nombre (s):	GERMÁN
-------------------	--------	-------------------	----------	-------------	--------

Número de registro:

B 0 1 1 3 7 2

Aspirante del Programa Académico de Posgrado:

Maestría en Ciencias en Ingeniería de Cómputo

Una vez que se realizó un análisis de similitud de texto, utilizando el software antiplagio, se encontró que el trabajo de tesis tiene 2 % de similitud. **Se adjunta reporte de software utilizado.**

Después que esta Comisión revisó exhaustivamente el contenido, estructura, intención y ubicación de los textos de la tesis identificados como coincidentes con otros documentos, concluyó que en el presente trabajo SI ☐ NO ☒ **SE CONSTITUYE UN POSIBLE PLAGIO.**


JUSTIFICACIÓN DE LA CONCLUSIÓN: *(Por ejemplo, el % de similitud se localiza en metodologías adecuadamente referidas a fuente original)*
El 2% representa solamente las referencias a fuentes originales y frases comunes

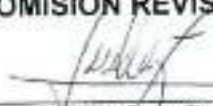
****Es responsabilidad del alumno como autor de la tesis la verificación antiplagio, y del Director o Directores de tesis el análisis del % de similitud para establecer el riesgo o la existencia de un posible plagio.**

Finalmente y posterior a la lectura, revisión individual así como el análisis e intercambio de opiniones, los miembros de la Comisión manifestaron **APROBAR** ☒ **SUSPENDER** ☐ **NO APROBAR** ☐ la tesis por **UNANIMIDAD** ☒ o **MAYORÍA** ☐ en virtud de los motivos siguientes:


Cumple con los requisitos para obtener el grado de Maestría

COMISIÓN REVISORA DE TESIS


M. en C. Pablo Manrique Ramírez
Director de Tesis


Dr. Luis Peñar Sánchez Fernández


Dr. Amador José Argüelles Cruz


Dr. Eusebio Riquelme Vázquez
2º Director de Tesis (en su caso)


Dr. José Giovanni Guzmán Lugo


Dr. José Juan Carrión Hernández


Dr. Francisco Hiram Calvo Castro
PRESIDENTE DEL COLEGIO DE
PROFESORES
DIRECCIÓN
IPN-CIC



INSTITUTO POLITÉCNICO NACIONAL SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA DE AUTORIZACIÓN DE USO DE OBRA PARA DIFUSIÓN

En la Ciudad de México el día 28 del mes de septiembre del año 2022, el que suscribe **Germán Quiroz González** alumno(a) del programa **Maestría en Ciencias en Ingeniería de Cómputo** con número de registro **B011372**, adscrito(a) a **Centro de Investigación en Computación** manifiesta que es autor(a) intelectual del presente trabajo de tesis bajo la dirección de **M. en C. Pablo Manrique Ramírez** y **Dr. Eusebio Ricardez Vázquez** y cede los derechos del trabajo intitulado **"Desarrollo de aplicaciones de reconocimiento facial con base en los resultados del análisis de tres familias de algoritmos"**, al Instituto Politécnico Nacional, para su difusión con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expresado del autor y/o director(es). Este puede ser obtenido escribiendo a las siguiente(s) dirección(es) de correo. gerquiroz@protonmail.com. Si el permiso se otorga, el usuario deberá dar agradecimiento correspondiente y citar la fuente de este.

Germán Quiroz González

Nombre completo y firma autógrafa del (de la)
estudiante

Resumen

Se presentan resultados del análisis de tres factores importantes en el diseño de un sistema de reconocimiento facial. Fueron seleccionados sobre la base del impacto en el resultado final, tal como la precisión, el tamaño de los datos de la aplicación, y la velocidad de respuesta.

Esto se refiere en primer lugar al algoritmo de detección de caras dentro de una imagen o flujo de video para formar una colección de imágenes sobre la cual aplicar el segundo factor, conocido como función de pérdida aplicado a las representaciones faciales obtenidas en una red convolucional; y, como tercer factor, el algoritmo de clasificación de dichas representaciones.

El trabajo se integró con código abierto y herramientas propias desarrolladas para implementar algunos pasos faltantes en las referencias utilizadas.

Se decidió abordar este tema debido a un trabajo previo en visión por computadora cuando se encontró que, para algo tan sencillo como la videovigilancia, con gran frecuencia es necesario adquirir equipos y programas ‘cerrados’. No se diga cuando se desea implementar una solución de reconocimiento facial. Se pretende aportar un documento que sirva como base para la implementación propia de esta tecnología con un alto desempeño.

Abstract

Results of the analysis of three important factors in the design of a facial recognition system are presented. These were selected because of their influence in the final results such as accuracy, application data size and response time.

At first, reference is made to an algorithm to detect faces within an image or video stream in order to get an image collection. These images are fed to a second factor known as loss function, applied to facial representations obtained from a convolutional network, which gets them apart in a high dimensional hyperspace. Then a third factor is applied to classify the representations.

The work is open source developed. Own coded tools helped to make an end to end solution for face recognition as a way to show the final result. It was decided to get into this topic because of previous work in computer vision after finding it difficult to make something as simple as personal video surveillance. You find it becomes necessary to buy closed systems which are impossible to work with. Let alone when trying to work with a face recognition application. The purpose of this work is to offer a way for others to get into this field in order to make their own high end tools.

Índice general

Glosario	9
1. Introducción	11
1.1. Motivación	12
1.2. Problemas a resolver	12
1.3. Justificación	13
1.4. Alcance	13
1.5. Hipótesis de trabajo	14
1.6. Objetivos	14
1.6.1. Objetivo general	14
1.6.2. Objetivos específicos	14
1.7. Metodología	14
1.8. Contribuciones del trabajo	15
1.9. Contenido del trabajo	16
2. Estado del arte	18
2.1. Introducción	18
2.2. Detección de caras	19
2.3. Funciones de pérdida	19
2.4. Clasificación	20
3. Marco teórico	22
3.1. Antecedentes del reconocimiento facial	22
3.2. El caso de una conocida implementación de reconocimiento facial	23
3.3. Algoritmos de detección facial	25
3.3.1. Cascadas de clasificadores débiles (Haar)	26
3.3.2. Histograma de gradientes orientados (dlib_hog)	27
3.3.3. Red neuronal convolucional de OpenCV (cv2_dnn)	29

3.3.4.	Red neuronal convolucional de dlib (<code>dlib_cnn</code>)	30
3.4.	Funciones de pérdida	31
3.4.1.	Distancia euclidiana	32
3.4.2.	Entropía cruzada	33
3.4.3.	Pérdida por tripleta	34
3.5.	Clasificadores	36
3.5.1.	PCA	37
3.5.2.	k -NN	38
3.5.3.	SVM	40
4.	Implementación	43
4.1.	Introducción	43
4.2.	Equipo de cómputo utilizado en este trabajo	43
4.3.	Elaboración de las colecciones de imágenes	44
4.3.1.	Caras de frente, de perfil y variantes	46
4.3.2.	Caras en imágenes de diferente resolución	52
4.4.	Algoritmos de detección facial	54
4.4.1.	Cascadas de clasificadores débiles (Haar)	54
4.4.2.	Histograma de gradientes orientados (<code>dlib_hog</code>)	62
4.4.3.	Detector facial con red neuronal de OpenCV (<code>cv2_dnn</code>)	66
4.4.4.	Detector facial con red neuronal de dlib (<code>dlib_cnn</code>)	72
4.4.5.	Estudio de tiempos de proceso de los algoritmos de detección respecto al área y cantidad de imágenes procesadas	76
4.5.	Funciones de pérdida	79
4.5.1.	Introducción	79
4.5.2.	Fragmento de código para procesar la red	83
4.5.3.	Resultados de las funciones de pérdida empleadas	84
4.6.	Clasificadores	91
4.6.1.	Introducción	91
4.6.2.	PCA	92
4.6.3.	k -NN	95
4.6.4.	SVM	96
5.	Evaluación de los resultados	98
5.1.	Introducción	98

5.2.	Método propuesto	99
5.3.	Omisiones	101
5.4.	Restricciones	101
5.5.	Mejora de los algoritmos o de su práctica	102
5.6.	Ejemplo de aplicaciones que se pueden desarrollar	103
5.7.	Comparación con otras soluciones	103
5.7.1.	Solución académica	103
5.7.2.	Solución comercial	105
5.8.	Evaluación global	106
5.9.	Problemas resueltos	108
6.	Conclusiones	109
6.1.	Introducción	109
6.2.	Conclusiones	109
6.3.	Recomendaciones	110
6.4.	Trabajo futuro	110

Índice de figuras

1.1. Esquema general de este trabajo.	16
2.1. Representación de la frecuencia de temas en los congresos CVPR. . . .	18
3.1. Un esquema de reconocimiento facial.	23
3.2. Diagrama de una implementación de reconocimiento facial.	25
3.3. Componentes básicos del algoritmo de Cascadas de clasificadores débiles.	26
3.4. Efecto de aplicar el algoritmo HOG a una imagen.	29
3.5. Arquitectura de la red tipo SSD en la que se basa OpenCV para su algoritmo de detección.	29
3.6. Arquitectura de la red tipo residual en la que se basa dlib para su algoritmo de detección.	30
3.7. Imágenes de dos personas muy parecidas [67].	35
3.8. Efecto de procesar tripletas.	36
3.9. Ilustración de los eigenvectores de un conjunto de datos.	38
3.10. Ejemplo de clasificación con el algoritmo k -NN, con $k = 3$	40
3.11. Líneas separando datos de dos clases.	41
3.12. Hiperplano óptimo mediante el mayor margen entre clases.	42
4.1. Las cuatro computadoras empleadas en este trabajo.	44
4.2. Aspecto de un programa para agrupar imágenes.	45
4.3. Aspecto del programa para seleccionar caras en imágenes.	45
4.4. Diez de las caras recortadas de la foto grupal de la Figura 4.3.	45
4.5. Ajuste de dimensiones debido al recorte.	46
4.6. Número de caras de frente y de perfil.	47
4.7. Detección automática de caras giradas a la derecha.	48
4.8. Detección automática de caras giradas a la izquierda.	48
4.9. Modelo i-bug de 68 marcadores faciales [4].	49
4.10. Ejemplos de caras clasificadas como perfil parcial.	49

4.11. Muestra de 10 caras de frente.	50
4.12. Caras ordenadas por ‘grado’ de inclinación.	50
4.13. Distribución de caras en las imágenes por ‘grado’ de inclinación.	51
4.14. Distribución de imágenes por ‘condición’ neutral, ocluida o gesticular.	52
4.15. Ejemplos de caras agrupadas manualmente.	52
4.16. Agrupamiento de imágenes de acuerdo al número de píxeles de alto.	53
4.17. Distribución de imágenes por resolución.	53
4.18. Diagrama de flujo de la detección.	55
4.19. Gráfica de correlación entre los parámetros de dimensión d y ventana mínima m sobre los resultados de detección s , n y tiempo t	56
4.20. Ejemplos de detección incorrecta.	57
4.21. Ejemplo de detección múltiple.	57
4.22. Ejemplo de cara no detectada.	57
4.23. Efecto de aumentar las dimensiones de la imagen.	58
4.24. Detección única sin modificar dimensiones originales.	58
4.25. Ejemplo de detección múltiple en ojos y boca.	59
4.26. Programa de agrupamiento manual de imágenes.	59
4.27. Comparación de detecciones y no detecciones.	60
4.28. Muestra aleatoria de diferentes condiciones.	61
4.29. Distribución de condiciones de caras no detectadas.	63
4.30. Cinco imágenes de cada categoría no detectada.	63
4.31. Dos imágenes tomadas al azar de la colección.	64
4.32. Ejemplo de cómo coinciden las coordenadas de las detecciones.	65
4.33. Efecto de aplicar el algoritmo de detección <code>dlib_hog</code> en 4 niveles de piramidación sobre una imagen redimensionada fuera del algoritmo.	65
4.34. Efecto del cambio de dimensiones en las detecciones con el soporte de redes neuronales de OpenCV.	67
4.35. Mosaico de imágenes incorrectamente detectadas sin escalamiento.	68
4.36. Detalle de anotaciones sobre una cara de perfil.	69
4.37. Mosaico de imágenes incorrectamente detectadas con escalamiento 2.0.	70
4.38. Imágenes no detectadas cambiando las dimensiones.	70
4.39. Caras no detectadas sólo en siete categorías.	73
4.40. Mosaico de imágenes de cada categoría no detectada.	74
4.41. Ejemplos de aparente detección doble con <code>dlib_cnn</code>	74
4.42. Algunas de las caras no detectadas por los otros algoritmos.	75

4.43. Estudio de tiempos de proceso de detección.	77
4.44. Distancia como medida auxiliar de convergencia.	79
4.45. Efecto de reducir un hiperespacio de 128D a 3D.	79
4.46. Diagrama de bloques de la red empleada.	81
4.47. Representacion tridimensional de la red empleada.	82
4.48. Grafo de cálculo de la red empleada.	82
4.49. Ejemplos de inestabilidad en las redes probadas.	86
4.50. Gráficas de rendimiento de la red.	87
4.51. Matriz de comparación. Las clases son: 0 para hombres y 1 para mujeres.	88
4.52. Estimación de clases sobre un lote aleatorio de la colección de prueba. .	90
4.53. Imágenes generadas con la transformada inversa de los datos originales.	93
4.54. 21 de las 22 imágenes generadas a partir de los 22 eigenvectores con 95% de varianza.	94
4.55. Resultados de la evaluación del algoritmo PCA, con $k = 3$	94
4.56. Resultados del algoritmo k -NN, con $k = 3$	95
4.57. Resultados del algoritmo SVM.	97
5.1. Diagrama de bloques del método propuesto para la tarea de reconocimiento facial.	99
5.2. Ejemplos de aplicaciones logradas en este trabajo.	103
5.3. Tablero de incidencias durante el día.	106

Índice de tablas

1.1. Algoritmos a comparar.	15
3.1. Cómo las redes convolucionales tratan de imitar el modelo biológico del campo visual felino.	23
3.2. Clasificadores analizados en esta sección.	37
3.3. Vecinos más cercanos para tres clases.	39
4.1. Resumen de las dimensiones de la colección para la detección facial. . .	53
4.2. Nombres cortos de los algoritmos.	54
4.3. Parámetros de la función <code>detectMultiScale</code>	55
4.4. Resultados de las 60 pruebas de detección	56
4.5. Resultados de aplicar el algoritmo <code>dlib_hog</code> a la colección de imágenes. .	62
4.6. Coordenadas de los puntos inicial y final de los rectángulos de detección facial usando el algoritmo <code>dlib_hog</code>	64
4.7. Funciones del módulo <code>dnn</code> de <code>OpenCV</code>	66
4.8. Resumen de resultados del algoritmo <code>dlib_cnn</code>	73
4.9. Comparación de algoritmos de detección facial.	75
4.10. Resumen de la red empleada.	80
4.11. Fragmento del resumen de la red con entrada de 180 x 180 píxeles. . . .	81
4.12. Detalles de la colección para funciones de pérdida.	85
4.13. Resultados en pérdida y ganancia de los tres métodos analizados. . . .	85
4.14. Detalle de las imágenes de prueba.	88
4.15. Reporte de clasificación.	89
4.16. Resultados de búsqueda por autor y algoritmo.	91
4.17. Relación de imágenes por identidad.	92
4.18. Resumen de resultados por algoritmo.	97
5.1. Algoritmos que se sugiere emplear en la tarea de reconocimiento facial. .	100
5.2. Variación de resultados al usar diferentes versiones de bibliotecas. . . .	102

5.3.	Comparación entre dos soluciones de índole doméstico.	105
5.4.	Comparación entre una solución comercial y la aquí propuesta.	105
5.5.	Resumen de resultados asociados a los Objetivos específicos.	107
5.6.	Solución a los problemas planteados en la Sección 1.2	108

Glosario

Adam

Del inglés *Adaptive moment estimation* o estimación de momento adaptativo. Algoritmo para la optimización basada en gradiente de primer orden de funciones objetivo estocásticas, basadas a su vez en estimaciones adaptativas de los momentos de orden inferior [42].

Amaxayac

Programa de reconocimiento facial de uso personal. El nombre proviene del náhuatl, que significa máscara.

Archivo de comas

Archivo tipo .CSV o de *comma separated values*. Literalmente, archivo de valores separados por coma. Se decidió usar este término para evitar el uso del acrónimo o su traducción.

CVPR

Computer Vision and Pattern Recognition Conference. Conferencia sobre visión por computadora y reconocimiento de patrones.

Descriptor

Cuando se procesan imágenes buscando patrones se identifican ubicaciones donde ocurren cambios bruscos en intensidad, entre otros factores. En estos lugares se aplican diferentes transformaciones y cambios de escala (piramidación) para observar su invariabilidad, y, en ese caso, formar parte de una entidad llamada descriptor, la cual, entre otras cosas, incluye la dirección del cambio y su magnitud.

Eigenvalor

Valor propio de un eigenvector o coeficientes de los eigenvectores que representan la cantidad de varianza de cada componente principal en un conjunto de datos.

Eigenvector

Cuando se usan en análisis de componentes principales o PCA, se refiere los vectores propios de la matriz de covarianza del conjunto de datos que representan las direcciones de los ejes donde hay más varianza y, por lo tanto, más información.

Error tipo 1

En prueba de hipótesis estadística, se refiere al **rechazo** de la hipótesis nula. Esta última propone que no hay diferencias entre ciertas características de una población.

Error tipo 2

En prueba de hipótesis estadística, se refiere a la **aceptación** de la hipótesis nula. Esta última propone que no hay diferencias entre ciertas características de una población.

Grafo de cálculo

Es un objeto de TensorFlow que contiene una estructura de datos con un conjunto de objetos tipo `tf.operation` que representa unidades de cálculo y objetos tipo `tf.tensor` que representan unidades de datos que fluyen entre las operaciones [32].

HOG

Del inglés *Histogram of Oriented Gradients* o histograma de gradientes orientados. Técnica estadística desarrollada por Robert K. McConnell en 1986 [51].

Pérdida progresiva de ancla

Del inglés, *Progressive anchor loss*. Es un algoritmo que combina diferentes tamaños en los varios niveles de una pirámide de imágenes de dos tomas distintas. Se toman menores tamaños de ancla de la primera toma y tamaños más grandes de la segunda.

Región de interés

Del inglés *Region of interest (ROI)*, se refiere al área de un rectángulo que encierra a un objeto que se está analizando y cuyo perímetro es resaltado.

SVM

Del inglés *Support Vector Machine (SVM)*. Máquina de soporte vectorial. Es una técnica de separación de clases mediante un hiperplano, desarrollada por Vladimir Vapnik en 1980.

Capítulo 1

Introducción

El reconocimiento facial puede considerarse un tema resuelto cuando éste alcanza una exactitud casi de 100% (99.939) [36] en ambientes controlados. Sin embargo, estas aproximaciones presentan problemas al identificar personas que, por su género u origen étnico, pueden ser no reconocidas correctamente en el mejor de los casos o, en casos más problemáticos, pueden ser confundidas con otra persona. Por estas razones el reconocimiento facial sigue siendo un tema de discusión cuando se abordan cuestiones tales como privacidad, seguridad y libertades civiles.

En este trabajo se acota el significado del reconocimiento facial como sigue: se parte de un sistema en el que se registran imágenes del grupo de personas que se pretende reconocer. Las imágenes son procesadas de tal forma que sus representaciones vectoriales son almacenadas en algún repositorio. En este caso, cada representación vectorial es un arreglo de 128 enteros de un byte. Cada vector es clasificado de forma que los que pertenecen a una misma identidad, están más juntos unos de otros en un hiperespacio de 128 dimensiones. Otras identidades también forman cúmulos, pero un mecanismo de agrupación trata de mantener cada cúmulo lo más separado de los otros. Cuando se quiere reconocer a una persona previamente registrada, otra parte del sistema se encarga de detectar una cara en una imagen fija o capturada de un flujo de video. Un encuadre de la cara detectada pasa por un proceso en el que se calcula su representación vectorial, y aquella se compara con los cúmulos o clases dentro del repositorio. Al no ser el reconocimiento un cálculo determinístico sino probabilístico, el sistema produce un arreglo de probabilidades y reporta como identidad el elemento con más alta probabilidad. Como todo esto ocurre en tiempo real, el sistema puede reportar varias identidades en un segundo debido a que se está en un ciclo continuo de detección y cálculo, y la persona puede estar moviendo su cabeza. Sin embargo, mediante un pequeño ajuste al programa se ha logrado reportar la identidad que más ‘votos’ obtiene en ese segundo. Una alternativa es fijar un umbral mínimo. Quienes hayan utilizado un sistema de control de acceso basado en reconocimiento facial, habrán experimentado que el sistema no ‘reconoce’ a la persona. Sistemas de ese tipo están programados para reportar una identificación positiva con un alto margen de certeza, ya que el fabricante prefiere no dejar pasar a un ‘conocido’ que dejar pasar a un ‘desconocido’. Otra forma de reconocimiento, esta vez en lote, se muestra en la sección 4.6, donde se puede ob-

servar que, para el conjunto de prueba (65 imágenes pertenecientes a 9 identidades), se obtienen resultados arriba del 73%. Como se aborda un estudio académico y no un sistema comercial, se considera suficiente para probar los objetivos planteados. Durante la defensa de este trabajo se presentará el programa Amaxayac de reconocimiento facial en tiempo real, con su interfaz gráfica en una computadora portátil, así como una versión de consola que se puede implementar en un microcontrolador.

1.1. Motivación

El interés por el tema surgió al trabajar con visión por computadora para detectar movimiento mediante la medición de cambios en un flujo de video. Después de experimentar con este tema y definiendo regiones de interés para rastreo de objetos en movimiento, el paso siguiente fue el reconocimiento facial.

1.2. Problemas a resolver

Se pretende dejar un trabajo en el que se puedan apoyar quienes se aproximan al problema de reconocimiento facial. El tema ha estado en los medios de comunicación por algún tiempo, al menos cinco años, y por un lado se le conoce como una herramienta de seguridad, como un mecanismo de toma de datos biométricos, o como un fenómeno de índole policiaco. Desde hace muchos años las fuerzas policiales han tomado fotografías de gente que se opone a los gobiernos. Uno de los estudios referidos en el marco teórico de este trabajo así lo indica. Ahora se sabe del uso de drones que sobrevuelan las manifestaciones para recabar datos de opositores y, en muchos casos, identificarlos en tiempo real. El tema de este trabajo tiene muchas aplicaciones y se espera que el lector le encuentre un uso no invasivo.

Considerando sólo un par de aplicaciones de reconocimiento facial, es posible enumerar los problemas que se pueden resolver.

1. Los sistemas de control de acceso y videovigilancia son cerrados.
2. No hay forma de acceder directamente a sus registros.
3. No hay forma de agregar funcionalidad.
4. No son transparentes en sus resultados (por ejemplo, razones por las que se niega el acceso a alguien debidamente registrado).
5. No ofrecen un canal de comunicación directo con el usuario.
6. La información relevante para producir un sistema de control de acceso desde cero es inaccesible o inexistente.

De los problemas que afectan a todo el espectro de reconocimiento facial, podemos destacar:

1. Hay información que no ha sido incluida en la documentación de funciones relevantes en esta materia.
2. No se conocen detalles sobre la implementación, tales como dimensiones de una aplicación, plataformas de despliegue, lenguajes de programación, bases de datos, colecciones de imágenes, velocidad de respuesta, precisión, complejidad y otros factores.
3. No es claro que hay otras aplicaciones en campos tales como: entretenimiento, sistemas de recomendación personalizados y realidad aumentada.

1.3. Justificación

Los trabajos científicos sobre reconocimiento facial no abordan de manera integral los diversos factores que están involucrados en este tema. Se enfocan en aquella pequeña parte (no por eso menos importante), en que su trabajo mejora el estado del arte de ese momento. El presente trabajo es un estudio de tres familias de algoritmos que influyen de manera determinante en el tema de este trabajo. Tales familias de algoritmos se muestran de manera armónica y exhaustiva para que el interesado encuentre elementos suficientes con los que pueda iniciar su propia investigación sobre el tema.

1.4. Alcance

Exponer lo que la documentación formal de las herramientas no contempla. Cada herramienta presenta una familia de funciones envuelta una sobre otra como capas en una cebolla, o como una cascada de funciones donde la primera se escurre en la segunda, y estas dos en la tercera, desde el punto de vista de un programador. Cuando se logra 'deshojar' o separar esas capas y exponer el interior, se descubren cosas que no están en la documentación. A veces lo que se descubre es de manera indirecta. Este trabajo presenta lo que se encontró al trabajar con esas funciones anidadas y que no considero la documentación.

1.5. Hipótesis de trabajo

El investigador puede controlar completamente su entorno de análisis cuando crea sus propias herramientas, incluyendo conjuntos de datos, y tiene elementos para elegir otros recursos disponibles. En este trabajo se muestra cómo hacerlo¹ cuando se trabaja en reconocimiento facial por computadora.

1.6. Objetivos

A continuación se establecen los objetivos general y específicos de esta tesis.

1.6.1. Objetivo general

Utilizar tres técnicas que, al funcionar unidas, permiten desarrollar aplicaciones que mejoran la velocidad de respuesta² en reconocimiento facial, enfocándonos en equipos de bajo rendimiento.

1.6.2. Objetivos específicos

- i) Encontrar el mejor detector facial, en términos de eficiencia y velocidad, de los cuatro analizados aquí.
- ii) 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 que contienen la cara de una persona.
- iii) Descubrir el mejor algoritmo de clasificación de representaciones vectoriales faciales de los tres analizados aquí.
- iv) Mostrar que se pueden hacer colecciones de imágenes y herramientas propias para nuevas investigaciones y para reproducir resultados de otros estudios.

1.7. Metodología

Se crean varias colecciones de imágenes (Objetivo iv) para aplicarles tres familias de algoritmos, cada una perteneciente a la detección de caras, representación vectorial de caras y clasificación facial.

Las colecciones son recopiladas mediante el proceso de documentos en formato .pdf, imágenes tomadas de Internet y capturadas desde un flujo de video mediante el programa Amaxayac, de uso personal.

¹Una vez aprobada esta tesis, quedará disponible el material: código, imágenes, textos traducidos, requiriendo un período de curación del mismo, en <https://github.com/germaKonda>

²Respecto a cualquier aplicación que realice menos de 6 estimaciones por segundo.

Tabla 1.1: Algoritmos a comparar.

Apartado	Nombre	Propósito	Autor	Año
Detección de caras	Cascadas de clasificadores débiles [72] y [26]	Detección de caras en una imagen	Viola y Jones	2001
	Histograma de gradientes orientados [51], [17] y [20]	Descriptor de características	Robert K. McConnell	1986
	Red neuronal convolucional [38] y [40]	Detección de caras en una imagen	Davis E. King	2009
	Red neuronal profunda [57]	Detección de caras en una imagen	OpenCV	2016
Función de pérdida	Distancia euclidiana [10]	No considera las distancia intra e interclase	—	—
	Entropía cruzada [13] y [63]	Crear representaciones castigando la intraclase	Reuben Y, Rubinstein	1997
	Pérdida por tripleta [61]	Las representaciones de clases iguales se acercan y las distintas se alejan	FaceNet	2015
Clasificación	PCA [58]	Reduccion de dimensiones por análisis de covarianza	Karl Pearson	1901
	k -NN [21] y [14]	Clasificación por vecindad	N.S. Altman	1991
	SVM [3], [76], [27] y [71]	Clasificación por hiperplano óptimo	Vladimir Vapnik	1982

Se lograron tres colecciones cuyas características se explican más adelante. Cada una de ellas se usó para uno o dos de los apartados de este trabajo. Por ejemplo, la colección de detección se usó también para la de funciones de pérdida. En la clasificación sólo se utilizaron las imágenes capturadas para ese apartado. La comparación de algoritmos se hizo de acuerdo a la Tabla 1.1. Además, para realizar análisis de tiempo de ejecución de los detectores de la Sección 4.4.5, se reunieron otras 1,457 imágenes con el único propósito de aumentar el número de imágenes a utilizar en dicha medición.

1.8. Contribuciones del trabajo

Se produjo una amplia colección de programas para reproducir y validar los resultados, lo que puede servir como punto de partida para otros emprendimientos y como ejemplo de implementación del cuerpo teórico en esta tesis. Se recopilieron tres colecciones de imágenes propias para cada uno de los temas y también se recabó una amplia colección de referencias en forma de artículos, presentaciones, tesis doctorales y libros en formato .pdf, cuya lista se puede entregar a petición. Muchos de los resúmenes de los artículos leídos fueron traducidos y, en varios casos, se tradujo el artículo completo por su relevancia.

1.9. Contenido del trabajo

La Figura 1.1 ilustra, en términos generales, el contenido de este trabajo. Pueden observarse los cuatro objetivos dentro de los círculos verdes, los tres grupos de estudio (círculos de mayor diámetro) y los 10 algoritmos a analizar. Abajo se encuentran los resultados, dos comparaciones y las conclusiones con las que se termina esta tesis.

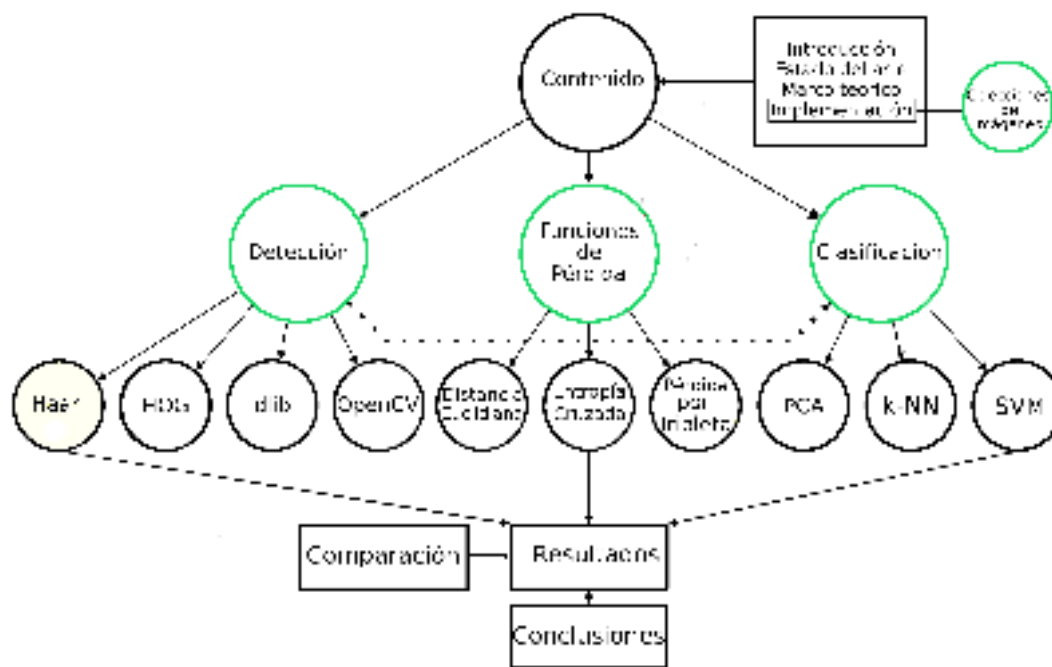


Figura 1.1: Esquema general de este trabajo.

En este primer capítulo se presenta una introducción al reconocimiento facial abordando varias ideas o enfoques que los investigadores consideraron adecuadas pero que, al no alcanzar los niveles de precisión actuales, fueron dejando de ser investigadas y favoreciendo otras aproximaciones de las que aquí se hablará. Aquí también se encuentra la justificación del trabajo y la hipótesis que se considera la base de éste. Así mismo, se encuentran los objetivos específicos que se plantearon para el desarrollo del trabajo.

En el Capítulo 2 se explica qué es lo que se está haciendo en materia de reconocimiento facial en general, y en particular en los tres temas de este trabajo: detección, funciones de pérdida y clasificación.

El marco teórico del Capítulo 3 servirá para explicar cada uno de los diez algoritmos analizados. Se presentan referencias a demostraciones formales de los conceptos matemáticos involucrados.

Durante el Capítulo 4, dedicado a la implementación, se presentarán los resultados de diversos programas utilizados en esta investigación. En cada tema se hará una pausa para mostrar detalles, gráficas y ejemplos, además de señalar aquello que no está documentado.

En el Capítulo 5, dedicado a la evaluación de resultados, se mostrarán gráficas y tablas como formas de visualización. En la Tabla 5.5 se podrá ver un resumen de los

objetivos de este trabajo, enumerando los resultados de cada algoritmo involucrado y destacando aquellos que mejor rendimiento mostraron. También se incluye la Tabla 5.6 con una respuesta a los problemas a resolver planteados en la Sección 1.2

En el Capítulo 6 se presentan conclusiones, recomendaciones y propuestas de trabajo futuro para los lectores. Finalmente se ofrece la lista de referencias citadas en esta tesis, pero que son sólo una parte de todo el material investigado, mismo que se pone a disposición de quien lo solicite.

- 1) Detección de caras
- 2) Funciones de pérdida
- 3) Clasificación

2.2. Detección de caras

Xiang Ming et al. [53] demuestran que no es necesario hacer una predicción facial en múltiples capas de la red. Concluyen que se puede hacer una predicción exitosa con una sola capa, a condición de tener un balance en las muestras de acondicionamiento tanto positivas como negativas en diferentes escalas. El trabajo de Xian Ming está orientado al proceso de imágenes fijas con muchas caras. Uno de los algoritmos usados por Xiang Ming es el de ‘región propuesta’. Esto quiere decir que la red selecciona una región y la propone al siguiente paso para detectar allí la cara.

Mahyar Najibi et al. [54] implementan una técnica llamada ‘Región flotante de propuesta para detección facial’ con la que pueden detectar caras en áreas tan pequeñas como 5 x 5 píxeles. Para lograrlo, evitan usar directamente los mapas de características generados con la aplicación de filtros a los píxeles, pero agrupan regiones propuestas en lugares específicos para reducir el número de regiones a procesar. Con su estrategia pueden colocar regiones fraccionadas e intercambiarlas sin hacer cambios al modelo calculado.

Jian Li et al. [47] proponen un método para generar regiones propuestas. Éstas consisten en plantillas de miles de rectángulos en una imagen. En tales rectángulos se busca una cara en base a un tamaño y forma de objeto y calculando cuál caja circundante tiene la más alta relación entre traslapes y no traslapes. Si la relación supera cierto valor, se ha detectado un objeto. En su trabajo se extiende el detector de imagen única a imagen doble. Luego aplican un algoritmo de pérdida progresiva de ancla para inicializar mejor la función de regresión, ya que típicamente se inicializa con valores aleatorios. Además, esto ayuda a no ignorar lo que ocurre entre las capas y entre cada una de las dos imágenes.

2.3. Funciones de pérdida

Jonathan T. Barron [5] presenta una función de pérdida general y adaptativa. A diferencia de otros tipos de función de pérdida con un solo parámetro que se tiene que ajustar de manera manual, su función requiere dos parámetros que se autorregulan. Demuestra que su función es un superconjunto de otras funciones de pérdida que incluyen la de Couchy y las conocidas L1 y L2. T. Barron Hace pruebas específicas en las tareas de visión por computadora conocidas como registro, agrupamiento, síntesis generativa y estimación de profundidad. En cada caso demuestra que su función mejora el desempeño de las redes originales en las que basa su estudio y demuestra también que este resultado se puede llevar a otras redes. La red de referencia que utiliza para

estimación de profundidad es la conocida como DispNet [52], trabajando en el banco de pruebas KITTI [23] orientada al desarrollo de vehículos autónomos.

Guillermo et al. [22] presentan veintidós funciones de pérdida focal orientadas a resolver un problema nuevo con herramientas maduras y probadas. Con antecedentes en estadística y cálculo, demuestran que usando varianza, gradiente y magnitud del laplaciano aplicado a datos en el espacio (x, y, t) , donde t es tiempo, se logran los mejores resultados en aplicaciones de estimación de profundidad y flujo óptico, segmentación y reconocimiento de objetos, entre otras. Se utiliza una cámara de eventos que, a diferencia de las tradicionales (de cuadro), es asíncrona, de alto rango dinámico e inmune a la distorsión por movimiento de la cámara o el objetivo. Las cámaras de evento capturan diferencias de brillo de cada uno de sus píxeles a velocidades de microsegundos. Por cada evento (cambio de brillo) se almacena la marca de tiempo, la intensidad y la polaridad (bajó o creció el brillo). Es posible imaginar un perfil rectangular que crece en el tiempo. Por cada instante de tiempo se activan en el rectángulo sólo aquellos píxeles que cambiaron de intensidad. Poniendo todos los rectángulos en una línea de tiempo podría imaginarse un canal de perfil rectangular. Los algoritmos que diseñan cada una de las veintidós funciones de pérdida focal logran formar trayectorias de puntos que, a fin de cuentas determinan una silueta. Dependiendo de la función de pérdida, la silueta puede ser más o menos nítida. Las funciones de pérdida que mejor se desempeñan son las mencionadas al principio, pues producen la silueta mejor definida. Un factor sobresaliente de estas cámaras es que el destello que las cámaras tradicionales captan con una luz de frente, no ocurre aquí. Ésta y otras características pueden hacer que esta tecnología tenga un futuro sobresaliente en visión por computadora.

Jiankang Deng et al. [19] presentan ArcFace, que es una técnica de función de pérdida que se monta sobre la última capa de red completamente conectada (como ResNet-50 [28]), que genera incrustaciones en forma de vectores en los que cada elemento es una dimensión. ArcFace usa 512 dimensiones y llama a este vector ‘incrustación de la característica’. Cada imagen genera un vector. Toma como ejemplo los trabajos en SphereFace [50] y CosFace [74] para mejorarlos mediante lo que llaman ‘función de pérdida de margen angular aditivo’, que en otras palabras suma cierto valor al ángulo formado entre el vector corriente y el resto de los vectores. Dependiendo del ángulo es el margen sumado. El resultado es que los vectores de objetos de la misma clase quedan más juntos (compactos, para los autores) y la distancia angular hacia objetos de clases diferentes es mayor. No pierde la esencia de cualquier función de pérdida, pero en este caso utiliza el concepto de distancia geodésica (arco) entre vectores. Utiliza las redes ResNet-50 y ResNet-100 [28] sobre una amplia variedad de colecciones de imágenes, incluyendo algunas que contienen más rangos de edad, etnias y video. Ofrecen código fuente para dar continuidad a su estudio.

2.4. Clasificación

Qiuyu Chen et al. [7] desarrollan un ‘algoritmo de incrustación para lograr más altas tasas de precisión en clasificación de imágenes a gran escala’ mediante el cual se entrenan ‘múltiples redes complementarias de manera secuencial’. Se enfocan en el

hecho de que las redes tradicionales y de reforzamiento le dan la misma importancia a los objetos fáciles de identificar que a los objetos difíciles de identificar. Como ejemplo, considérese el caso de confundir un perro con un rascacielos contra confundir un perro husky con un pastor alemán. Por esta razón, su algoritmo trabaja en el reforzamiento de su habilidad discriminatoria procesando múltiples redes secuenciales que convergen en características que van de las clases de objeto más fáciles de reconocer a los que son más difíciles. Su algoritmo es probado con tres colecciones de imágenes y seis arquitecturas de red.

En el artículo de Yann Lifchitz et al. [48] se propone una solución en la que usan una red existente (VGG-16) y le implantan capas distribuidas estratégicamente para trabajar en dos flujos de datos. En el primero y base, se obtienen mapas de características convencionales. Tales mapas son alimentados a la red implantada para ejercer sobre ellos una segunda etapa de proceso en la que se aplican otros filtros a una cuadrícula más densa. Dichos filtros están diseñados para obtener otra serie de características sobre las cuales se ejecuta la tarea de encontrar nuevas clases de objetos que no son localizados por la red base. Cuando un mapa de características es derivado hacia la red implantada, su proceso en la red actual se congela y es incorporado a la salida de su capa equivalente en la red secundaria. De esta forma se evita que el mapa de actividad de la red original sea ‘aplanado’ y procesado en la siguiente capa. En vez de eso, se analiza en conjunto con el mapa producido en la misma etapa de la red secundaria. Su trabajo presenta un análisis detallado de su estrategia y ofrece código fuente para su estudio.

En Tong He et al. [29] la modificación interna de algunas redes existentes permite lograr mejoras en su desempeño sin necesidad de hacer una nueva red o cambiar la arquitectura de otra. Entre las modificaciones realizadas destacan: tasa de aprendizaje programada, en la que dicha tasa varía con base en un rango de épocas; sustitución de dimensiones de filtros de convolución, por ejemplo, de 7×7 con paso (*stride*) 2 a tres de 3×3 con paso 1; derivación de flujo de datos a otro camino de filtros de convolución y suma de mapas de activación a la salida; cambio en el paso de aplicación de filtros. Como ventajas de su aproximación destacan: la utilización de sus mejoras en otras redes y la transferencia de convergencia que permite pasar del dominio de clasificación de imágenes al de detección de objetos y segmentación semántica. Los autores presentaron su trabajo con el título *Bag of Tricks for Image Classification with Convolutional Neural Network* [29] en 2018, y en 2019 presentaron una variante con el título de *Bag of Freebies for Training Object Detection Neural Networks* [77], en el que trabajaron con otras redes, otras colecciones de imágenes, e incluyeron proceso de video. Los autores ofrecen el código fuente.

Para organizaciones como MegaFace [36] el asunto del reconocimiento facial es un caso resuelto debido a que, en su último reto, una empresa privada logró una precisión de 99.939%. Aunque el reto es demandante, ciertamente es acotado. En 2015 se logró 99.63%, también por parte de una empresa [61]. En 2016, un trabajo académico [2] logró 92.92%. A la fecha (CVPR 2020) puede observarse que un trabajo sobre Convergencia de pérdida curricular adaptativa [31] alcanza una precisión de 99.80%; y el otro sobre Redes de atención diversa de jerarquía piramidal [75] logra 99.25%.

Capítulo 3

Marco teórico

3.1. Antecedentes del reconocimiento facial

Con el trabajo de Woody Bledsoe [6] se hizo el proceso híbrido de fichas policíacas a razón de 40 imágenes por hora. Híbrido porque se usaba una tableta electrónica sobre la que se hacía un punteo manual. Otro ejemplo de este trabajo se encuentra en la tesis doctoral de Takeo Kanade [34], con la que se logró identificar 15 de 20 personas en un conjunto de más de 800 imágenes. Aquí también se usó una tableta para el punteo manual de marcadores faciales. Los puntos se usaron para hacer mediciones entre ellos, de manera similar a como lo hacía un analista con mediciones manuales.

Se puede decir que el reconocimiento facial es una rama especializada dentro de otra más general conocida como reconocimiento de patrones. En ambas se han intentado muchas aproximaciones holísticas (tomando la imagen como un ente único y global) de las cuales se pueden calcular representaciones tales como histogramas, varianzas y transformadas de Fourier, entre otras. Sin embargo, todas ellas han sido rebasadas por un enfoque diferente. Este enfoque empezó con el trabajo de los neurocientíficos David Hubel y Torsten Wiesel [24], quienes observaron que ciertas neuronas individuales respondían cuando ciertas imágenes aparecían en regiones específicas de una pantalla puesta frente a un gato. Esta aproximación se refiere al concepto de red convolucional. Simplificando, se puede decir que dicha red trata de capturar tres propiedades de una región del cerebro llamada V1 o corteza visual primaria. Una comparación entre el aspecto biológico y su contraparte convolucional se observa en la Tabla 3.1

Los primeros resultados sobresalientes del uso de redes convolucionales pueden encontrarse en el trabajo de Yan Lecun et al. [46]. Aquí puede verse que mediante la red LeNet-5 y algunas variantes, se logró implementar una solución en hardware mediante la cual se procesaron diariamente millones de cheques bancarios. La aplicación, si no resulta obvia, es la de reconocer las cantidades numéricas escritas a mano o a máquina. Fue cuestión de tiempo que esta solución fuera adoptada para reconocer cualquier otro objeto en una imagen, incluidos los rostros humanos. Se recomienda ampliamente la lectura del artículo de LeCun et al. [46].

Tabla 3.1: Cómo las redes convolucionales tratan de imitar el modelo biológico del campo visual felino.

Prop.	Modelo biológico	Modelo convolucional
1	La región V1 está dispuesta como un mapa espacial de dos dimensiones en espejo de la estructura de la imagen en la retina.	Se captura esta propiedad mediante la definición de mapas de activación en dos dimensiones.
2	V1 contiene muchas células simples que se pueden caracterizar por una función lineal de la imagen en un campo receptivo pequeño espacialmente localizado.	Los filtros utilizados están diseñados para emular las propiedades de esas células simples.
3	V1 también contiene células complejas que responden a características que son invariantes a pequeños desplazamientos en la posición del objeto.	Se tienen unidades de agrupamiento que son invariantes a cambios de luz que no pueden ser capturados simplemente agrupando espacios. Como ejemplo, agrupamiento de canal cruzado ^a o por maximización.

^aEs una técnica de reducción de parámetros en redes convolucionales [25].

3.2. El caso de una conocida implementación de reconocimiento facial

Se usará un par de diagramas tomados del trabajo de OpenFace [2] para ilustrar una de las muchas posibles implementaciones de reconocimiento facial usando redes convolucionales:

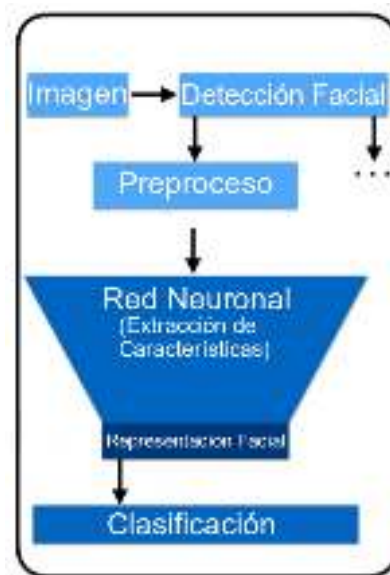


Figura 3.1: Un esquema de reconocimiento facial.

En la Figura 3.1 se observa que **1)** Hay un objeto de entrada. Este puede ser una imagen o un flujo de video. En cualquier caso, la entrada debe contener una cara. **2)** La imagen es convertida a píxeles y al sistema entra una matriz de dimensiones fijas, pero que puede variar según convenga. **3)** La imagen pasa por un proceso específico y especializado en detección de caras. Si en el objeto de entrada no hay una cara, el sistema continúa analizando lo que se le presente y no produce ningún resultado mientras no haya una cara. Es posible que sí haya una cara, pero el detector puede no ser suficientemente robusto para encontrarla, o bien, las condiciones de iluminación dificultan la detección (Objetivo 1)). Las caras pueden estar alineadas según un patrón predefinido o se alinean al vuelo. La cara original puede no estar alineada pero el resultado de este paso entrega la imagen con una cara alineada. La cara resultante puede tener alguna deformación, pero es despreciable para efectos prácticos. **4)** La matriz pasa entonces a un proceso de generación de una representación vectorial en 128 dimensiones. Para ello se usa un modelo de red convolucional precondicionado que produce dicha representación como en el Apartado 4.6.4. **5)** Posteriormente se hace una comparación mediante un clasificador que mantiene registro de todas las identidades dadas de alta previamente y se entrega algún resultado en forma de vector ponderado. El valor con mayor peso se sugiere como la identidad de la persona procesada. De lo anterior se desprende que el reconocimiento facial no se resuelve de manera determinística sino probabilística.

Un modelo como el anterior puede usarse como base en alguna de estas aplicaciones:

- a) Permitir el acceso a personas autorizadas.
- b) Identificar una persona dada una imagen.
- c) Identificar si alguna persona pertenece a algún grupo de personas diferentes con las que comparte rasgos.
- d) Con algunas modificaciones, pueden aplicarse máscaras de deformación para efectos de entretenimiento.
- e) Con el soporte de una base de datos, se puede implementar un sistema de recomendación/atención personalizada o un sistema de detección personas problemáticas.

Para lograr cualquiera de estas aplicaciones es necesario que antes ocurra lo que se muestra en el diagrama siguiente:

En la Figura 3.2 se observan dos paneles. El de la derecha representa el proceso de una red definida e implementada en la plataforma Torch [11]. El modelo de red es el conocido como *Inception* [66]. En el panel derecho hay una implementación completamente desarrollada en Python. Ahora se describen las partes de cada panel. En el primero se observa **1)** Cierta cantidad de imágenes que serán procesadas por la red. **2)** La red del tipo mencionado produce representaciones faciales de 128 dimensiones. **3)** A esta representación se le aplica una función de pérdida llamada ‘pérdida por tripleta’

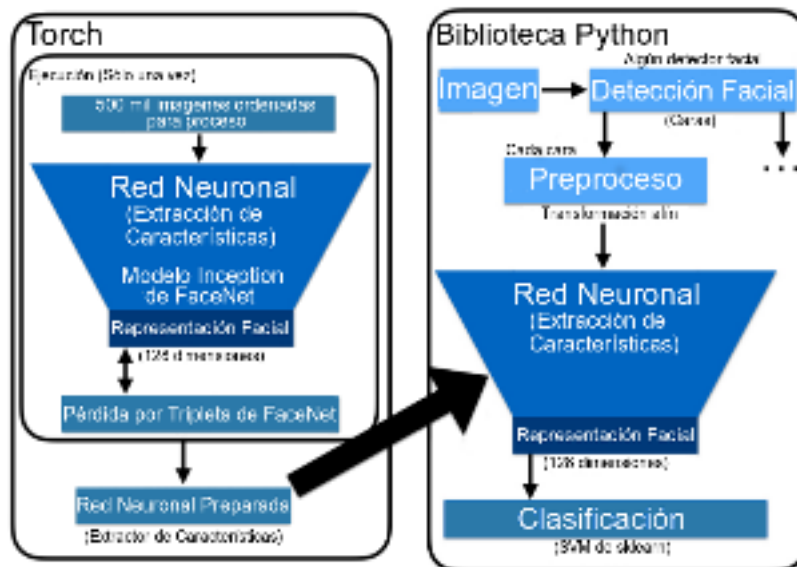


Figura 3.2: Diagrama de una implementación de reconocimiento facial.

con el propósito de crear una fuerte separación entre clases diferentes y una mayor cercanía entre miembros de la misma clase. **4)** El resultado es un archivo de pesos (Red neuronal preparada) que se carga en el bloque correspondiente en el panel de la derecha (Extracción de características) y se procede como se menciona en el punto 4 de la Figura 3.1 (Objetivo **II**). Ahora se desarrollarán los diez conceptos mencionados en la Tabla 1.1.

3.3. Algoritmos de detección facial

Se mostrará que las imágenes procesadas de cierta manera permiten detectar objetos en ellas. Los objetos, siendo entidades de alto nivel de abstracción, son compuestos de entidades que pueden llamarse características. En una imagen, prácticamente cada píxel es una característica si se considera que tiene varios ‘valores’, por ejemplo, su ubicación espacial y su valor de color tomado como una tripleta de sus componentes de rojo, verde y azul. Pero considerar cada píxel como una característica no dice mucho del objeto presente en la imagen. Para ello procesamos la imagen con algunos ‘filtros’ que permiten detectar bordes. Los bordes también están en lugares específicos y dan otra información tal como su orientación en la imagen. De aquí es posible saber si son bordes curvos, planos o en forma de esquina. Reuniendo cierta cantidad de estos y haciendo una piramidación de la imagen, podemos saber si estos bordes son invariantes a la escala y a la translación. Con ello, a grandes rasgos, se obtiene una colección de descriptores que no dicen si el objeto es un plato o la cara de una persona, pero que sí, definitivamente, nos dice que hay algo. A partir de este ejercicio básico de proceso de una imagen, y mediante otros más complejos, los investigadores como Viola y Jones [72] han logrado detectores de objetos en general, y de caras en particular con una velocidad y precisión sobresalientes.

3.3.1. Cascadas de clasificadores débiles (Haar)

En 2001 Paul Viola y Michael Jones [72] demostraron que mediante su algoritmo¹ es posible detectar una cara en 0.067 segundos en una imagen de 384 por 288 píxeles en una computadora con procesador Pentium III a 700 MHz. Desde entonces se han hecho una gran cantidad de algoritmos, pero según las pruebas que respaldan el presente estudio, este enfoque es adecuado en equipos de cómputo convencionales. Es frecuente que ciertos algoritmos sean implementados como parte de las bibliotecas del lenguaje de programación Python y otras herramientas tipo MATLAB. En este trabajo se usa la clase `CascadeClassifier` de la biblioteca OpenCV [57]. En [56] se hace una amplia explicación de la función, incluidos ejemplos en C++, Java y Python. Además, se puede ver un video muy ilustrativo en [73]. El algoritmo se encarga de hacer un barrido con una familia de filtros sobre una imagen. Cuando ciertas condiciones se cumplen, se cambia de filtro y luego de una serie de cambios, el algoritmo resuelve si se encontró o no una cara. Una segunda función recibe el resultado en forma de una lista de coordenadas de las esquinas superior izquierda y el alto y ancho de un rectángulo (más usualmente un cuadrado), delimitando cada una una cara. En el proceso se emplea una colección de imágenes llamadas positivas que sí tienen una cara, y otras llamadas negativas, donde no hay caras.

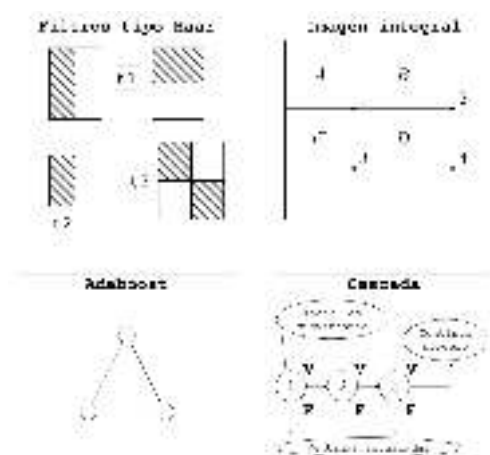


Figura 3.3: Componentes básicos del algoritmo de Cascadas de clasificadores débiles.

En la Figura 3.3 se presentan los componentes básicos de la teoría de Cascada de clasificadores débiles. El recuadro 'Filtros tipo Haar' se refiere a la aplicación de una familia de filtros deslizantes que se aplican a cada imagen para detectar diferentes tipos de características tales como bordes horizontales, verticales y diagonales así como esquinas. Con esto se forma una tabla de datos que servirá para las siguientes etapas. La 'Imagen integral' es un método abreviado y por ello muy rápido y eficiente de cálculo de áreas con base en un artículo de 1984 [15]. El recuadro de 'Adaboost' es un algoritmo que a partir de la tabla de datos forma una cadena (o 'bosque') de ramas binarias llamadas arbustos (o *stumps*) para alimentar lo que en el recuadro de esta misma figura

¹Cascada de clasificadores débiles aumentados.

se denomina 'Cascada'. Cada nodo en la cascada es un clasificador débil. La suma de clasificadores débiles forma un clasificador fuerte. Al lector le puede interesar el documento completo citado antes [72].

¿Qué problemas presenta este algoritmo? Primero, puede no detectar caras donde sí las hay. Segundo, a veces detecta caras donde no las hay. Tercero, para una misma cara puede hacer más de una detección.

Cabría preguntarse las razones de la falta de detección. Una de ellas se debe a las propiedades de la imagen. Si ésta es de baja resolución o muestra una 'gran' cantidad de caras, por ejemplo, unas 50 en las que cada cara ocupa menos de 20 píxeles aproximadamente, entonces el algoritmo falla. Sin embargo, si se aumenta el tamaño para que cada cara ocupe un cuadro de al menos 100 x 100 píxeles, el algoritmo encuentra la cara faltante. En las pruebas que se hicieron se cambiaron las dimensiones de las imágenes y también se modificó un par de parámetros de la función de detección. Más detalles se presentan en la Sección 4.4.1

Respecto a las detecciones incorrectas también es posible reducirlas modificando un parámetro (tamaño de ventana mínima) de la función de detección, pero no es garantía de que deje de reportar caras donde no las hay.

Siempre que se resuelve un problema surgen preguntas sobre las ventajas y desventajas de la solución. En este caso la ventaja se superpone a las desventajas. Considerando una alta tasa de aciertos y una gran velocidad de respuesta, en este caso se ha preferido la velocidad.

Y no es a cambio de nada. Como podrá verse más adelante, se han hecho mejoras que permiten reducir las dimensiones de un recuadro donde se ubica una cara; y siempre se encuentra una, si la hay. Se puede argumentar que dichas modificaciones también funcionarán con los otros algoritmos con los que se compara, pero se aclara que se trabaja con máquinas de recursos limitados, y la combinación de los factores *velocidad* y *precisión* indicará cuál algoritmo elegir.

3.3.2. Histograma de gradientes orientados (**dlib hog**)

HOG es una técnica completamente establecida en una patente estadounidense de 1986 [51]. El autor establece que su invento tiene varias aplicaciones: '...alineación de un objeto con respecto a una dirección de referencia, tal como el margen de una página impresa... el borde de una pieza en el torno o el eje principal de un cromosoma visto a través del microscopio'. El autor incluye análisis de series de tiempo y de voz. Su objetivo es establecer métodos de 'control de procesos en general'. Hace un planteamiento con base a la teoría de la información de Shanon [63] y establece su equivalencia entre la transmisión de información y la entropía en un sistema. La información que se encuentra en una imagen debe ser descrita en términos probabilísticos de sus características que permitan que un mensaje sea formado, transmitido y recibido. El mensaje que el autor codifica está en forma de símbolos. Considera el número de ocurrencias de cada símbolo y la probabilidad de que ocurra el símbolo en curso. El número de ocurrencias se controla en acumuladores en los que reconocemos el propio histograma: 'cada celda en

un histograma puede ser considerado equivalente a un símbolo de mensaje, y el número de tantos en la celda del histograma es, por ende, equivalente al número de ocurrencias del símbolo en el mensaje’.

Aquí termina la referencia al invento de Robert K. McConnell y se procede a tratar un segundo hito en este tema. Este ocurrió en 2005, cuando Dalal y Triggs [17] propusieron *Histograms of Oriented Gradients for Human Detection*. Aquí los autores toman una imagen y la pasan por un proceso de varias etapas, de las cuales se pueden destacar:

- 1) La normalización de los valores de la imagen;
- 2) El cálculo de los gradientes y de la orientación de los mismos;
- 3) El voto para ubicar la celda en algún acumulador del histograma; y
- 4) Su clasificación mediante SVM para determinar si hay una figura humana en la imagen o no.

Aunque el primer trabajo utilizó principalmente figuras geométricas, y el segundo se enfocó en detección de personas en imágenes, los resultados fueron llevados al campo de la detección de caras. En el caso de las figuras humanas, es claro que la proporción entre alto y ancho es diferente al de una cara; más o menos 8:1. No obstante, se puede usar el mismo enfoque de cuadricular la imagen en zonas de las dimensiones que se deseen. Por ejemplo, pasar la imagen de la cara a un rectángulo fijo de 64 píxeles de ancho por 72 píxeles de alto, y luego calcular el histograma de cada celda de 8 x 8. Además de esto, se pueden aplicar todas las operaciones adicionales antes y después del histograma, incluyendo zonas traslapadas, piramidación, cambios en las dimensiones de las celdas, y cualquier otra operación que el investigador estime necesaria para mejorar el estado del arte.

Precisamente Qiang Chu et al. [78] hicieron varias modificaciones al trabajo de Dalal y Higgs con las que se obtuvieron resultados superiores. Estos autores mencionan un intento de Viola y Jones (autores de Haar) para detectar humanos en movimiento. El trabajo de [78] no se aplica a personas en movimiento, pero con su variante de cascada de clasificadores débiles logra los resultados mencionados en su reporte.

En la Figura 3.4 se observa una representación de los vectores resultantes de aplicar el algoritmo HOG a una imagen. Se muestran pares de imágenes compuestas de un original del lado izquierdo y su representación HOG a la derecha. En la subfigura (a), se observa la imagen original. En (b), el recuadro de una zona de la cara y en (c), un recuadro de la zona de la oreja. La representación de esta última muestra una especie de destellos sobre un fondo negro. Dichos destellos son los vectores calculados en magnitud y dirección. Puede verse que algunos muestran una dirección preponderante mientras que los de la parte baja de la oreja muestran varias direcciones. Esta es una señal de que en esa zona hay más cambios de intensidad.

de la red consiste en saltarse una capa para conectarse a una tercera y ‘sumarse’ a la segunda. Para describir mejor el procedimiento, la salida de la primera capa se alimenta a la segunda y tercera capas. La salida de la segunda capa también se alimenta a la tercera. Otra característica de esta red es que usa SSD [49], es decir, un sólo recuadro propuesto, lo que difiere de otras redes que procesan múltiples recuadros propuestos sobre la misma imagen. La Figura 3.5 muestra la arquitectura base. El modelo fue procesado en ‘CAFFE: *Convolutional Architecture for Fast Feature Embedding*’ [33] que es un marco de procesamiento con el que se tiene ‘una limpia separación de la definición de la red (usualmente la parte novedosa para el investigador) de su implementación real’, de la misma forma en que lo hacen Tensorflow, Theanos o Torch.

3.3.4. Red neuronal convolucional de dlib (dlib_cnn)

Dlib [38] es una ‘biblioteca multiplataforma de propósito general de código abierto escrita en C++’ y portada a Python que empezó David King en 2002. Con sus componentes se puede trabajar con estructuras de datos complejas, álgebra lineal, procesamiento de imágenes y, a últimas fechas, con herramientas de aprendizaje mecánico de base estadística. La referencia es a su publicación de 2009, describiendo sus herramientas para dicha disciplina.

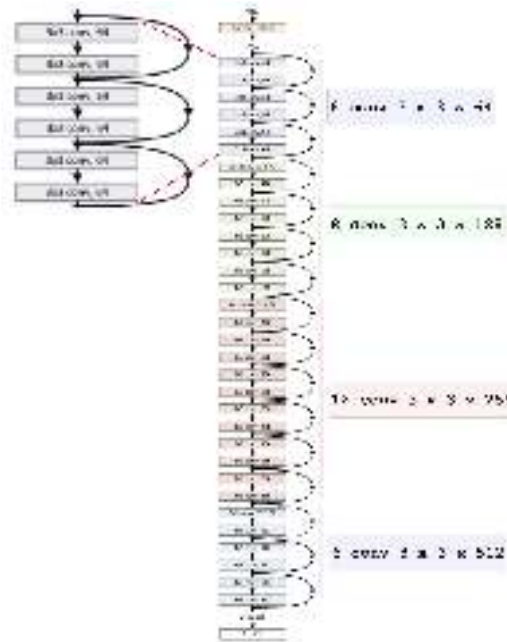


Figura 3.6: Arquitectura de la red tipo residual en la que se basa dlib para su algoritmo de detección.

El módulo dnn de dlib ocupa un archivo con los pesos de una red neuronal convolucional (cnn_face_detection_model_v1). El nombre del módulo es un acrónimo de *Deep Neural Network*. La red, como se dijo, es una tipo ResNet [28] que el autor pone disponible al público y establece que: ‘este modelo tiene un 99.38% de precisión cuando se ejecuta sobre la colección de LFW [30], y que es comparable con otros métodos del

estado del arte en febrero de 2017' [41]. La Figura 3.6 es una edición gráfica de la red ResNet-34 [28] para efectos de nitidez.

El autor utiliza una función de pérdida denominada *Max-Margin Object Detection* (MMOD) [39] en una red neuronal definida en [40]. Como lo indica en su publicación, MMOD se basa en HOG pero, a diferencia del HOG usado originalmente sobre 68 marcadores faciales, ahora usa sólo 5 para alinear la cara y, citándolo: '... excepto que aquí reemplazamos las características HOG con una CNN y procesamos el detector completo de principio a fin. Esto nos permite hacer detectores mucho más poderosos' y 'la contribución principal de este enfoque es la introducción de un nuevo método para detectar objetos en imágenes. No hace ningún submuestreo sino que optimiza todas las subventanas... puede usarse para mejorar cualquier método de detección de objetos que sea lineal... tal como HOG... Se muestra que un filtro HOG rígido puede superar los modelos deformables (de esa época) cuando el filtro HOG se produce por medio de MMOD'.

El autor de MMOD utiliza una formulación parecida a la de SVM y, por tanto, es tratada mediante multiplicadores de Lagrange. En consecuencia, encuentra una solución convexa, lo cual es mejor que conformarse con mínimos locales para no continuar buscando un posible máximo global, como ocurre en otros problemas de convergencia. Se recomienda la lectura completa del artículo citado [39].

Se dijo durante la Sección 3.3.3 que los algoritmos de detección proponen áreas que probablemente incluyen objetos y cada uno decide cómo proponer las áreas, cómo procesarlas, cuáles son sus dimensiones, cómo escalarlas, si es el caso, y cómo decidir si en esa área propuesta está o no el objeto. En el caso del presente estudio el objeto es una cara. En la Tabla 4.9 pueden compararse los resultados de los cuatro algoritmos después de implementarlos.

3.4. Funciones de pérdida

Como ya se ha dicho y es bien conocido, los métodos holísticos de detección de patrones no han sido lo suficientemente buenos ni en precisión ni en rapidez para detectar caras en imágenes, principalmente debido a cuestiones tan cotidianas como la iluminación, el cambio en la postura corporal y de la cara, así como por el uso de distractores tales como anteojos, gorras o la oclusión parcial de una cara por otra cara u objeto.

El uso de redes convolucionales en el reconocimiento de cantidades monetarias en cheques bancarios en 1998 [46] marcó la pauta para usar este tipo de herramienta. En general y sin entrar en aspectos históricos, es posible considerar este invento como uno de los intentos de simular el comportamiento cerebral mediante uno o varios programas de cómputo.

En general las redes neuronales se componen de una capa de entrada, de varias capas que procesan la entrada, y de una capa final que produce resultados. Los resultados se comparan con una salida esperada y si no coinciden con suficiencia, los resultados se reprocesan en sentido contrario. La diferencia entre la salida y el resultado esperado

es lo que se conoce como costo o pérdida. La función que se usa para tal comparación es la que se conoce como función de pérdida.

Hay varias formas de obtener un resultado para un mismo conjunto de datos. Puede alimentarse un vector en forma de histogramas, pueden ser un vector de características faciales, puede ser una representación probabilística o cualquier grupo de valores que el investigador suponga representativos del objeto de estudio.

En todo caso, la representación obtenida debe compararse con un patrón de referencia. Debido a que las representaciones vectoriales pueden establecerse en el hiperespacio, las distancias entre ellos pueden ser de tipo euclidiano, geodésico o estadístico. De esta forma se obtiene un grupo de distancias entre las que está la simple distancia euclidiana, la distancia coseno, la distancia tangente o la distancia de Mahalanobis. Una representación también puede tener la forma de densidad probabilística; y para su aprovechamiento se hace uso de un método conocido como entropía cruzada, cuya característica que lo hace similar a la distancia es que su valor siempre es mayor que cero [24] p.148. En general, en las funciones de pérdida se trabaja con un término conocido como ‘margen’ para efectos de que el valor no se haga extremadamente pequeño y la convergencia se estanque. En las siguientes secciones se cubrirán los tres tipos de función de pérdida planteados en la Tabla 1.1. Para ello se ocupará una red neuronal convolucional que se encontró adecuada y que se detalla en el Apartado 4.5.

3.4.1. Distancia euclidiana

La referencia más directa y simple proviene de la distancia entre dos puntos en el plano. Los puntos que se comparan en el caso del presente estudio no son simples coordenadas en el plano, sino vectores de n dimensiones, donde n puede ser 128, 500, 1,000 o cualquier número que el investigador decida usar para representar el objeto de estudio. En este caso, los investigadores analizaron 128 dimensiones y más, pero señalaron que 128 eran suficientes debido a que no se perdía precisión de manera significativa y se ganaba en tiempo de proceso [2]. Además, se obtenía una representación compacta porque cada elemento del vector queda representado por un entero. Es decir, al final se obtiene una representación facial en 128 bytes.

La función de pérdida queda representada por

$$l = \frac{1}{2N} \sum_i^N \|x_i^1 - x_i^2\|_2^2 \quad (3.1)$$

En la ecuación se identifica x^1 como la representación vectorial del objeto de referencia y x^2 como la representación vectorial del objeto procesado. Por ejemplo, si se trabaja con 10 clases, y tanto la clase de referencia como la clase procesada corresponden a la clase con índice 4, los vectores de ambas clases serán $[0,0,0,0,1,0,0,0,0,0]$. Es claro que el único valor no cero es el que corresponde al índice 4 (partiendo de 0). El vector de salida se obtiene mediante una capa densa a la que típicamente se le aplica la función softmax, la cual se puede definir como la probabilidad calculada de que la j -ésima clase, dado un vector de muestra \mathbf{x} y un vector calculado \mathbf{w} sea:

$$P(y = j|\mathbf{x}) = \frac{e^{x^\top \mathbf{w}_j}}{\sum_i^K e^{x^\top \mathbf{w}_i}} \quad (3.2)$$

Entonces, la aplicación de distancia euclidiana consiste simplemente en resolver la ecuación (3.1), sustituyendo el vector \mathbf{x} de referencia y el vector \mathbf{w} calculado. La biblioteca Keras [10] reduce la ecuación (3.1) a la siguiente línea:

```
de = K.sqrt(K.sum(K.square(y_pred - y_true), axis=-1))
```

Para mostrar una de las aplicaciones de distancia euclidiana se hace referencia a una de las herramientas que la usan, como `dlib`, que en la descripción de su programa de reconocimiento facial establece: ‘Este ejemplo muestra cómo usar la herramienta de reconocimiento facial de `dlib`. Esta mapea la imagen de una cara humana en un vector en el espacio de 128 dimensiones, donde las imágenes de la misma persona (*sic*) están cerca una a otra y las imágenes de diferentes personas están separadas. Por lo tanto, puedes realizar reconocimiento facial mapeando caras en el espacio de 128 dimensiones y luego verificar si su distancia euclidiana es suficientemente pequeña’. Una de las formas de verificar la cercanía es k -NN, de la cual se hablará en el Apartado 4.6.3

3.4.2. Entropía cruzada

Como se mencionó al hablar de HOG, hay una relación probabilística entre la información y la entropía. El ‘valor informativo’ de un mensaje comunicado depende del grado en que el contenido del mensaje informa. Si un evento es muy probable, casi no hay información nueva. Si no es probable pero se sabe que ocurrirá, el valor informativo es alto porque comunica el resultado de un evento de muy poca probabilidad [13], [63].

Así como en pérdida euclidiana se generan vectores que se comparan mediante la distancia euclidiana en el hiperespacio, en este caso se generan densidades probabilísticas que se comparan contra una referencia. Una ecuación que relaciona tales componentes es la Divergencia Kullback-Leibler (KL en adelante) [24], p.74, que tiene la forma:

$$D_{KL}(P||Q) = \mathbb{E}_{x \sim P} \left[\log \frac{P(x)}{Q(x)} \right] = \mathbb{E}_{x \sim P} [\log P(x) - \log Q(x)] \quad (3.3)$$

Una propiedad de esta ecuación es que siempre es positiva, y otra es que para que sea cero, se requiere que si y sólo si P y Q sean la misma distribución, lo cual es ventajoso, puesto que se hablaría entonces de la misma clase. Debido a que la Divergencia KL es no negativa y mide la diferencia entre dos distribuciones, se usa como sinónimo de medir algún tipo de distancia entre esas distribuciones [24], p.74.

Una imagen de entrada tiene cierta información autocontenida que se puede calcular mediante su entropía. Esta entropía se encuentra plenamente demostrada en la teoría de la información [63], y tiene la forma:

$$H(x) = \mathbb{E}_{x \sim P} [I(x)] = -\mathbb{E}_{x \sim P} [\log P(x)] \quad (3.4)$$

En esta ecuación se puede identificar el término $\log P(x)$ que se encuentra en la ecuación (3.3). Si se omite este término, queda la fórmula de la entropía cruzada,

$$H(P, Q) = -\mathbb{E}_{x \sim P} \log Q(x), \quad (3.5)$$

por lo que es posible concluir que la entropía cruzada es la diferencia de la entropía menos la Divergencia KL. ‘Minimizar la entropía cruzada con respecto a Q es equivalente a minimizar la divergencia KL porque Q no participa en el término omitido’. [24], p.75.

La variante de entropía cruzada usada en este trabajo es la ‘Entropía cruzada categórica dispersa’, la cual se recomienda cuando se trabaja con dos o más clases (de allí lo categórico). Lo disperso se refiere a que los valores de referencia² se codifican en un vector donde todos los elementos son cero excepto el que corresponde a la clase en cuestión. Ya antes se habló de codificar³ una clase como $[0,0,0,0,1,0,0,0,0]$ para indicar que es la que tiene el índice 4, empezando desde 0.

3.4.3. Pérdida por tripleta

La pérdida por tripleta es una técnica que ‘genera un mapeo de imágenes de caras en un espacio euclidiano compacto donde las distancias corresponden directamente a una medida de similaridad facial’ [61].

Las imágenes de caras pueden ser procesadas por una red neuronal convolucional para generar representaciones vectoriales en el espacio de 128 dimensiones. Como ocurre con cualquier red, en la capa de salida se produce un vector que se compara contra el resultado deseado. En este caso se usan tripletas de imágenes formadas por:

1) Un elemento llamado ancla, **2)** Un elemento llamado positivo que es de la misma identidad que el ancla, y **3)** Un elemento llamado negativo que es de una identidad diferente al ancla.

Se quiere lograr que las representaciones de la misma persona estén más cerca entre sí, y que las de personas diferentes formen su propio agrupamiento, pero que estén separadas de los otros grupos.

Es posible formalizar esta condición mediante:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \quad (3.6)$$

En la ecuación anterior se observan los argumentos a , p y n que son representantes de las tripletas enumeradas antes. El primer término, que representa la distancia entre identidades iguales más un margen (α), trata de ser mayor que el segundo término,

²La mayoría de referencias sobre aprendizaje mecánico llaman a esto ‘Ground truth’, término tomado de la Geografía.

³La literatura en redes neuronales, llama a esto ‘One-hot encoding’, término tomado del campo de Sistemas digitales.

que representa la distancia entre identidades diferentes. Se usa la siguiente ecuación para llevar las tripletas a cumplir aquella condición:

$$\mathcal{L}(a, p, n) = [\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha - \|f(x_i^a) - f(x_i^n)\|_2^2] \quad (3.7)$$

Por ejemplo, en esta formulación:

$$d(ap) - d(an) \quad (3.8)$$

donde $d(ap)$ tiende a cero por tratarse de identidades iguales, y $d(an)$ tiende a un número mayor que cero porque son identidades diferentes, se aprecia que la resta entre algo cercano a cero y algo mayor que cero siempre será un valor negativo, y en ese caso a la función de convergencia no le importa cuánto se aleje de cero en el sentido negativo. La intención es que se aproxime a cero pero desde el lado positivo:

$$[\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha - \|f(x_i^a) - f(x_i^n)\|_2^2].$$

Entonces se establece un margen (α en las ecuaciones 3.6 y 3.7). Para obtener siempre un valor positivo hay que sumar dicho margen a la diferencia de la misma identidad:

$$d(ap) + m - d(an) \quad (3.9)$$

Para hacer más efectiva la separación, se incorpora el concepto de tripletas difíciles. Esto quiere decir que se seleccionan imágenes en las que resulta difícil decidir si pertenecen a la misma identidad. También es posible incorporar pares de imágenes ancla-negativo que tengan un evidente parecido. Un ejemplo de esto es el caso de imágenes de gemelas o la Figura 3.7.



(a) Perfil

(b) Frente

Figura 3.7: Imágenes de dos personas muy parecidas [67].

Ahora se citan algunos fragmentos de Andrew Ng [55]: ‘El efecto de elegir esas tripletas es aumentar la eficiencia computacional del algoritmo. Si eliges las tripletas aleatoriamente, entonces muchas tripletas serán muy fáciles y el descenso de gradiente no haría nada porque la red sólo las dejaría pasar. Sólo eligiendo tripletas duras es que el descenso de gradiente tiene trabajo que hacer para separar estas cantidades ($d(ap), d(an)$) entre sí’. ‘Cuando a la red le presentas tripletas duras en las que $d(ap) \sim d(an)$, la obligas a efectuar más ciclos en los que trata de alejar los términos hasta alcanzar el valor de umbral y salir del evento.’

En la Figura 3.8 se observa la interpretación gráfica que los autores de *FaceNet* [61] dan a su algoritmo: ‘La pérdida por tripleta minimiza la distancia entre un *ancla* y un *positivo*, los cuales tienen la misma identidad y maximiza la distancia entre el *ancla* y un *negativo* de diferente identidad’.

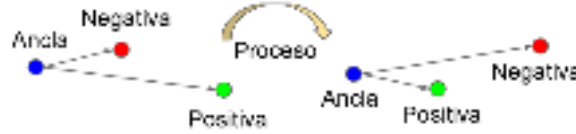


Figura 3.8: Efecto de procesar tripletas.

3.5. Clasificadores

Conviene traer a la memoria una definición comúnmente aceptada del tema de reconocimiento de patrones a efectos de prevenir cualquier confusión. Esto se refiere a los trabajos de [44], p.1, que define el reconocimiento de patrones como: ‘la discriminación que trata de estimar o predecir la naturaleza desconocida de una observación, una cantidad discreta tal como negro o blanco, uno o cero, enfermo o sano, real o falso. Una observación es una colección de medidas numéricas tales como una imagen (la cual es una secuencia de bytes, uno por píxel), un vector de datos climáticos, un electrocardiograma o la firma en un cheque digitalizado’. Más formalmente, una observación es un vector x de d dimensiones. La naturaleza desconocida de la observación se denomina **clase**⁴. Se denota por y y toma valores de un conjunto finito $\{1, 2, \dots, M\}$. En reconocimiento de patrones, uno crea una función $g(x) : \mathcal{R}^d \rightarrow \{1, \dots, M\}$, que representa la propia estimación de y dado x . El mapeo g es llamado clasificador. El clasificador falla en x si $g(x) \neq y$. También se hace referencia a [45], p.2, que lo define como que: ‘se enfoca en el problema de clasificar objetos, que frecuentemente son representados como vectores o como cadenas de símbolos, en categorías. La dificultad es sintetizar y luego calcular eficientemente la función de clasificación que mapee objetos en categorías, dado que los objetos en una categoría pueden tener representaciones de entrada ampliamente variantes. En la mayoría de los casos, la tarea es conocida por el diseñador a través de un conjunto de patrones de ejemplo cuya categoría conoce, y en general, a través de un conocimiento a priori de la tarea, tal como: “la categoría de un objeto no cambia cuando éste es ligeramente trasladado o rotado en el espacio”.’, y a [70], p.16, que establece que el reconocimiento de patrones es: ‘una disciplina científica cuyo objetivo es la clasificación de objetos en un número de categorías o clases’.

Se ha realizado este trabajo en el supuesto de que esta es una acepción comúnmente usada en el medio académico y científico, de suerte que en el apartado 4.6, se da por sentado que el lector entenderá ‘reconocimiento’ como un sinónimo de ‘clasificación’.

En redes neuronales convolucionales para clasificación de imágenes en modo supervisado es necesario asignar una clase al objeto a identificar. Aquí se usa un método casi trivial para informar a la red las diferentes clases con las que trabajará. Se trata de una simple estructura de directorio donde cada carpeta contiene las imágenes de

⁴Negrillas puestas por el autor de esta tesis.

una identidad, y hay tantas carpetas como identidades⁵. También se ha incorporado un método de adquisición de imágenes a través de un flujo de video de cuyos detalles se hablará en el Capítulo 4.

Ha sido necesario derivar algunas funciones para leer colecciones de imágenes propias y dejarlas en un formato compatible con las redes neuronales usadas aquí. Se menciona que, poco antes de concluir este trabajo, se encontró que TensorFlow [1] desarrolló una clase llamada *Dataset*, la cual ya incluye todo lo necesario para pasar como parámetro una ruta a la colección, y dicha clase se encarga de leer los datos y dejarlos preparados para su proceso en la red que se elija.

Este tipo de ayuda facilita el análisis de datos pero oculta los detalles de los mismos. Aquellos se encapsulan en estructuras difíciles de desenmarañar y el desarrollador se tiene que conformar con los medios de acceso disponibles o continuar con sus propias primitivas.

Los clasificadores que aquí se estudiarán son de uso generalizado y con buenos resultados. También están soportados por un sólido fundamento matemático y tienen tiempo siendo empleados. En la Tabla 3.2 son enumerados y se establece el orden en que serán abordados.

Tabla 3.2: Clasificadores analizados en esta sección.

Método	Técnica	Inventor	Año
PCA	Reducción de dimensiones por análisis de covarianza	Karl Pearson	1901
k -NN	Clasificación por vecindad	E. Fix y J. Hart	1991
SVM	Clasificación por hiperplano óptimo	Vladimir Vapnik	1982

3.5.1. PCA

PCA es el acrónimo de *Principal Component Analysis* o análisis de componentes principales. Es un método estadístico cuyas propiedades se han encontrado útiles para representar y reconocer patrones. Las propiedades más sobresalientes son que en cualquier conjunto de datos se pueden encontrar al menos dos ejes principales que representan a todos ellos mediante algo parecido al ajuste de una recta de mínimos cuadrados. Los ejes son perpendiculares entre sí, y el de mayor tamaño representa la dirección en la que hay más varianza en los datos y, por tanto, es más representativo de ellos. El segundo en dimensión representa el segundo nivel de representación de datos.

Los ejes principales mencionados son otra forma de hablar de dos términos más reconocidos y asociados a PCA, denominados eigenvalores y eigenvectores. Para un conjunto de datos representado en forma matricial, se desea encontrar un conjunto de coeficientes que multipliquen un vector de forma tal que sean igual a la matriz original multiplicada por otro vector:

⁵Luego de adquirir las representaciones, las imágenes se pueden borrar puesto que ahora se tiene un repositorio binario en disco. En la sección 4.6.1 se muestran detalles de esta afirmación

$$\mathbf{Ax} = \lambda \mathbf{x}$$

Cuando se resuelve el sistema de ecuaciones asociado, se obtienen los coeficientes (λ) o eigenvalores y el vector o eigenvector.

En términos de representación, los eigenvalores indican las coordenadas del punto de origen de otro sistema coordenado. Para dos dimensiones, se pueden calcular unas coordenadas a lo largo de los dos ejes de ese nuevo sistema utilizando la media, la desviación estándar y el eigenvector de cada eje. El eje principal o mayor queda orientado en la dirección de una línea de mínimos cuadrados que representa la ‘tendencia’ de los datos. El otro eje, perpendicular al primero, apunta en la dirección determinada por el segundo eigenvector. La Figura 3.9 muestra la gráfica de unos datos generados al azar en la que se puede apreciar cierto estiramiento del origen hacia el extremo superior derecho de los datos. El eje que apunta hacia arriba está determinado en su centro por la media de los datos, y el punto final del vector es calculado utilizando el primer eigenvector. El eje que apunta hacia abajo es similar en su origen, y el extremo está determinado por el eigenvector del segundo componente.

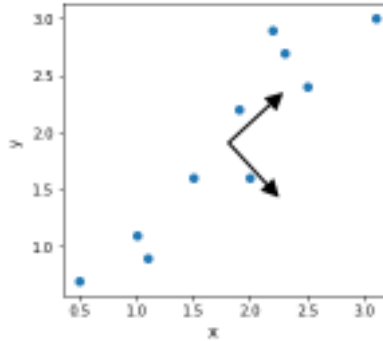


Figura 3.9: Ilustración de los eigenvectores de un conjunto de datos.

PCA se relaciona con el reconocimiento facial debido a una técnica mediante la cual se agrupan representaciones faciales de bajas dimensiones. Los grupos se promedian y arrojan un vector. En el mismo hiperespacio existen otros vectores que representan a otras caras. Cuando se quiere identificar alguna cara que no estaba en los datos originales, se hace una comparación de distancia del vector representativo de la nueva cara con todos los otros vectores en el hiperespacio. El criterio de distancia comúnmente usado es distancia euclidiana. La decisión de la clase normalmente se logra con la técnica k -NN.

3.5.2. k -NN

k -NN proviene de *k-Nearest Neighbors* o k -vecinos más cercanos. Es un método registrado en 1951 por Evelyn Fix y Joseph Hodges [21], pero no se publicó en su momento, sino hasta 1970 (año en que se desclasificó). Luego, Thomas Cover y P. E. Hart, investigadores de la Universidad de Stanford, publicaron el trabajo denominado *Nearest Neighbor Pattern Classification* [14].

Dentro de la familia de métodos estadísticos, k -NN cae en los conocidos como no paramétricos. Esto quiere decir que no depende del conocimiento previo de una distribución probabilística de los datos. Otros casos de métodos no paramétricos son los histogramas y SVM.

El método es intuitivo y no requiere una elaborada demostración a no ser que se quiera hablar de sus propiedades: una de ellas es que su error máximo nunca supera al doble del error de Bayes [24], p.131. Otra lectura de esto es que el vecino más cercano contiene la mitad de la información del conjunto de datos cuando éste es infinito. Otra propiedad es que para una cantidad infinita de datos, la clasificación de una muestra desconocida es 100% precisa para $k = 1$ [14].

Básicamente consiste en poner en memoria todos los datos que se tienen, fijar un valor de k , que típicamente es bajo (hasta 7) y debe ser non para evitar el empate. Luego se introduce un dato y se buscan los k elementos más cercanos. Es necesaria una medida de distancia para poder hacer la comparación. Típicamente se usa distancia euclidiana, que como se sabe, es útil para calcular distancias en un hiperespacio. En este trabajo las muestras analizadas son representaciones vectoriales de caras de n dimensiones, donde n va de 22 a 10,000. El empate se puede producir debido a que cada valor guarda una distancia respecto al valor de prueba y con $k = 4$; si dos de una clase están cerca pero dos de otra clase también, habrá un empate. De allí la conveniencia de que k sea impar. La clase resultante es aquella que tiene más elementos cercanos a la que se busca. La Tabla 3.3 muestra los posibles resultados si $k = 3$ para un hipotético nuevo elemento de tres clases disponibles.

Tabla 3.3: Vecinos más cercanos para tres clases.

Clase A	Clase B	Clase C	Resultado
1	1	1	Gana el azar
1	2	0	Gana clase B
2	1	0	Gana clase A
3	0	0	Gana clase A

En la Tabla 3.3 los números indican cuántos elementos de cada clase están más cerca de la clase buscada. Se puede observar una especie de registro de votos, y por ello hay autores que denominan a este método como uno de ‘votación’. En el primer caso, tres clases tienen la misma distancia a la clase de prueba, y por ello ‘gana’ una clase elegida aleatoriamente. En la Figura 3.10 se muestra el ejemplo de un programa que genera, de manera aleatoria, 13 muestras de una clase y 12 de otra, a partir de 25 elementos pedidos en un rango de valores entre 1 y 100. Luego se pide otro valor aleatorio en ese rango y se busca su clase. En la Figura 3.10 se pueden ver dos clases: cuadrados y círculos. Un nuevo elemento en forma de triángulo en color negro debe ser clasificado. Es clara la razón por la cual el algoritmo indica que pertenece a la clase de cuadrados.

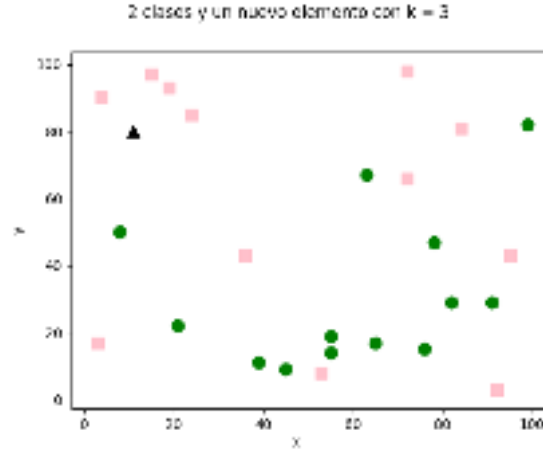


Figura 3.10: Ejemplo de clasificación con el algoritmo k -NN, con $k = 3$.

Formalizando el método, así se plantea:

Sea x_i una muestra de entrada con p características $(x_{i1}, x_{i2}, \dots, x_{ip})$,

n el total de muestras de entrada $(i = 1, 2, \dots, n)$, y

p el número total de características $(j = 1, 2, \dots, p)$.

La distancia euclidiana entre las muestras x_i y $x_l (l = 1, 2, \dots, n)$ queda definida como:

$$d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \dots + (x_{ip} - x_{lp})^2} \quad (3.10)$$

La Figura 3.10 muestra un ejemplo de k -NN con $k = 3$.

3.5.3. SVM

SVM es el acrónimo de *Support Vector Machines*, o máquinas de soporte vectorial. El nombre original no tenía nada que ver con un mecanismo, pero debido a que mucha gente cambia de nombre a las cosas, con el paso del tiempo alguno de ellos queda fijo en la literatura. Véase, por ejemplo, [27], donde se les llama patrones de soporte.

SVM es un método inventado en 1964 por Vladimir Vapnik, quien lo presentó en su tesis doctoral. En ésta sentó el precedente de una herramienta que empezó a utilizarse en 1992 para el reconocimiento de números escritos a mano como los que vienen en los cheques y en las cartas postales. Resulta interesante ver la diferencia en fechas, porque pasaron 28 años antes que su teoría saliera de la oscuridad académica rusa y llegara a una entidad privada (*Bell Labs*), en la que sus investigadores ha ganado nueve Premios Nobel y cuatro Premios Turing, entre otros.

En [3], p.150 se presenta una comparación de la tecnología de Máquinas de soporte vectorial contra una red neuronal de cinco capas [46], obteniendo un resultado de un punto porcentual por encima de otros algoritmos analizados.

Debido a que se considera aplicable a casi cualquier tipo de problema de separación lineal de clases, se dice que es un mecanismo universal. Se puede trabajar con clases múltiples que, en su estado original, no son linealmente separables, pero que mediante el aumento de una dimensión al espacio de trabajo se hacen linealmente separables [3].

En la Figura 3.11 se ilustran puntos representando dos clases: círculos verdes y cuadrados rojos. También se observan tres líneas rectas que, cada una, separa linealmente ambas clases. La pregunta es: ¿existe una recta que separe óptimamente ambas clases? La respuesta es el ‘hiperplano óptimo’. Además, cabe preguntarse: ¿cómo encontrar dicha línea? El autor demostró que no sólo se puede encontrar una línea de separación, sino que es posible determinar una franja de margen máximo entre clases, como se ve en la Figura 3.12. La búsqueda de esta franja es una de las partes más importantes del algoritmo.

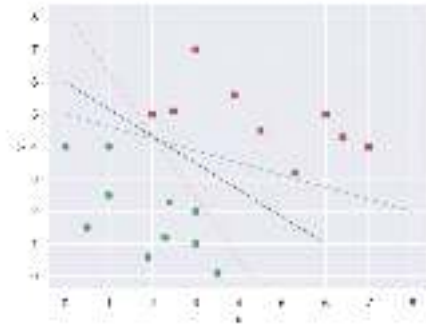


Figura 3.11: Líneas separando datos de dos clases.

En las siguientes expresiones se muestra la respuesta mediante pasos abreviados. La deducción completa se puede encontrar en una variedad de fuentes, pero se recomienda la del profesor Patrick H. Winston, del MIT⁶ [76].

Es posible enunciar un conjunto de datos:

$$(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m), \quad x \in \mathbb{R}^n; \quad y_i \in \{+1, -1\} \quad (3.11)$$

donde \vec{x}_i es una muestra en forma de vector e y_i es una clase. El conjunto es representado en la Figura 3.12, la cual, por simplicidad, muestra sólo cuatro puntos.

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

$$(\vec{w} \cdot \vec{x}) + b = 0 \quad (3.12)$$

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

⁶Patrick Winston, un amado profesor y científico del MIT, falleció el 19 de julio de 2019, a los 76 años de edad, en el Hospital General de Massachusetts, en Boston.

$$(1 - b - 1 + b)/\|\vec{w}\| = 2/\|\vec{w}\|, \quad (3.13)$$

punto en el que es posible introducir los multiplicadores de Lagrange que ayudarán a optimizar la siguiente expresión:

$$\mathcal{L} = 1/2\|\vec{w}\|^2 - \sum \alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1] \quad (3.14)$$

Luego de resolver las restricciones de Lagrange, se llega a la siguiente solución:

$$\sum \alpha_i y_i \vec{x}_i \cdot \vec{w} + b \geq 0 \quad (3.15)$$

Aquí se puede concluir que la maximización de la distancia entre los vectores de soporte de las dos clases depende principalmente del producto punto de los vectores dados.

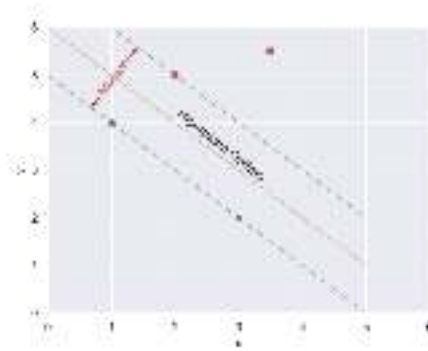


Figura 3.12: Hiperplano óptimo mediante el mayor margen entre clases.

La Figura 3.12 ilustra el concepto gráficamente. Las líneas sólidas a ambos lados del hiperplano, y paralelas a éste, son tocadas por puntos conocidos como vectores de soporte. Los puntos mencionados forman parte del conjunto de datos bajo análisis. En esta figura se observa una línea perpendicular al hiperplano. Ésta representa el margen óptimo encontrado.

Capítulo 4

Implementación

4.1. Introducción

Para sustentar la hipótesis de este trabajo se usaron los algoritmos antes mencionados, implementados en programas principalmente en Python, algunos en bash y otros en R. Para cada una de las secciones que se han planteado en el marco teórico el autor codificó al menos un programa que muestra cómo se analiza cada uno de los conceptos citados. En la Tabla 5.5 se muestra el número aproximado de programas hechos durante este trabajo.

Se partió de una serie de imágenes grupales e individuales que se usaron en diferentes pruebas. Una de ellas fue la de detección facial de manera grupal y otra de manera individual. Para el cálculo de las funciones de pérdida se usaron caras individuales que se recortaron de todo el conjunto disponible. El recorte se hizo primero por detección automática y después se depuró manualmente. Para el reconocimiento se usaron imágenes individuales de personas conocidas por el autor a partir de la captura de un flujo de video.

En adelante se procederá con el mismo enfoque de los capítulos anteriores, en los que se ha ido abordando cada sección siguiendo los objetivos específicos de este trabajo. De esta forma se trabajará primero en la parte de detección, luego en la de funciones de pérdida, y finalmente en la de clasificación.

4.2. Equipo de cómputo utilizado en este trabajo

En la Figura 4.1 se ven los cuatro equipos empleados. El equipo principal es la computadora Vaio con procesador Intel Centrino 2 con 4 GB de memoria RAM y sistema operativo Ubuntu 16.04. Como se menciona adelante, este equipo no fue capaz de procesar una red con imagen de entrada de 180 x 180 píxeles y por ello se empleó la Mac Mini con 8 GB de memoria RAM y procesador Intel Core i5. Otra máquina empleada durante el proceso en lote de imágenes es una Mac Book Pro con 4 GB de memoria RAM y procesador Intel Core 2 Duo con sistema operativo Ubuntu 20.04.

Finalmente y como prueba de concepto, se usó una computadora de placa reducida Raspberri Pi modelo B+. De esta última podemos decir que hay un descuido en el mantenimiento de paquetes ya que no se pueden reproducir las condiciones originales de una instalación (2019), una vez que se formatea la unidad de almacenamiento.

<ul style="list-style-type: none"> ▪ Vaio Centrino con 4GB RAM @Ubuntu 16.04 ▪ Procesó la mayor parte de este trabajo 	<ul style="list-style-type: none"> ▪ Mac Mini Core i5 8GB RAM ▪ Ejecución de red cuando la Vaio se atoró 
<ul style="list-style-type: none"> ▪ Mac Book Pro Core 2 duo 4GB RAM @Ubuntu 20.04 ▪ Ejecución de pruebas en lote 	<ul style="list-style-type: none"> ▪ Raspberri Pi modelo B+ ▪ Probar la ejecución del reconocedor 

Figura 4.1: Las cuatro computadoras empleadas en este trabajo.

4.3. Elaboración de las colecciones de imágenes

En este apartado se trabaja en el cumplimiento del Objetivo **iv**).

Para efectos de automatizar la obtención de ciertos valores estadísticos que se grafican adelante, se hizo una serie de programas que separan las imágenes en diferentes carpetas. En algunos programas se realizó una selección manual, y en otros, automática. La selección manual fue de dos tipos. En un caso, un programa muestra una lista y despliega cada una de las imágenes. Se tienen botones que representan el criterio de separación, se da clic a uno de ellos y la imagen se acomoda en la clase elegida. Un programa de este tipo se muestra en la Figura 4.2. Los tres criterios de separación son: cara neutra, ocluida y gesticular.

En otro caso, se usó un programa de selección manual como el que se muestra en la Figura 4.3. Aquí se usa el ratón para seleccionar una región donde hay una cara posible de detectar sin importar mucho sus características visuales. El programa

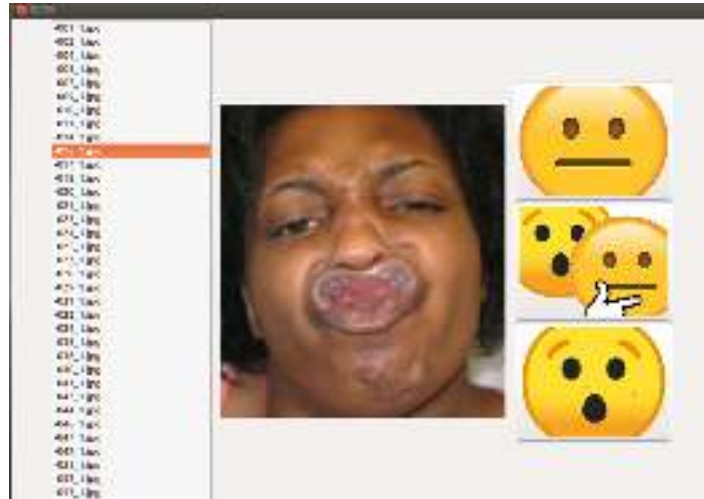


Figura 4.2: Aspecto de un programa para agrupar imágenes.

utilizado para seleccionar las caras y recortarlas, deja un rastro de los rectángulos formados al arrastrar el ratón.



Figura 4.3: Aspecto del programa para seleccionar caras en imágenes.

La selección automática se realizó ejecutando una serie de programas sobre imágenes grupales y generando carpetas donde se acomodaron las caras detectadas. En las imágenes había al menos 10 personas, pero el número de caras utilizables era inferior, principalmente debido a oclusiones severas o dimensiones reducidas. En la Figura 4.3 se muestra un ejemplo de foto grupal. También se muestra un rastro de recuadros en color azul claro dejados por el programa de recorte. Se puede ver una ampliación de una cara recortada y una muestra de cómo quedó en la colección, ya como una cara individual de 100 píxeles por lado. En la Figura 4.4 se muestra un mosaico de 10 de las imágenes recortadas en la foto grupal de la Figura 4.3



Figura 4.4: Diez de las caras recortadas de la foto grupal de la Figura 4.3.

En cualquier grupo de imágenes donde hay personas fotografiadas se pueden identificar atributos que permiten separar unas de otras. Un atributo básico se refiere a si la cámara está ‘viendo’ de frente a la persona o si la cámara ‘ve’ a la persona de perfil. También es posible decir si los ojos están alineados horizontalmente, parcialmente inclinados o muy inclinadas, y a esto se le puede llamar una pose secundaria. Las caras también pueden estar parcialmente ocluidas, en actitud neutral o haciendo un gesto. A esto, aquí se le llama condición. Finalmente se separa por resolución, la cual puede ser baja, media o alta.

Se pueden identificar algunas características generales de las imágenes. Cuando hay suficiente iluminación, como en el caso de fotos diurnas, los valores del fondo disparan los valores claros (cercaos a 255). Si la imagen fue tomada de noche o el fondo es oscuro, se disparan los valores oscuros (cercaos a 0)¹. En ambos casos, resulta difícil evaluar si la cara está de perfil derecho o perfil izquierdo. Otro factor que impide una decisión adecuada es la resolución. En la mayoría de los casos de baja resolución, la distribución de frecuencias es muy pareja y dispersa en ambos lados de la cara. Un último factor a mencionar es el del recorte. Durante este proceso cada cara se redimensionó para tener al menos 100 píxeles por lado. En la Figura 4.5 se aprecia en (a), las imágenes tal como se seleccionaron para recorte, y en (b), las imágenes redimensionadas para tener 100 píxeles por lado.



Figura 4.5: Ajuste de dimensiones debido al recorte.

4.3.1. Caras de frente, de perfil y variantes

Las 2,427 imágenes de caras de la colección fueron detectadas y recortadas a partir de imágenes grupales e individuales en pose casual en su mayoría. Sólo algunas fueron tomadas de retratos, lo que implica una clara intención de posar. Debido a que son 4 los mecanismos de detección estudiados, a continuación se presenta un ejercicio genérico de preparación y posteriormente se aborda cada mecanismo por separado.

Uno de los ejercicios produjo 1,780 imágenes con una cara. De este conjunto se creó una primera partición de las imágenes, separando la posición de la cara respecto a la cámara en un plano principalmente horizontal. Dicha posición corresponde al atributo básico conocido como frente o perfil. Se usó uno de los programas de detección automática en lote. El resultado es como se muestra en la gráfica de la Figura 4.6.

¹Tratándose a imágenes en escala de grises.

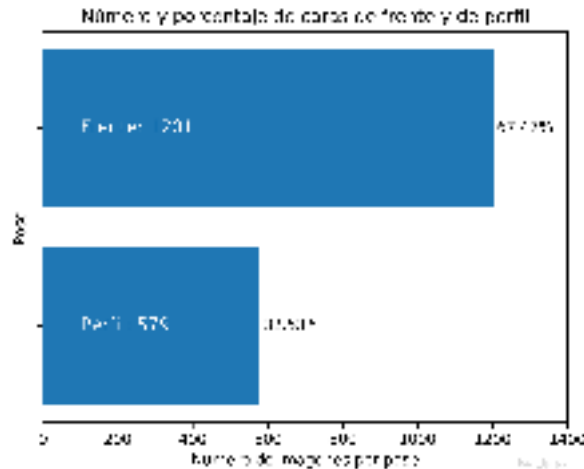


Figura 4.6: Número de caras de frente y de perfil.

Algunas caras fueron recortadas indebidamente a causa de la resolución. Esto creó imágenes de caras de perfil donde al frente de la cara había mucha luz o escasa iluminación. Esto impide calcular un histograma representativo, como se puede apreciar en las Figuras 4.7 y 4.8.

Se encontró que es necesario hacer diferenciaciones adicionales en cuanto a la variabilidad de la pose, ya que una cara de frente puede estar inclinada respecto al eje horizontal y el perfil tiene un amplio rango de ángulos a considerar respecto al eje vertical.

4.3.1.1. Caras de perfil

Se dividirá este apartado en las caras de perfil completo y perfil parcial. El perfil completo se considera como uno en que se ve uno de los ojos de manera completa y el otro ojo de manera parcial. El perfil parcial es uno en que ambos ojos se ven, pero la cara definitivamente no está de frente. Por supuesto, el criterio elegido es completamente arbitrario.

Perfil completo Se intentó la detección automática de caras de perfil con base en un criterio un poco ingenuo: se separaron manualmente las imágenes de perfil completo y parcial. Cada imagen fue partida por la mitad sobre la vertical y, para cada sección, se calculó un histograma como se muestra en las Figuras 4.7 y 4.8. Se esperaba encontrar una franca diferencia en valores pico en la mayoría de los casos pero no fue así. Se supuso que el lado sobre el que está la nuca es más oscuro que el lado donde está la nariz. Sin embargo, varios factores van contra este supuesto. En la Figura 4.7 se muestran caras que se estimaron como giradas a la derecha de la persona. Las imágenes (a) y (b) muestran una detección correcta del lado del perfil. Las imágenes (c) y (d) muestran detecciones incorrectas. En la Figura 4.8 se muestra lo que el programa de detección automática de orientación estimó como giradas a la izquierda de la persona. Las imágenes

(a) y (b) muestran una detección correcta. Las imágenes (c) y (d) muestran detecciones incorrectas.

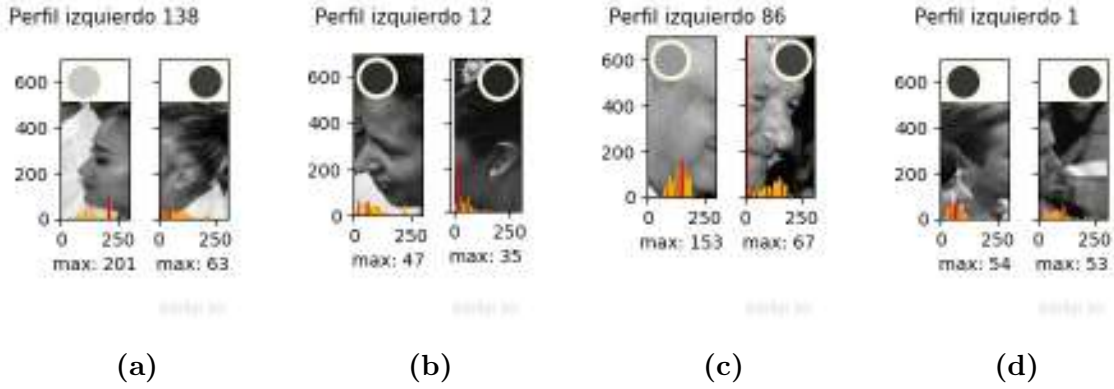


Figura 4.7: Detección automática de caras giradas a la derecha.

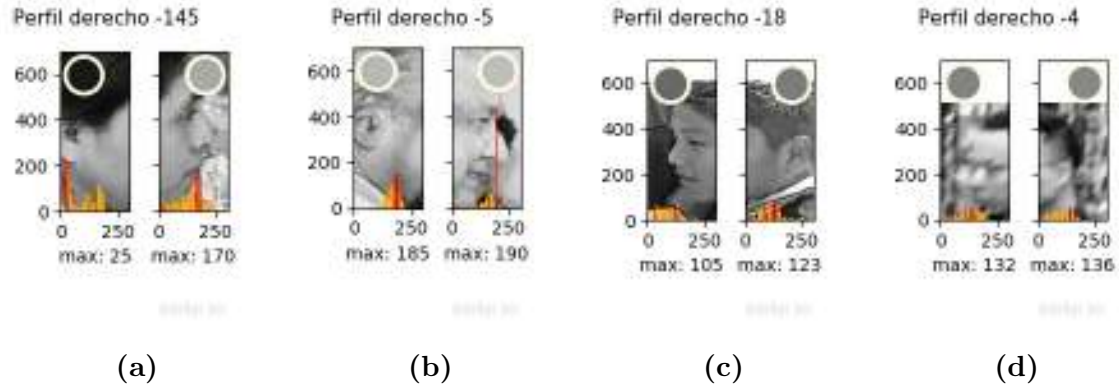


Figura 4.8: Detección automática de caras giradas a la izquierda.

Las imágenes de las Figuras 4.7 y 4.8 requieren una explicación. Al trabajar en la detección se exploraron las imágenes en niveles de gris. De este modo, el 0 representa el negro y el 255 el blanco. Cada histograma completo se muestra en color naranja encima de la cara. El histograma parcial en color rojo representa los 15 valores de pixel más altos por lado vertical. El número abajo de cada lado indica el valor en torno al cual se reunieron los valores más representativos de la imagen. En la imagen 4.8 (b) se puede ver un pico de más de 400 valores cercanos a 190 del lado derecho y una concentración de valores cerca de 185 del lado izquierdo. La diferencia entre valores máximos es mínima mientras que en la Figura 4.8 (a), la diferencia es mayor. De cualquier forma, debido a que se cumple el criterio de que el lado más claro tiene la parte de la cara, y la parte oscura la nuca, el cálculo del lado al que está girada la cara fue correcto en este caso. Las Figuras 4.7 (a) y 4.8 (a) muestran que nuestro criterio es válido en estos casos, ya que la diferencia de valores es la más alta de las imágenes procesadas. Tres de las imágenes agrupadas incorrectamente (4.7 (d) y 4.8 (d) y (c)), muestran poca diferencia

en los valores más frecuentes. Finalmente, los círculos representan el valor de gris más frecuente en cada imagen.

De las 55 imágenes de perfil completo, el programa calculó 37 de perfil izquierdo y 18 de perfil derecho. Revisando los resultados se encontró que 17 de 37 imágenes fueron incorrectamente calculadas como de perfil derecho (45.94% de error), y 5 de 18 fueron calculadas incorrectamente como de perfil izquierdo (27.77% de error). Los resultados así obtenidos no fueron adecuados para dejarle a este programa resolver el tipo de agrupamiento buscado.

Perfil parcial Se utilizó un programa para este ordenamiento con base en la siguiente observación. Cuando una cara es detectada en una imagen, independientemente del algoritmo empleado, se producen las coordenadas de un recuadro que delimita la cara. Luego, un algoritmo de detección de marcadores faciales calcula las coordenadas de 68 puntos [60]. La Figura 4.9 muestra el modelo correspondiente desarrollado por [4].



Figura 4.9: Modelo i-bug de 68 marcadores faciales [4].

En la Figura 4.8 es posible trazar una línea uniando los puntos 1 y 17 como paralela a otra línea, uniando los puntos 37 y 46. Debido a la simetría de la cara, es de esperar que la distancia entre los puntos 1 y 37 sea igual a la distancia entre los puntos 46 y 17. Cuando esta distancia no es igual bajo cierto umbral, entonces puede sostenerse que la cara está girada o en posición de perfil parcial. Se aplicó este criterio a la colección de imágenes y se llegó a la distribución mostrada en la Figura 4.6. En la Figura 4.10 se muestran 10 de las caras clasificadas como perfil parcial.



Figura 4.10: Ejemplos de caras clasificadas como perfil parcial.

4.3.1.2. Caras de frente

Para este caso sólo se usaron los puntos 37 y 46, que son las orillas exteriores de los ojos. Aquí el criterio de decisión fue que la diferencia entre las distancias de la parte

exterior del cada ojo hacia el borde de la cara no excediera un umbral de 11 píxeles. En la Figura 4.11 se pueden observar unas líneas magentas que unen los extremos de los ojos, y unas líneas amarillas que unen los bordes de la cara un poco abajo de la horizontal de los ojos. El recuadro naranja es el área sobre la que trabaja el algoritmo de marcadores faciales.



Figura 4.11: Muestra de 10 caras de frente.

4.3.1.3. Caras no inclinadas e inclinadas

Un programa calcula la inclinación de la cara con base en los extremos exteriores de los ojos. La idea surgió cuando se consideró usar la ecuación de distancia entre dos puntos para calcular el ángulo de inclinación de la recta que los une. En vez de desarrollar todo el algoritmo de cálculo del ángulo, se decidió sólo tomar los píxeles en el eje vertical y compararlos; así, si están a la misma altura, la imagen está alineada horizontalmente. Si están en un rango menor que 11 píxeles de diferencia, la imagen se considera horizontal. Si están fuera de este rango, la imagen se considera inclinada. Si la diferencia es negativa o positiva, la imagen está inclinada a un lado o al otro. Ver la Figura 4.12 para comparar:



Figura 4.12: Caras ordenadas por 'grado' de inclinación.

(a) Completamente horizontales, o derechas con hasta 10 píxeles de inclinación. (b) Hasta 10 píxeles de diferencia vertical entre las orillas exteriores de los ojos. (c) y (d) caras inclinadas con más de 10 píxeles de diferencia hacia un lado o hacia el otro. Los

subsecuentes usos de ‘grado’ se refieren al número de píxeles que la línea que une los extremos de los ojos se separa de la horizontal.

En la Figura 4.13 se presenta la distribución de ‘grados’ de inclinación calculado por un programa. La mayoría de las 1,780 caras detectadas se consideran ‘derechas’. Cabe notar que originalmente se calcularon 1,201 caras de frente bajo el criterio de diferencia entre distancias entre la orilla exterior del ojo y el borde correspondiente de la cara. En el cálculo de esta gráfica se utilizó el criterio de inclinación de la línea que une las orillas exteriores de los ojos.

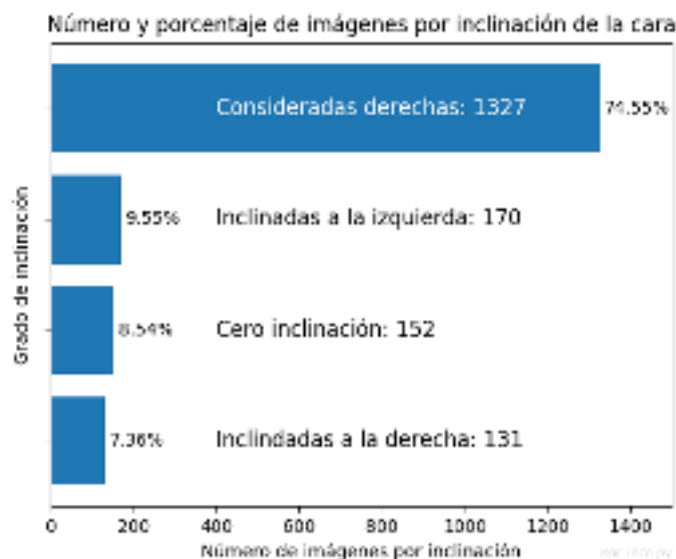


Figura 4.13: Distribución de caras en las imágenes por ‘grado’ de inclinación.

4.3.1.4. Caras con alguna condición particular

Aquí se define ‘condición particular’ como la presencia de una oclusión o a que la persona retratada está haciendo un gesto. Así que puede haber caras neutras, ocluidas y gestuales. Un programa de agrupamiento manual, cuya interfaz gráfica se presenta en la Figura 4.2, se utilizó para este propósito. Ejemplos de cada clase se muestran en la Figura 4.15: (a) neutras, (b) ocluidas y (c) gesticulares.

En la Figura 4.14 se presenta la distribución de caras por ‘condición’, considerando la totalidad de las 2,427 imágenes debido a que no se usó un algoritmo de detección de cara. Los lentes, los sombreros y los tatuajes se consideraron un tipo de oclusión. En el caso de los gestos, la mayoría se da en caras mostrando una sonrisa.

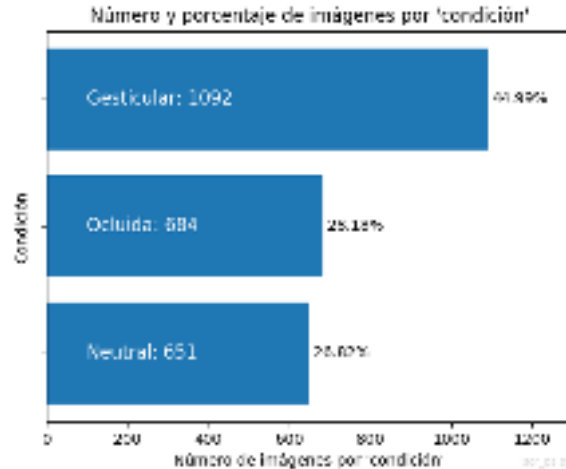


Figura 4.14: Distribución de imágenes por ‘condición’ neutral, ocluida o gesticular.

4.3.2. Caras en imágenes de diferente resolución

La resolución, aunque formalmente tiene asociada una área geométrica, aquí se considera como una condición de nitidez en un área no geométrica sino digital², es decir, en número de píxeles por lado. La mayoría de los recortes de cara tienen una dimensión de 100 píxeles por lado, pero algunos exceden estas dimensiones debido al recorte original. Hay imágenes tomadas con cámaras digitales muy antiguas, o con



Figura 4.15: Ejemplos de caras agrupadas manualmente.

distancia al objetivo de algunos metros, lo que hace que algunas caras salgan de foco. Las más modernas presentan una nitidez mayor y, en algunos casos, uno de sus lados es mayor a 500 píxeles. Se consideró como baja resolución digital, una dimensión de 100 píxeles de alto; como mediana resolución, una imagen con altura de más de 100

²Considerando que el término ‘resolución’ puede implicar dos dimensiones: número de píxeles y longitud, se aclara que en este trabajo sólo se hace referencia al número de píxeles por lado en una imagen.

píxeles y hasta 300 píxeles de alto; y como alta resolución, aquella imagen con más de 300 píxeles de alto.



Figura 4.16: Agrupamiento de imágenes de acuerdo al número de píxeles de alto.

La Figura 4.17 muestra la distribución de las imágenes de acuerdo al alto de la imagen e indirectamente, por resolución, en el sentido que se explicó anteriormente.

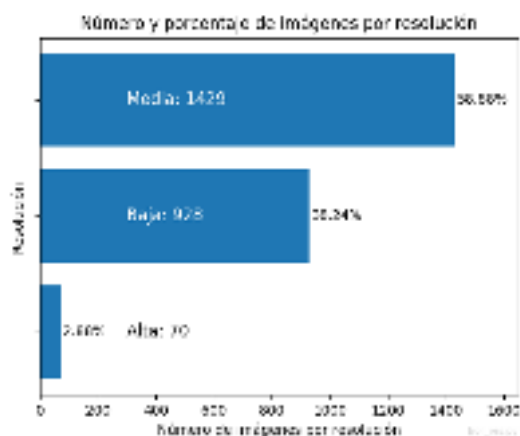


Figura 4.17: Distribución de imágenes por resolución.

Tabla 4.1: Resumen de las dimensiones de la colección para la detección facial.

Dimensión	Mínimo	Máximo	Promedio
Alto	100	526	313
Ancho	51	413	232

En la Tabla 4.1 se muestra un resumen de las dimensiones máxima, mínima y promedio de las imágenes de la colección utilizada para la evaluación de detectores faciales. Las cantidades están en píxeles.

A continuación se observará el comportamiento de cuatro algoritmos de detección facial sobre la totalidad de este conjunto de imágenes preparado como ya se ilustró.

4.4. Algoritmos de detección facial

Se observarán cuatro tipos de detector facial y se usarán los nombres cortos de la Tabla 4.2 para una referencia más breve.

Tabla 4.2: Nombres cortos de los algoritmos.

	Nombre usual	Nombre corto
1	Cascadas de clasificadores débiles	Haar
2	Histograma de gradientes orientados	dlib_hog
3	Red neuronal profunda	cv2_dnn
4	Red neuronal convolucional	dlib_cnn

4.4.1. Cascadas de clasificadores débiles (**Haar**)

Como ya se dijo anteriormente, OpenCV es una biblioteca de funciones especializadas en visión por computadora, originalmente desarrollada en C++ y posteriormente portada a Python. En este trabajo se usa la versión Python 2.7. OpenCV contiene una variedad de funciones que facilitan el procesamiento de imágenes no sólo a nivel básico como filtros de detección de bordes, redimensionamiento de imágenes o selección de color, sino que también usan funciones empleadas en redes neuronales utilizando modelos propios y generando archivos de pesos. Tal es el caso del archivo `haarcascade_frontalface_default.xml` que fue producido por OpenCV basado en el trabajo [72] y que se usa ampliamente en este desarrollo.

La función `detectMultiScale` es una instancia del objeto `CascadeClassifier` que previamente se ha encargado de leer el archivo de pesos citado en el párrafo anterior. Uno de sus parámetros es la imagen en la que se quiere detectar una o más caras, que es convertida a niveles de gris. Otro de sus parámetros es el de `scaleFactor` que se encarga de hacer una pirimidación de la imagen para aplicar la cascada de clasificadores débiles en cada nivel de la pirámide. Esto permite detectar caras que, de otra forma, no serían detectadas, pero en ocasiones hace detecciones incorrectas. Aunque este parámetro se puede manipular, es cuestión del investigador fijar un valor adecuado a la colección de imágenes con la que trabaja. No hay un valor fijo y universal que garantice el mejor desempeño en cualquier grupo de imágenes. En el presente estudio se trabajó con el valor 1.1. En términos de OpenCV es ‘un coeficiente entre el que se dividen las dimensiones de una escala a la otra en el siguiente nivel de la pirámide’ [26].

Otro parámetro importante y sobre el cual se pueden tener mejores resultados de comparación es el de `minNeighbors` o número mínimo de vecinos a considerar. En este caso se encontró mayor efectividad cuando este número se fija en 3. De nuevo, este parámetro funcionó bien en el presente estudio pero no es indicativo que funcione para cualquier grupo de imágenes. Lo mismo puede decirse del siguiente y último parámetro llamado `minSize`, que se refiere a las dimensiones mínimas de la ventana bajo análisis. Ventanas cuadradas de dimensiones menores a ese valor se ignoran. En este estudio se probaron valores entre 10 x 10 y 100 x 100, pero la ventana de 15 x 15 funcionó mejor en general, aunque el mejor resultado de todos, en términos de tiempo, se logró con

una ventana de 20 x 20 píxeles, como se puede apreciar en la Tabla 4.4. La Figura 4.18 muestra un diagrama de flujo para el uso de la función `detectMultiScale`, representada en el diagrama por la caja del proceso ‘Detecta’. Se tiene un grupo de imágenes que se lee una por una. Se verifica que tengan el tamaño correcto para ser procesadas por el detector. De otra forma, se redimensiona. Se carga un archivo de pesos y se realiza la detección. El resultado puede ser usado para almacenar las detecciones, para visualizarlo o para alimentar un proceso de identificación.

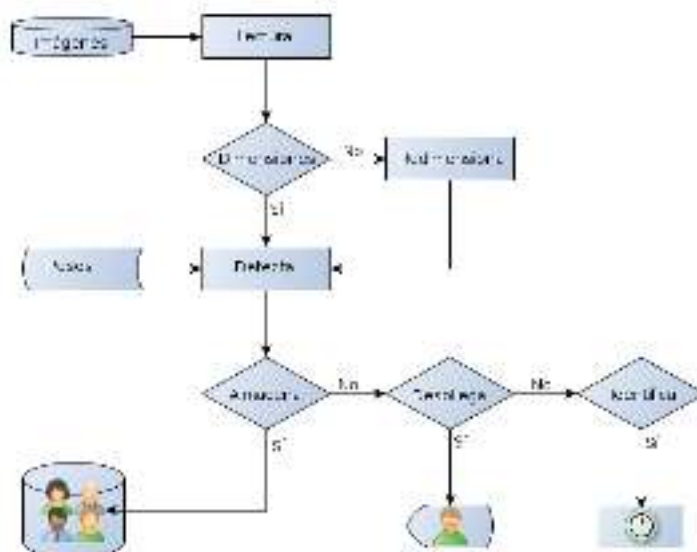


Figura 4.18: Diagrama de flujo de la detección.

Para conocer la mejor combinación de parámetros se ejecutaron 60 pruebas con un programa en el que variaron los siguientes parámetros:

Tabla 4.3: Parámetros de la función `detectMultiScale`.

	Parámetro	Rango de valores
1	Dimensiones	Del tamaño original ('TO' en la Tabla 4.4) a 100, 300 y 500 píxeles de alto
2	Número de vecinos	3, 5, 7
3	Ventana mínima	$(\ell \times \ell)$ $\ell = 10, 15, 20, 50, 100$

Luego de aplicar cada prueba sobre las 2,427 imágenes de la colección, se encontró que la mejor combinación de ellos fue:

1. No aumentar las dimensiones originales de la imagen.
2. Usar 10, 15 o 20 píxeles como ventana mínima.
3. Mantener el número de vecinos en 3.
4. Para una mayor velocidad, usar una ventana mínima de 20 píxeles.

La Tabla 4.4 muestra un resumen de los mejores resultados de las 60 pruebas, mientras que en la Figura 4.19 se presenta otra forma de visualizarlos.

Tabla 4.4: Resultados de las 60 pruebas de detección

Dimensiones	Ventana mínima	Sí de- tectadas	No de- tectadas	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

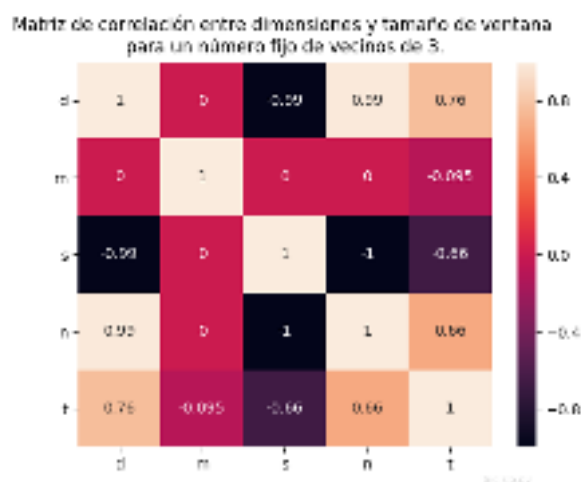


Figura 4.19: Gráfica de correlación entre los parámetros de dimensión d y ventana mínima m sobre los resultados de detección s , n y tiempo t .

En la Figura 4.19, obsérvense las columnas s , n y t . El parámetro s representa las detecciones, n las no detecciones, y t el tiempo en segundos de cada prueba. Como el número de vecinos quedó fijo en 3, su efecto ya está considerado al provenir este conjunto de datos de ese rango de prueba. La columna d , de dimensiones, dice que está correlacionada negativamente de manera indiscutible con la detección de caras; mientras mayor es el número en que se aumenta el tamaño de la imagen, menor es la cantidad de caras detectadas. Se cree que esto se debe a que el algoritmo usa una ventana deslizante de 24×24 píxeles, y que al aumentar las dimensiones de la imagen, el efecto de aplicar los filtros disminuye. Recíprocamente, la no detección está directamente correlacionada con las dimensiones de la imagen; mientras mayores son las dimensiones de la imagen, menos caras se detectan (véase el último renglón de la Tabla 4.4). En cuanto al tiempo, se puede afirmar que definitivamente existe una relación directa entre éste y las dimensiones de la imagen; a mayor tamaño, más tiempo. Las dimensiones de la ventana mínima no influyen en el número de detecciones para un mismo rango de dimensiones de imagen.

Ahora se muestran ejemplos de los resultados de las detecciones. Se incluyen aspectos a favor y en contra, tales como detecciones múltiples, detecciones incorrectas y no detecciones. En la Figura 4.20 se puede ver: la imagen original con la detección incorrecta, un recuadro de aquella y los datos de ambas imágenes. La Figura 4.21 muestra el caso de detección múltiple: (a) muestra la imagen original; (b) muestra las dos detecciones; (c) presenta una detección incorrecta; (d) muestra la detección de mayor área que corresponde a la detección correcta; (c) y (d) se agrandaron y separaron para mostrarlas con mayor detalle. El caso de las no detecciones se observa en la Figura 4.22, en la que se ve una imagen con dos personas. (a) Cada una fue recortada como en (b) y (c). El resultado es que sólo una fue detectada como en (d) y la otra no³.

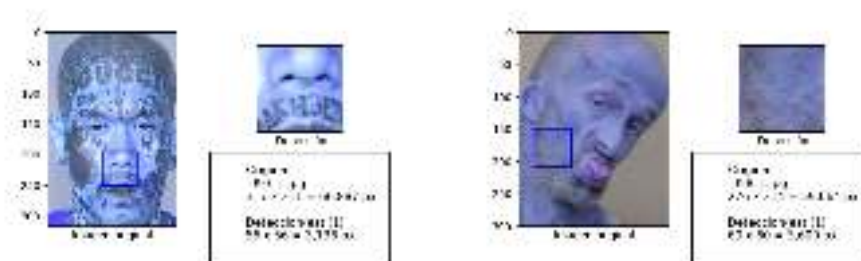


Figura 4.20: Ejemplos de detección incorrecta.

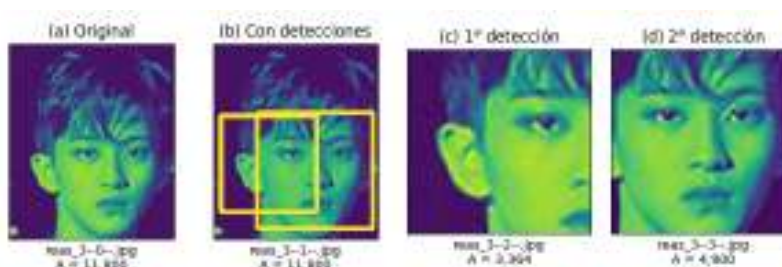


Figura 4.21: Ejemplo de detección múltiple.



Figura 4.22: Ejemplo de cara no detectada.

³Como se explicó en su momento, las caras en las fotos grupales fueron separadas automáticamente. En este caso, cuando se observó que una de estas caras no se detectó, se separaron manualmente para ilustrar el problema de la no detección. Además, se tenía especial interés en observar el sesgo racial que argumentan quienes se oponen al reconocimiento facial.

Cuando se estaban analizando los resultados de incrementar el tamaño, se encontró que en algunas imágenes ocurría más de una detección (hasta 3) y se buscó la causa. Una de ellas es que aumenta el número de ventanas generadas en las que puede haber una cara y que cumplen el umbral de decisión. Como ejemplo, véanse las Figuras 4.23 y 4.25.

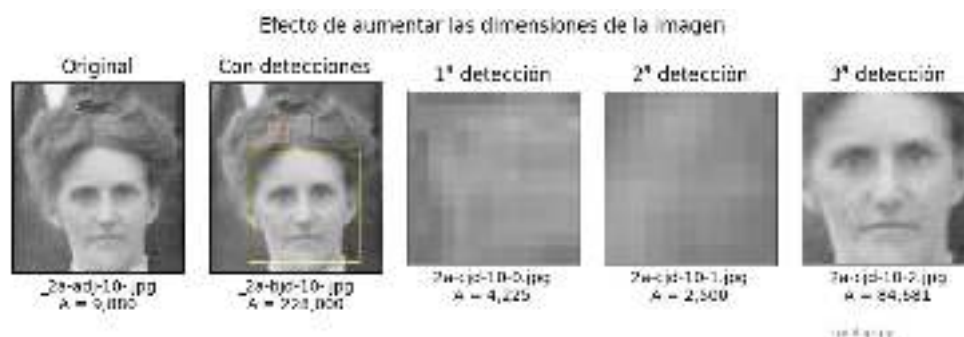


Figura 4.23: Efecto de aumentar las dimensiones de la imagen.



Figura 4.24: Detección única sin modificar dimensiones originales.

En la Figura 4.24 se ve que no era necesario redimensionar. En la Figura 4.23 primero se muestra la imagen original con área de 9,800 píxeles. Luego la imagen redimensionada a 300 píxeles de alto, sobre la que se realiza la detección. Se ven claramente tres recuadros, cada uno representando una detección. Los dos primeros son incorrectos y se muestran en los recuadros 1ª detección y 2ª detección. La última detección, y la correcta, se muestra al final con un área inferior a la original. La letra 'A' al pie de la subfigura denota área en píxeles. En la Figura 4.24 la imagen de la izquierda tiene las dimensiones originales. En la de en medio se muestra la detección única y correcta. La imagen de la derecha es el recuadro detectado y ya recortado. En la Figura 4.25 se muestran detecciones múltiples, siendo incorrecta una de ellas.

Se hizo un programa para ordenar manualmente los resultados de la no detección y tratar de explicarlos. La Figura 4.26 muestra la interfaz de ese programa. Puede verse, a la izquierda, una lista de nombres de archivo. En el centro se va mostrando la imagen del rostro conforme se selecciona de la lista. A la derecha hay una serie de botones que representan algún tipo de atributo observado en la imagen. Los botones de arriba hacia abajo representan una cara: neutra, con algún tipo de oclusión, con algún tipo de gesto, de perfil, pixelada, con destello, con sombra, inclinada, recortada, borrosa. Conforme



Figura 4.25: Ejemplo de detección múltiple en ojos y boca.

se va seleccionando una imagen y un botón, las imágenes se van acomodando en las carpetas correspondientes.



Figura 4.26: Programa de agrupamiento manual de imágenes.

En la Figura 4.27 pueden verse 10 condiciones y una excepción en las condiciones en las que se agruparon las caras. La excepción se refiere a los errores en la detección que no pueden ocurrir en la no detección, es decir, encontrar caras donde no las hay. La gráfica está ordenada de forma ascendente de detecciones y empieza por la izquierda. Sólo se detectó una de las 51 caras recortadas. El algoritmo no es bueno en este caso. En las imágenes sombreadas tampoco es muy bueno, ya que no detectó alrededor de la

mitad de las caras en esa condición. En el caso de las caras con destello, superó el 77% de aciertos. En el caso de caras inclinadas no se comportó tan bien. Las caras ocluidas representan el 16.33% del total de imágenes, pero de ellas detectó el 67.68%, lo cual, para este estudio, se consideramos aceptable. Los errores, debe reiterarse, se refieren a que el algoritmo detectó algo que le ‘pareció’ una cara donde no la había. El tipo de error más frecuente fue dentro de una cara y no fuera de ella.

Tomando como referencia a la Figura 4.27 se comparó cómo se comportó el algoritmo en cada tipo de cara. El caso de las caras de perfil es similar al de las caras sombreadas. El algoritmo no es muy bueno en este caso. El caso de las caras borrosas se refiere a recortes de imágenes grupales en los que algunas personas estaban fuera de foco. Las imágenes eran de buena resolución y las personas al centro y al frente eran bastante nítidas. Las de los extremos y más atrás, no. Las caras neutras se refieren a caras típicamente de frente y casi sin expresión. Aquí se alcanzó cerca de 94% de aciertos⁴. Se consideró una cara ocluida aquella con lentes, con algún tipo de gorro, con tatuajes, alteraciones faciales (objetos insertados en la cara) y caras parcialmente visibles debido a la cabeza de alguien más. Aquí el algoritmo mostró un desempeño de casi el 68% de casos correctos. Las caras pixeladas se refieren a imágenes de baja calidad o resolución. Cabe observar que éstas, junto con las imágenes borrosas que se procesaron en este trabajo, constituyen cerca del 24% de la totalidad, y la eficiencia del algoritmo en ambos casos refleja que es muy bueno para este tipo de imágenes. Finalmente se observan las caras gesticulares. La mayoría de ellas se refiere a personas con una sonrisa. El algoritmo resultó 95% eficaz en un corte de casi el 29% de la colección de imágenes.

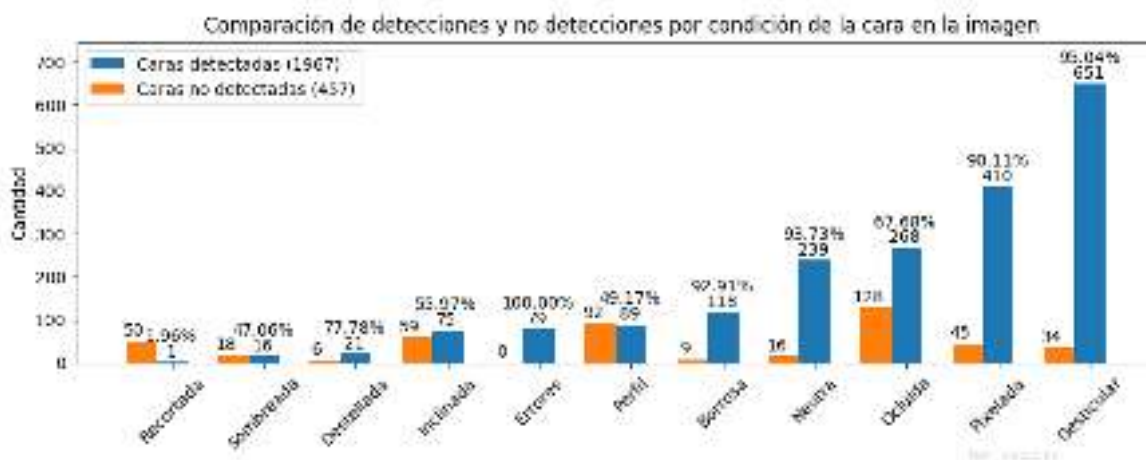


Figura 4.27: Comparación de detecciones y no detecciones.

Como resultado de agrupar manualmente las condiciones de la cara en la imagen, se obtuvo el mosaico mostrado en la Figura 4.28. Muchas caras satisfacen más de una

⁴Se aprovechó este comportamiento en un programa de detección (*enrolamiento*), haciendo que el usuario mantenga la cara en un perímetro rectangular iluminado. Cuando los ojos están alineados horizontalmente, se captura un cuadro de video.

condición pero se favoreció la oclusión, la gestualidad y la inclinación. Se tomaron al azar 5 muestras por caso.



Figura 4.28: Muestra aleatoria de diferentes condiciones.

Con esto concluye la implementación de las pruebas de desempeño del algoritmo de Cascada de clasificadores débiles.

4.4.2. Histograma de gradientes orientados (`dlib_hog`)

Como ya se expuso anteriormente, existe un mecanismo de detección de caras basado en el concepto de Histograma de gradientes orientados o HOG, del inglés *Histogram of Oriented Gradients*. Igual que en el caso de Haar, existe una implementación del trabajo original, pero esta vez hecha en la biblioteca `dlib` [20] de Davis E. King. Este modelo de detección facial usa cinco filtros relacionados con la posición de la cara (uno de frente, dos de perfil total y dos de perfil parcial). La función `dlib.get_frontal_face_detector()` carga el modelo `dlib`, y la función `hogFaceDetector()` se encarga de hacer la detección sobre la imagen que se le pasa como parámetro. Para esta función hay un segundo parámetro que permite que la imagen se escale. Usualmente el valor es 1, y al aumentarlo a 2 o 3, el tiempo de procesamiento se incrementa notablemente.

Se produjo otra detección de caras desde cero, tomando las mismas imágenes grupales e individuales de la sección anterior, por lo que se hará un sólo cambio de parámetro y se mostrará el resultado de este algoritmo.

La Tabla 4.5 presenta los resultados de aplicar el algoritmo de detección `dlib_hog` a la colección de 2,427 imágenes. Se tabulan caras detectadas, caras no detectadas y tiempo.

Tabla 4.5: Resultados de aplicar el algoritmo `dlib_hog` a la colección de imágenes.

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

En la Tabla 4.5 se puede observar que, sin que el algoritmo cambie las dimensiones de las imágenes, el número de detecciones no es tan bueno, pero es bastante rápido. Cuando el algoritmo aumenta las dimensiones al nivel inmediato, las detecciones aumentan aproximadamente 8%, pero el tiempo se cuadruplica. Cuando el tamaño se incrementa al siguiente nivel, el número de detecciones disminuye y el de no detecciones crece. El tiempo aumenta 15 veces.

Se tomaron los resultados de la tercera prueba por ser numéricamente similares a los de Haar (parámetro de escala = 2). Se hizo una comparación pensando que habría fuertes coincidencias, pero no fue así. Vale la pena expresarlo como sigue: se quería ver lo ocurrido en los no detectados por ser una cantidad menor. De esta forma se reunieron $456 + 457 = 913$ caras no detectadas en conjunto. De éstas, 477 fueron no detectadas por alguno de los dos algoritmos y 218 fueron no detectadas por ambos algoritmos. Cada algoritmo dejó de detectar 238 de las caras de su contraparte. De esta forma, al sumar 238 no detectadas en lo individual, más 218 no detectadas en conjunto, se obtienen 456 en el algoritmo `dlib_hog` y $218 + 239 = 457$ en el caso Haar.

Inicialmente hubo una confusión porque se esperaba que hubiera más coincidencia en las no detecciones. Sin embargo, sólo 218 imágenes entraron en esa categoría. Se hizo una clasificación manual por condición de manera similar al caso Haar con

el resultado de la distribución de la Figura 4.29. Las caras de perfil fueron las menos detectadas. En la Figura 4.30 se puede ver una muestra aleatoria de 5 imágenes de cada condición. En el caso de las destelladas, de las que sólo hay tres ejemplares, se ocuparon los espacios faltantes con una imagen de relleno para evitar que el espacio quedara en blanco.



Figura 4.29: Distribución de condiciones de caras no detectadas.



Figura 4.30: Cinco imágenes de cada categoría no detectada.

Se hizo una medición indirecta para determinar qué hace el algoritmo con la proporción de incremento, porque la documentación no lo indica, y se encontró que no hay un gran incremento en las dimensiones sino un remuestreo en el que se aumentan

muy ligeramente las dimensiones de la imagen. Se tomaron dos ejemplos y se tabularon los resultados en la Tabla 4.6

Tabla 4.6: Coordenadas de los puntos inicial y final de los rectángulos de detección facial usando el algoritmo `dlib_hog`.

Imagen	Nivel	X1	Y1	Ancho	Alto	$X2=X1+Ancho$	$Y2=Y1+Alto$
A	0	16	54	86	87		
	1	16	56	90	90	106	146
	2	16	48	93	93		
	3	6	50	97	96	103	146
B	0	13	37	72	72		
	1	13	38	75	75	88	113
	2	18	40	64	65		
	3	5	32	80	81	85	113

En la Tabla 4.6 se puede ver que el origen de la detección es casi siempre el mismo punto, y que crece en alto y ancho en los niveles de 'piramidación' 1 y 2. En el nivel 3, el origen de la detección se recorre un poco hacia el origen de la imagen; el final de esta detección casi coincide en el eje x , y es el mismo en el eje y , respecto al nivel 1.

A continuación se muestran dos imágenes de las caras analizadas en la Tabla 4.6. En ambas, la primera imagen es el recorte individual original. Luego siguen las detecciones conforme se incrementa el nivel de piramidación del algoritmo. Se esperaba un cambio importante en dimensiones pero, como se puede ver, el incremento es marginal.

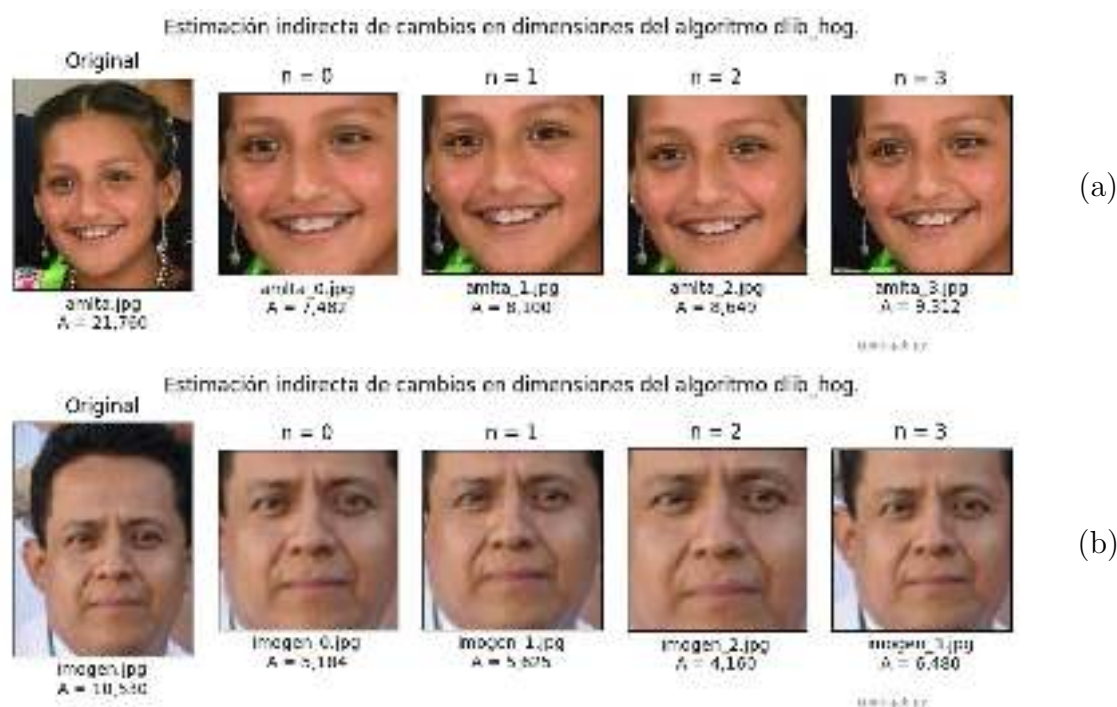


Figura 4.31: Dos imágenes tomadas al azar de la colección.

Haciendo una superimposición de las detecciones con $n = 1$ y $n = 3$ de la imagen

(a) de la Figura 4.31, se obtiene la ilustración de la Figura 4.32, en la que se puede ver que la primera detección encaja perfectamente en la segunda. También se observan, en la esquina inferior derecha, las detecciones con niveles 1 y 3. Parece irrelevante, pero cabe preguntarse cuál es el beneficio de la piramidación si consume tantos recursos y la imagen no fue agrandada. Sólo cambiaron las coordenadas de los puntos inicial (arriba a la izquierda) y final (abajo a la derecha) del recuadro. También cabe preguntarse si el algoritmo incrementa internamente las dimensiones, hace la detección y posteriormente mapea las coordenadas a la imagen original.



Figura 4.32: Ejemplo de cómo coinciden las coordenadas de las detecciones.

Para conocer el efecto de aplicar el algoritmo `dlib_hog` a imágenes redimensionadas fuera del propio algoritmo, modificando el parámetro de nivel de piramidación, se utilizó la función `resize()` de OpenCV sobre esta misma imagen, pero primero se ajustó a 100 píxeles de alto y luego se aumentó a 300 y 500 píxeles de alto. La Figura 4.33 muestra el efecto de dicho cambio sobre la versión de 500 píxeles.



Figura 4.33: Efecto de aplicar el algoritmo de detección `dlib_hog` en 4 niveles de piramidación sobre una imagen redimensionada fuera del algoritmo.

En la Figura 4.33 se puede ver que las cuatro detecciones son diferentes, aunque las de niveles 0 y 3 tienen muy cercana la esquina inferior derecha. Los niveles 0, 1 y 2, prácticamente comparten la línea base.

Con esto concluyen las pruebas de desempeño del algoritmo de detección facial mediante Histograma de gradientes orientados, incluido en la biblioteca `dlib`.

4.4.3. Detector facial con red neuronal de OpenCV (`cv2.dnn`)

OpenCV tiene un mecanismo de detección facial basado en su propia implementación de redes neuronales. Las funciones que se utilizarán son las que se muestran en la Tabla 4.7.

Tabla 4.7: Funciones del módulo `dnn` de OpenCV.

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

De las cuatro funciones mostradas en la Tabla 4.7, la única que soporta parámetros del usuario es la segunda, misma que se desarrolla a continuación. La función mencionada tiene la siguiente firma:

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

La función recibe en primer lugar la imagen que se va a procesar, como un arreglo NumPy. El segundo parámetro es un factor de escala que típicamente es 1 o menor. Luego vienen las dimensiones de la imagen al entrar a la red de convolución. Se tomará uno de los valores típicos (alto = 299 píxeles, ancho = 299 píxeles). Luego, como cuarto parámetro, está el valor promedio que se desea restar a la imagen. Este valor se pasa como valor único o como tripleta en formato (R,G,B), donde cada valor representa el promedio que se restará de la imagen en cada uno de sus tres canales. Una de las razones por las que se resta el promedio es para tener más uniformidad en los valores. De esta forma estos no se disparan los valores base y tampoco disparan los pesos calculados por la red [18]. Se usan los siguientes valores debido a que son los que representan los promedios de la colección de imágenes con la que se procesó la red: (104.0, 177.0, 123.0). Finalmente, se tiene la opción de intercambiar los colores rojo y azul debido a que diferentes funciones de lectura de imágenes tienen su propio formato. Esta función espera que el orden de colores venga en BGR, pero el formato de promedios a sustraer es RGB, así que debe ponerse como `true` para respetar el orden de colores. El sexto y último parámetro es una bandera lógica para ‘indicar si queremos centrar el recorte de las imágenes. Si no es puesto en `true`, la imagen de entrada es recortada a partir del centro, de forma que la dimensión más corta es igual a la dimensión correspondiente en tamaño, y la otra dimensiones será igual o mayor. Sin embargo, si se deja en `false`, redimensionará respetando las proporciones.’ [16].

Se apreciará cómo afecta modificar los parámetros de manera similar a como se hizo con Haar.

Se realizó un ciclo de cambio de dimensiones sobre este conjunto de valores: {0, 229, 300, 500}. Se usó el 0 para decirle al programa que no realice cambios al tamaño de la imagen. No se usó 100 como en los anteriores casos debido a que la función marca error cuando se la pasa un tamaño inferior al mínimo de entrada (224). El valor 229 es un valor típico en el procesamiento con esta red. Otro valor típico es 227. Sólo se usó el valor 229. Los valores 300 y 500 son valores empleados en los dos algoritmos anteriores y se conservarán para ser consistentes en estas pruebas. No se hizo intercambio de colores y no se recortó la imagen (`crop = false`).

La Figura 4.34 indica que prácticamente no hay efecto en redimensionar las imágenes externamente, ya que todas ellas son redimensionadas al tamaño de la entrada a la red: 229 x 229 píxeles en este caso.

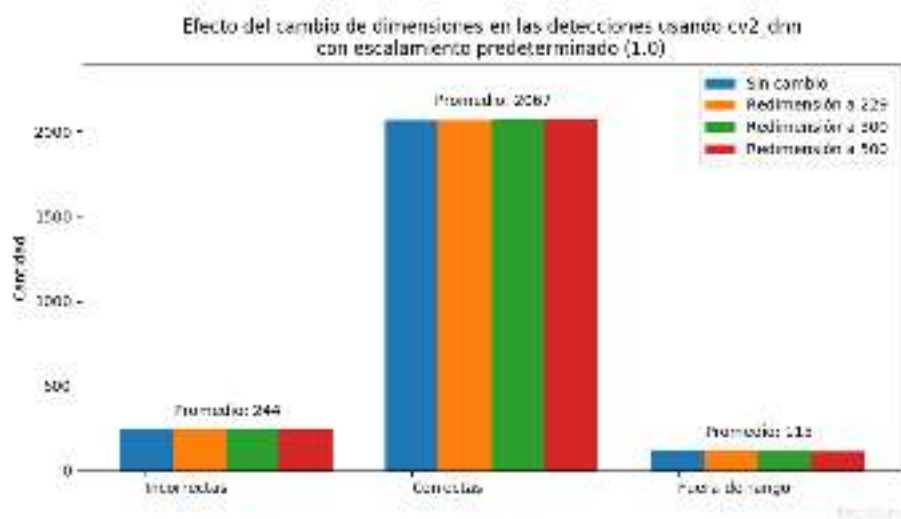


Figura 4.34: Efecto del cambio de dimensiones en las detecciones con el soporte de redes neuronales de OpenCV.

Otra forma de visualizar los resultados de cada redimensionamiento se muestran en el mosaico de la Figura 4.35. Se observó que los valores regresados por el detector representan un porcentaje de las coordenadas de los puntos inicial y final del rectángulo respecto al alto y ancho de la imagen. Las imágenes son tratadas en el segundo cuadrante del plano cartesiano. De esta forma, su origen queda en la esquina superior izquierda. El punto inicial de cualquier detección está cerca del origen y se extiende sobre su diagonal hasta un punto más alejado del origen. No se encontraron detecciones que se comporten de otra forma. Sí se encontraron detecciones, incorrectas, con su *origen fuera* de la imagen y/o *final fuera* de la imagen. Esto no ocurrió con los dos detectores que fueron analizados antes.

Otro de los valores que vienen en el objeto regresado por la función anterior es el nivel de confianza, cuyo rango de valores significativos está entre 0 y 100. Este valor es engañoso. Sólo coincide cuando el recuadro no es visible debido a que es muy bajo. Sin embargo, hay niveles de confianza muy altos, asociados con recuadros en el segundo

cuadrante⁵ que se extienden fuera de la imagen. Hay ocasiones que reporta niveles de confianza del orden de miles, y se decidió usar esto como un indicador de error.

En los dos algoritmos Haar y dlib_hog se utilizó el valor retornado por el detector para saber si había ocurrido alguna detección. En este caso fue una sorpresa notar que ninguna imagen reportó una no detección, pero revisando los resultados se observó que cierta cantidad de caras no habían sido correctamente enmarcadas y otras ni siquiera tenían marcado un recuadro. Esta es la razón por la que se pusieron en la categoría de detecciones incorrectas. Las imágenes agrupadas como fuera de rango se deben a que el algoritmo reporta sus dimensiones fuera de los límites de la imagen. Sólo el inicio del recuadro cae dentro de la imagen. Las dimensiones que caen fuera pueden ser sobre el eje x , el y , o ambos.

Los tiempos reportados por cada lote son tres minutos para la entrada directa de la imagen y cuatro para cualquier redimensión. Se aclara que estos tiempos dependen de la cantidad de procesos que esté atendiendo la computadora. En una máquina con modo gráfico habilitado, cualquier algoritmo se tarda más debido a las múltiples tareas del sistema operativo. Además, la implementación requiere múltiples lecturas y escrituras de imágenes. Durante las pruebas se encontró un promedio bajo de 0.0988 segundos por imagen, y uno alto de 0.1236 segundos por imagen.



Figura 4.35: Mosaico de imágenes incorrectamente detectadas sin escalamiento.

En la Figura 4.35 se puede observar que la mayoría de recuadros de detección están trazados en el segundo cuadrante (poniendo el origen del plano en el centro de la imagen). Las imágenes en las que casi no se nota una letra E (de la que se habla adelante), tienen mayor resolución que aquellas en las que esa letra es más visible. Se puede apreciar que el nivel de confianza bajo no fue utilizado para aceptar o rechazar una

⁵Respecto al centro de la imagen.

detección, puesto que hubo imágenes de muy baja resolución con bajo nivel de confianza con un adecuado registro del recuadro en la cara. Se observó que las detecciones en las que la ordenada inicial de la detección era mayor que 30% de la altura, mostraban un recuadro en el segundo cuadrante, básicamente de la punta de la nariz a la izquierda de la cara, en la mayoría de los casos, y eso fue nuestro indicador de detección incorrecta.

En la Figura 4.36 se muestra el detalle de una de las caras del mosaico anterior. En primer lugar, la cantidad en color gris (57.35) representa el nivel de confianza de la detección. Se puede ver que es un número bajo. La cara está completamente de perfil, pero hubo muchos casos similares en que se delimitó la cara con precisión. Los números de color azul representan: los dos primeros, las coordenadas del punto inicial del recuadro, en porcentaje de alto y ancho de la imagen. Los dos segundos son el alto y el ancho del recuadro. La letra E, de error, se puso en las coordenadas del punto inicial. En amarillo se muestra la sección de recuadro que resuelve el algoritmo y que se encima a la imagen con la función `cv2.rectangle()`, de OpenCV.

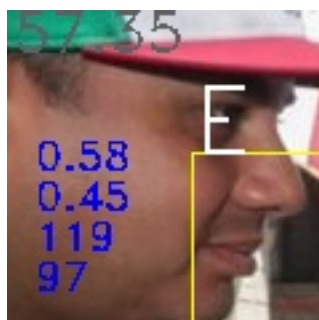


Figura 4.36: Detalle de anotaciones sobre una cara de perfil.

Después de ver el comportamiento del algoritmo ante el cambio exterior de dimensiones, se hizo un ciclo de cambios al parámetro de escala, el cual es un proceso interno del algoritmo. Su efecto tampoco fue observable aunque el tiempo de procesamiento sí aumentó.

En la Figura 4.37 se presenta un mosaico de imágenes tomadas del conjunto de caras no detectadas, cuando se usó el factor de escala 2.0 en el conjunto de dimensiones mencionado arriba.

Surgió la pregunta de si había cierta consistencia en la no detección. Se hizo un programa para realizar una comparación exhaustiva de los resultados de la corrida con redimensionamiento a 500 píxeles por lado y factor de escala 2.0. Aquí se obtuvo una mayor consistencia de detecciones incorrectas con sólo tres elementos de diferencia, pero con cuatro imágenes diferentes.

Se observó que por cada parámetro modificado se genera un conjunto de resultados distinto. Ya sea que se deje fijo uno y se cambie el otro o se cambien los dos, el número de elementos incorrectos era similar. Esto sugirió que eran las mismas imágenes no detectadas. Al comparar unas con otras, se observó que los conjuntos de imágenes eran diferentes en casi un 50%. Esto llevó a analizar un ejemplo donde se mantuvo fijo



Figura 4.37: Mosaico de imágenes incorrectamente detectadas con escalamiento 2.0.

el nivel de escala interno, pero se cambiaron las dimensiones de las imágenes antes de comenzar el procesamiento. El resultado está en la Figura 4.38. Aquí se comparó un conjunto de 278 elementos con otro conjunto de 281 elementos. Se encontró que todas las caras del primero están en el segundo. Las cuatro imágenes de la Figura 4.38 son aquellas que no están incluidas en el primer conjunto⁶.

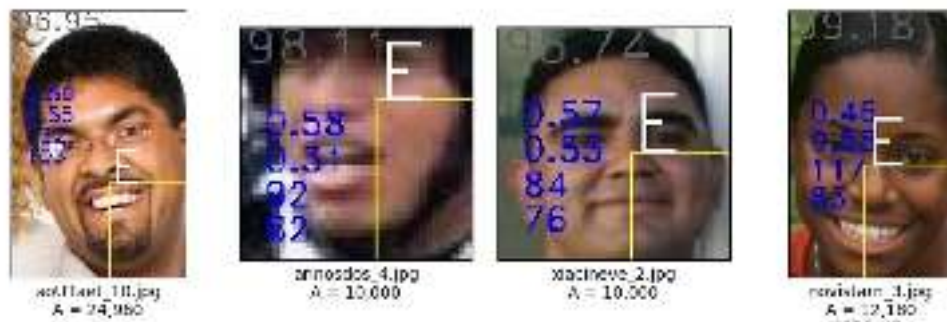


Figura 4.38: Imágenes no detectadas cambiando las dimensiones.

⁶No se hace referencia a la diferencia en el número de elementos de cada conjunto.

En resumen, el algoritmo `cv2_dnn` encontró más de la mitad de las caras no detectadas por los anteriores, incluyendo una buena cantidad de caras de perfil completo, otras ocluidas y otras normales. El algoritmo no resultó eficiente en las imágenes pixeladas. En general, el tiempo no fue favorable. Conviene revisar la Tabla 4.7, ya que en aquella se explica el funcionamiento de esta red neuronal a partir de las cuatro funciones enumeradas.

4.4.4. Detector facial con red neuronal de `dlib` (`dlib_cnn`)

`dlib` también tiene una implementación de detección facial basada en redes neuronales. El archivo de pesos `mmod_human_face_detector.dat` es resultado de su procesamiento. El modelo de red se llama `dlib.cnn_face_detection_model_v1`.

Este modelo convolucional es mucho más preciso que el modelo basado en HOG de la Sección 4.4.2, pero es más demandante en recursos de cómputo, de suerte que se recomienda el uso de GPU para lograr una velocidad razonable [37].

Al principio se instancia un objeto de la siguiente manera:

```
detector=dlib.cnn_face_detection_model_v1  
("mmod_human_face_detector.dat")
```

Posteriormente se lee una imagen (o se alimenta un cuadro de video) utilizando la función de lectura de imágenes de `scikit`. Ya en la versión 19.16.0 de `dlib` se puede usar un lector propio (`dlib.load_rgb_image()`). Por ahora se usará el de `scikit`:

```
imagen = io.imread(archivo)
```

Luego de tener el contenido de la imagen, éste se pasa por el detector, el cual regresa una tupla de datos que se mostrará después.

```
detecciones = detector(imagen, escala)
```

El segundo argumento indica si se debe remuestrear la imagen para hacerla más grande y permitir la detección de más caras. Es parecido a la función de `dlib_hog` y sólo acepta valores enteros. Se hicieron pruebas con 0, 1 y 2.

Este detector regresa un objeto `mmod_rectangles` que contiene una lista de objetos `mmod_rectangle` a los que se tiene acceso mediante su iteración. Tiene dos miembros: un objeto `dlib.rectangle` y un nivel de confianza. El objeto `rectangle` contiene las coordenadas de los puntos inicial (arriba a la izquierda) y final (abajo a la derecha), como números enteros. El nivel de confianza es un número real que llega a ser mayor que uno (en las pruebas, máximo=1.143899, mínimo=0.005268).

Se creó un programa exploratorio antes de almacenar cualquier resultado. Se decidió efectuar un ciclo para valores de remuestreo de 0 (sin remuestreo), 1 y 2. Se observó que el primero es muy rápido pero poco preciso. El segundo es medianamente rápido pero más preciso. El tercero es muy lento y sólo rescató 4 elementos más que la prueba precedente. Se considera inviable un remuestreo a nivel 2 en equipo sin GPU.

Se intentó procesar un lote con remuestreo de 1.2 pero, como en el caso de `dlib_hog`, la función de detección no admite piramidación fraccional. Luego se hicieron pruebas redimensionando las imágenes a una altura de 500 píxeles con niveles de remuestreo 0 y 1, y se puso especial atención a los resultados del segundo caso debido a que mostró un número mayor de detecciones en un tiempo razonable. Se revisaron los resultados y se encontró que una buena parte de detecciones (235) venían en 0 o

contenido nulo. Otra parte, 38 elementos, mostraban la detección de manera vertical (recuadro más alto que ancho). De estos, sólo una detección fue horizontal (recuadro más ancho que alto). Se buscó la razón de tales anomalías y se encontró que en esos casos los valores `objeto.rect.left()` y `objeto.rect.top()` venían: uno, otro, o ambos, con valores negativos. Se invirtieron los signos en tales casos y dejó de haber imágenes nulas o en forma de franjas. Adelante se muestran ejemplos de cada caso. Como parte de los datos que el objeto `detecciones` contiene, están las coordenadas que encierran una detección. Las coordenadas sirven para extraer un recuadro y almacenarlo como una cara detectada.

Se hizo una clasificación de condiciones de la imagen igual que en los tres algoritmos anteriores, y se encontró que tres categorías no tenían representante (neutra, destellada y gesticular), y las otras cuatro tenían menos de 5 elementos (inclinada, sombreada, recortada y borrosa). Las no detecciones se concentraron en las categorías de pixelada y ocluida. En la Tabla 4.8 se muestra un resumen de resultados de los cuales es posible concluir que la piramidación interna es prohibitiva con redimensionamiento y con nivel de remuestreo superior a 1.

Tabla 4.8: Resumen de resultados del algoritmo `dlib_cnn`.

Prueba	Remuestreo	Redimensionamiento ^a	Detecciones	No detecciones	Tiempo (s)	Tiempo promedio (s)
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

^apíxeles de alto.

Un fenómeno que se observó más en Haar, se refiere a las detecciones múltiples. Con este método sólo ocurrieron 4. Ahora se presentarán algunas gráficas similares a las de los apartados anteriores.



Figura 4.39: Caras no detectadas sólo en siete categorías.

Se inicia con la distribución de condiciones de caras no detectadas en la Figura 4.39, y a continuación el mosaico de la Figura 4.40 muestra las siete categorías halladas con este método. Las imágenes de relleno se deben a que no se alcanzaron 5 muestras de cada clase.

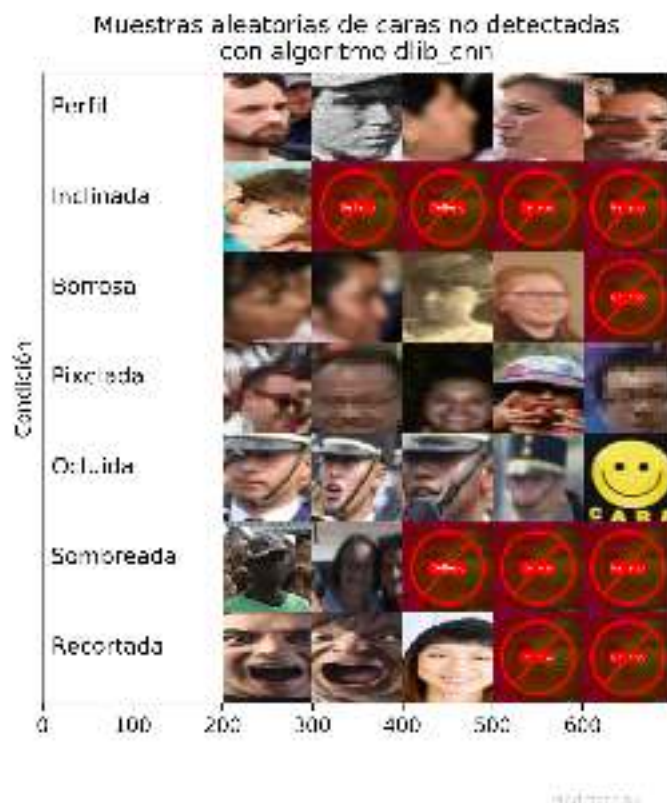


Figura 4.40: Mosaico de imágenes de cada categoría no detectada.

Inicialmente se pensó que las ‘detecciones dobles’ se debían a errores en el algoritmo, pero una inspección de los resultados aclaró el punto. Se trata de imágenes donde una cara está en primer plano y otra está parcialmente ocluida por la primera. En la Figura 4.41 se puede ver que sólo dos de los recuadros no registran correctamente toda la parte de la cara. Además de ocluidas, las caras de las detecciones secundarias están recortadas.



Figura 4.41: Ejemplos de aparente detección doble con dlib_cnn.

En la Figura 4.42 se observan las caras que consistentemente fueron no detectadas por los tres algoritmos antes vistos pero que este algoritmo sí detectó. Puede verse que algunas presentan condiciones difíciles.



Figura 4.42: Algunas de las caras no detectadas por los otros algoritmos.

Se ha mostrado la implementación de cuatro diferentes algoritmos de detección facial. Cada uno presenta características diferentes. Se han exhibido resultados de aplicar diferentes parámetros de configuración a cada una de las funciones de detección, siendo algunas más modificables que otras. En la Tabla 4.9 se observa una comparación cuantitativa de los algoritmos.

Tabla 4.9: Comparación de algoritmos de detección facial.

Concepto	Algoritmo			
	Haar	HOG	dlib	cv2
Aciertos	1,931	1,975	2,376	2,067
No aciertos	496	456	51	244
No detecciones	0	4	0	115
Tiempo (s)	47	353	1,408	269
Parámetros de mejor desempeño	Tamaño original, ventana mínima 20 píxeles, 3-NN	Tamaño original, remuestreo de 1 ^a	Parámetro de escala: 2	Escalamiento predeterminado a 1.0, redimensionamiento a 500 píxeles de alto

^aCon el parámetro de remuestreo = 2 se detectaron 3 caras más, pero se tardó 5,847 s.

4.4.5. Estudio de tiempos de proceso de los algoritmos de detección respecto al área y cantidad de imágenes procesadas

Inicialmente no se tenía considerado este apartado pero en atención a la sugerencia del doctor José Giovanni Guzmán, sinodal de esta tesis, se decidió aplicar el enfoque siguiente:

1. Incrementar el número de imágenes de 14,530 a 15,897.
2. Dividir el total de imágenes en lotes múltiplos de 1,000.
3. Medir tiempos estrictamente en el ciclo de detección, dejando de lado otras operaciones tales como: verificar el número de detecciones, mostrar el recuadro de la detección, registrar archivos de comas y graficar los resultados.
4. Usar archivos de comas para resultados intermedios y para generar las gráficas.

Supuestos

1. Siempre se supuso que el comportamiento de los cuatro algoritmos era lineal.
2. Relacionado con el punto anterior, se supuso que conforme aumentaba el número de imágenes, aumentaba el tiempo de proceso de manera lineal.

No previsto

1. No se sabía que las dimensiones de la imagen afectan la velocidad de proceso. En efecto, influyen en la velocidad, pero no de forma muy diferente a un comportamiento lineal.

Resultados

La primera gráfica generada no confirmaba lo supuesto, como se puede observar en la Figura 4.43(a). Parece contraintuitivo que procesar las primeras mil imágenes represente casi un tercio del tiempo requerido para procesar la totalidad. Se atribuye este efecto a que las primeras mil imágenes tienen dimensiones mayores debido a que así las ordena el sistema operativo en la carpeta. En la Figura citada, también se observa que a mayor tamaño y mayor cantidad de imágenes, el tiempo de proceso va aumentando. Cuando las dimensiones disminuyen en dos tercios y el número de imágenes crece, el tiempo crece muy poco (véase una muy ligera meseta entre las 4 y 13 mil imágenes con un pico en las 10 mil). Cuando las dimensiones vuelven a crecer de 14 mil a las 15,897 del conjunto, el tiempo vuelve a crecer notablemente.

En base a lo anterior, se hicieron mediciones con diferentes lotes de imágenes. Se tomaron piezas de mil de forma aleatoria y no se apreciaron cambios notables. Entonces se decidió hacer lo siguiente:

1. Guardar los nombres de las imágenes en un arreglo.
2. Hacer un ciclo de lectura de cada imagen.
3. Tomar las dimensiones de cada una.
4. Con el nombre de archivo y sus dimensiones (área en pixeles de cada imagen), formar otro arreglo.
5. El arreglo anterior, ordenarlo por dimensiones y guardarlo en un archivo de comas.
6. Para cada algoritmo, usando el archivo de comas, tomar lotes crecientes en miles, hacer la lectura de la imagen, hacer la detección y tomar el tiempo.
7. Por cada lote, registrar el número de lote, área promedio y tiempo en segundos.
8. Graficar resultados.

77

debe a que este algoritmo hace la detección usando tres operaciones en cascada, a diferencia de las otras tres, a las cuales sólo se les pasa la imagen en formato de arreglo NumPy.

Concluimos este estudio de tiempo estableciendo que su comportamiento en todos los algoritmos de detección es lineal, aunque se ve influido por las dimensiones de las imágenes procesadas.

4.5. Funciones de pérdida

4.5.1. Introducción

Antes de entrar al análisis de las tres funciones de pérdida de este apartado, se presentará un par de ayudas visuales que permitan entender el espacio sobre el cuál se mueven dichas funciones. Primero, en la Figura 4.44, se muestra una hiperesfera (reducida a tres dimensiones), y la cara de dos personas. Cada cara está enmarcada por un rectángulo de color. En ese mismo color aparece un punto, así como sus coordenadas espaciales⁷, que representa el lugar en el hiperespacio que le corresponde a un vector de representaciones faciales de cada persona. Uno de los algoritmos que se observarán aquí se encarga de establecer la máxima separación o distancia en el hiperespacio de 128D, de cada identidad.



Figura 4.44: Distancia como medida auxiliar de convergencia.

En la Figura 4.45 se aprecia otro efecto de aplicar ese mismo algoritmo, y mediante la técnica de reducción de dimensiones conocida como PCA, es posible situar, en un espacio de tres dimensiones, los puntos que representan a cada una de las caras de las personas incluidas en una colección de imágenes, creada exprofeso para este análisis. Puede observarse un recuadro del lado izquierdo que muestra el detalle de dos identidades que se encuentran alejadas de las demás pero que entre sí están cercanas. Hay una razón: las identidades 5 y 6 corresponden a hermanas gemelas. A pesar de ello se puede trazar una línea de separación que atraviesa dos puntos de ambas identidades, y sólo dos valores de la identidad 6 caen como atípicos en la región de la identidad 5.

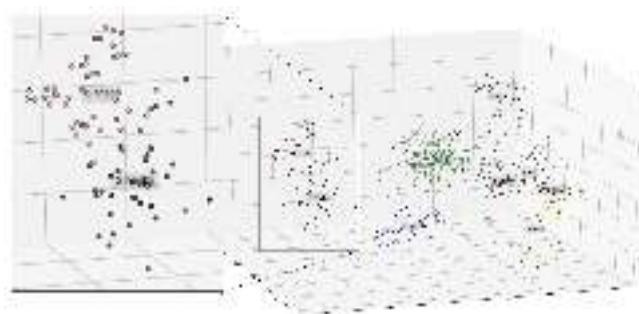


Figura 4.45: Efecto de reducir un hiperespacio de 128D a 3D.

⁷En términos de un vector unitario

En adelante se van a comparar tres funciones de pérdida. Para comprobar la superioridad de una sobre las otras se utilizaron varias redes convolucionales. Se empezó con una versión modificada de una red llamada Xception [8] en la que se analizaron las funciones de pérdidas propuestas. Cuando se usó la de tripleta, se observó que la red no convergía, sino que se alejaba del objetivo, por lo que se intentó probar con otra red con la que se logró mejor resultado, pero de cualquier forma no fue el resultado previsto. Se probaron otras cuatro redes ([43], [69], [9] y [64]), y de todas se seleccionó la que se detalla adelante debido a su rapidez, y precisión lograda. La Tabla 4.10 enlista el nombre y tipo de las capas de la red seleccionada; el formato de salida de cada capa, el número de parámetros, cuántos son modificables y cuántos son fijos.

Tabla 4.10: Resumen de la red empleada.

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	896
leaky_re_lu (LeakyReLU)	(None, 28, 28, 32)	0
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 14, 14, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262272
leaky_re_lu_3 (LeakyReLU)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258
Total params: 355,778		
Trainable params: 355,778		
Non-trainable params: 0		

La Tabla 4.11 muestra la gran diferencia en número de parámetros cuando la red recibe como entrada una imagen de 180 x 180 píxeles. Los reportes que se muestran en la Tabla 4.10, Tabla 4.11 y Figura 4.46, son generados por las utilerías de TensorFlow, una vez que el modelo de red [64] ha sido definido, instanciado y compilado.

Tabla 4.11: Fragmento del resumen de la red con entrada de 180 x 180 píxeles.

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 180, 180, 32)	896
...		

Total params: 8,760,770
Trainable params: 8,760,770
Non-trainable params: 0

La Figura 4.46 es un diagrama de bloques de la misma red. La diferencia respecto a la Tabla 4.10 es que no incluye el número de parámetros pero incluye el formato de los datos de entrada y salida de cada capa. La figura que originalmente se generó en modo vertical se editó para mostrarla en modo horizontal. En la Figura 4.48 se muestra el grafo de cálculo [32] utilizado por la red neuronal de este trabajo. La explicación del mismo va más allá del tema del presente estudio y sólo se añade que es un diagrama generado por un aditamento de TensorFlow llamado TensorBoard [68], que es una herramienta diseñada para la visualización del trabajo interno de las redes neuronales con las que se esté trabajando.

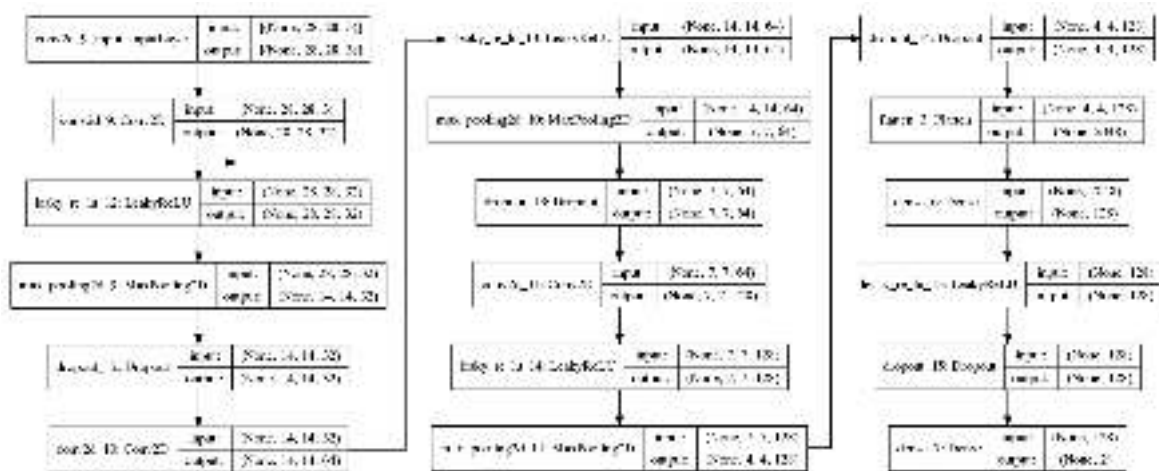


Figura 4.46: Diagrama de bloques de la red empleada.

Otra representación de esta red es como la que se muestra en la Figura 4.47⁸. Esta ilustración en tres dimensiones muestra mejor la arquitectura de la red porque nos brinda una apreciación espacial que el diagrama de bloques no nos proporciona. En aquel sólo leemos números. En esta representación tridimensional, podemos apreciar la profundidad de las capas de convolución y sus conjuntos de filtros, además del efecto que producen las capas de agrupamiento sobre las dimensiones. También vemos una representación de las capas 'aplanadas' o completamente conectadas (*dense*, en inglés).

⁸Se hizo una considerable modificación de la herramienta en: <https://github.com/HarisIqbal88/PlotNeuralNet>. visitada: 2022-08-27.

4.5.2. Fragmento de código para procesar la red

A continuación se presentan partes sobresalientes del código en Python para el procesamiento de la red. Se puede encontrar que algunos bloques y algunas líneas están comentadas para explicar su uso. Aunque el código usualmente no contiene texto con acentos, aquí se marcan por apego a la ortografía.

```
# sección de importación de módulos
import tensorflow as tf
from tensorflow import keras
...

# ruta a los datos:
ruta='/home/usuario/folder/donde/estan/las/imagenes'

# parámetros de la red
image_size = (28, 28)
batch_size = 32
epocas = 20
num_clases = 2

# lectura y acondicionamiento de imágenes
X = tf.keras.preprocessing.image_dataset_from_directory( # función muy importante que ahorra
                                                         # tiempo en la preparación de los datos
                                                         ruta,
                                                         validation_split=0.2, # lar ruta definida arriba
                                                         subset="training", # parte la colección en 80% proceso y 20% validación
                                                         seed=1337, # indicación de que el subconjunto es para proceso
                                                         label_mode='categorical', # semilla de generación de lotes aleatorios
                                                         image_size=image_size, # la red es categórica. Es para armar las clases
                                                         batch_size=batch_size, # dimensiones de la imagen
                                                         batch_size=batch_size, # tamaño del lote
                                                         )

# definición del modelo:
modelo = keras.Sequential([capas]) # en la Diap. 21 están las capas empleadas

# Si se quiere analizar el grafo de la red, hay que dar un folder de registro,
logdir="logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")

# y una función de retrollamada a TensorBoard:
callbacks = [keras.callbacks.TensorBoard(log_dir=logdir)]

# se compila el modelo. Aquí se le pasa el tipo de función de pérdida,
# el optimizador y el tipo de medición.
modelo.compile(loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
               optimizer=keras.optimizers.Adam(), metrics=['accuracy'])

# se ejecuta el modelo y el resultado se instancia en el tensor hist (de historial)
# que nos sirve para mostrar los resultados
hist=modelo.fit(
    train_ds, epochs=epocas, callbacks=callbacks, validation_data=Xv,
)

# rescatamos los valores del tensor de historial para graficar resultados:
# (las gráficas de rendimiento mostradas más adelante son creadas con los valores
# anteriores)
ganancia = hist.history['accuracy']
pérdida = hist.history['loss']

# otro resultado se obtiene mediante:
teseval = modelo.evaluate(Xp) # la función recibe la partición de imágenes de prueba (Xp)

# esta línea produce la matriz de comparación en base a un arreglo de clases
# y un arreglo de estimaciones; éste se forma con la función modelo.predict\_classes(Xp),
mc = tf.math.confusion_matrix(labels=clases, predictions=estimaciones).numpy()
# y se obtiene una gráfica como la mostrada adelante.
```

```

# La siguiente función es del módulo sklearn
rc = classification_report(tes_ds, predicted\_classes, target_names=target\_names)
# y este produce la matriz mostrada en la Figura 4.47.

# Para visualizar mejor los resultados, se hace un mosaico con las 32 imágenes
# de un lote aleatorio de la partición de prueba. Este se hace con el mismo arreglo
# de estimaciones indicado arriba y se compara contra las clases reales.
# la Figura 4.48 se creó con este código:
# algunos parámetros de control del ciclo
row=3
col=11
i=1
# funciones de matplotlib
plt.figure(figsize=(1.4*col,2*row))
plt.subplots_adjust(bottom=0,left=0.1,right=.99,top=0.80,hspace=.4)

# ciclo de formación del mosaico
for t, la in zip(im,cl): # im, cl = arreglos NumPy de imágenes y clases
    plt.subplot(row=3,col=11, i)
    i+=1
    p = tf.expand_dims(t, axis=0) # extrae la imagen
    pa = np.argmax(modelo.predict(p), axis=-1)[0] # estima su clase
    imu=t.astype(np.uint8) # la convierte a enteros
    plt.imshow(imu) # la agrega al mosaico
    roloc='r' # ...para saber qué texto poner
    if la==pa:
        roloc='g'
    testo='estmda: %s\n real: %s'%(str(pa),str(la)) # se abrevia por espacio
    plt.title(testo, color=roloc)
    plt.axis("off")

# fin del fragmento de código

```

4.5.3. Resultados de las funciones de pérdida empleadas

Como ya se indicó, la colección de imágenes se formó con una parte tomada de internet, en tanto que otras fueron extraídas de revistas en formato .pdf. En la Tabla 4.12 se muestra un resumen de la colección y sus particiones. Antes de entrar a la red, las imágenes tienen características variables (número de colores, dimensiones, resolución), pero cuando entran a la red todas se convierten a un cuadrado de 28 x 28 píxeles y son tratadas como de color, independientemente de si vienen como escala de grises. Se definen lotes de 32 imágenes y se procesa la red por 15 épocas. Se utiliza un optimizador Adam con tasa de convergencia de 0.001. En algunos ejercicios se modificó el número de épocas y la tasa de convergencia. También se hicieron modificaciones a la función de activación de las capas de convolución, densas⁹ y de salida, pero en lo general se mantuvo el modelo original con el que mejores resultados se obtuvieron y que fue adaptado de [64].

Las funciones de pérdida que se analizaron fueron:

- 1) Error medio absoluto o *Mean absolute error*, que aquí se denota como EMA, como representante de distancia euclidiana.
- 2) Entropía cruzada categórica dispersa o *Sparse categorical cross entropy*, que aquí se denota como EC.
- 3) Pérdida por tripleta o *Triplet loss*, que aquí se denota como 3L.

⁹La literatura también se refiere a este tipo de capa como ‘completamente conectada’.

Tabla 4.12: Detalles de la colección para funciones de pérdida.

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

A continuación se presenta la Tabla 4.13 que compara los resultados de los métodos analizados. Inicialmente se concentran en dos términos conocidos como pérdida y ganancia (al término ganancia se le conoce en inglés como *accuracy* o *precision*). Debido a que en este trabajo se consideran términos complementarios, y como a uno invariablemente se le conoce como pérdida, se decidió llamarle al otro ganancia, pensando que no se pierde el sentido de lo que expresa cada término. Una pérdida baja cercana a cero, y una ganancia alta cercana a 1, son señales de que la red ha convergido en una buena solución y es capaz de hacer estimaciones bastante aceptables.

Tabla 4.13: Resultados en pérdida y ganancia de los tres métodos analizados.

Tipo de pérdida					
EMA		EC		3L	
Pérdida	Ganancia	Pérdida	Ganancia	Pérdida	Ganancia
0.4966	0.4968	0.2298	0.9031	0.5273	0.4986

En la Tabla 4.13 se puede ver que la que dio mejor resultado corresponde a la de entropía cruzada categórica dispersa. En cuanto a la pérdida en valor absoluto promedio, se aprecia que ambos valores son muy parecidos. Durante las pruebas se observó que sin importar la modificación de parámetros, tales como el optimizador, la tasa de convergencia o la función de activación, siempre se encontraba una oscilación entre 0.5 de pérdida y 0.5 de ganancia, o entre 1 y 0.5, respectivamente. En el caso de pérdida por tripleta también se observaron oscilaciones, aunque sus resultados no son tan parecidos como el caso de EMA.

Entre las pruebas que se hicieron también se modificó el número de ciclos de proceso también conocido como épocas. Se hicieron pruebas con 5, 15, y 50 ciclos, y se observó que 15 fue suficiente. En ciertos casos se observó que con 5 ciclos era suficiente para notar la oscilación. Esto daba lugar a interrumpir el proceso y reintentarlo con modificaciones a los parámetros.

Un parámetro modificado que dio excelentes resultados fue el de definir una imagen de entrada de 28 x 28 píxeles. Originalmente se intentó con 180 x 180. Debido a los resultados alentadores de pérdida por entropía cruzada, se decidió no emplear la red VGG [65] que utiliza imágenes de 224 x 224. Otro factor por el que se decidió 28 x 28 es el de tiempo de procesamiento.

Utilizando una máquina convencional con 4 GB de RAM y procesador Centrino, el uso de imágenes de 180 x 180 píxeles ocasionó que la máquina dejara de responder

con una colección de sólo mil imágenes por clase. Luego se procesó en otra máquina con 8 GB de RAM y procesador de 4 núcleos. En esta máquina se pudo procesar correctamente la red con imágenes de 180 x 180, pero cuando se bajó la dimensión a 28 x 28, el proceso se agilizó de dos horas por ciclo a 5 minutos por ciclo.

Como se dijo cuando se vieron los pobres resultados de pérdida EMA y pérdida 3L, se decidió no continuar su evaluación. En cambio, la entropía cruzada permitió ahondar en el análisis y por ello se presentan los siguientes resultados, empezando por la evolución del proceso de convergencia. En la Figura 4.44 se muestran las gráficas de ganancia y pérdida. En redes neuronales es usual hacer una evaluación de pérdidas y ganancias en dos conjuntos de datos conocidos como conjunto de procesamiento y conjunto de validación. La literatura llama al primero *trainning dataset*, y al segundo *validation dataset*. El segundo es indicativo de qué tan bueno puede ser el modelo para estimar valores no procesados. El caso aquí analizado trata de construir un modelo que estima correctamente si la cara que se le presenta en la etapa de prueba es de mujer o de hombre. Cabe recordar que la cara que se le presenta no pertenece a ninguno de los conjuntos de procesamiento o validación.

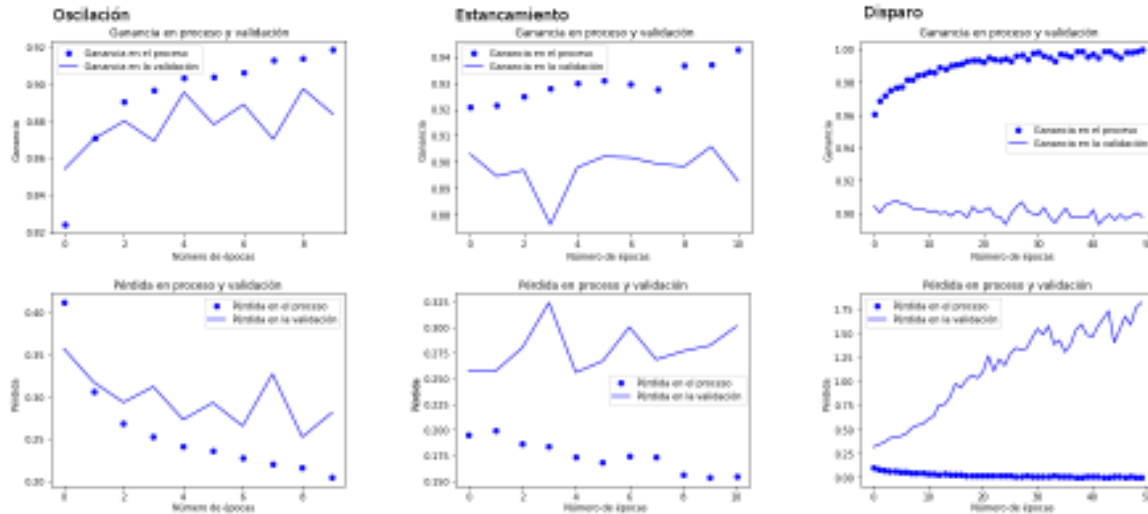


Figura 4.49: Ejemplos de inestabilidad en las redes probadas.

La Figura 4.49 muestra tres ejemplos de lo que puede llamarse *inestabilidad* en el procesamiento de la red debido a que pueden encontrarse, básicamente, tres tipos de comportamiento. En primer lugar, es posible observar una *oscilación* de valores cuando se procesan datos de validación. Sus resultados oscilan respecto a un crecimiento suave con los datos de proceso, tanto en ganancia como en pérdida. El segundo caso, *estancamiento*, muestra que no hay una tendencia clara hacia la convergencia. El último par de gráficas muestra un disparo de valores y un alejamiento en las curvas, yendo cada una por su lado, mostrando una aparente falta de representación del conjunto de validación respecto al conjunto de datos de procesamiento. Las Figuras 4.50 (a) y (b) muestran que no es así, puesto que se está trabajando con los mismos conjuntos de procesamiento y validación. Es posible que la explicación más razonable se deba a que las funciones de pérdida empleadas no son adecuadas para procesar este caso binario, con esta red específica. Se puede observar que se utilizó un número variable de épocas

(de 9 a 50) y que en los casos de oscilación y estancamiento, el proceso de convergencia se interrumpió cuando éste no mostró resultados alentadores.

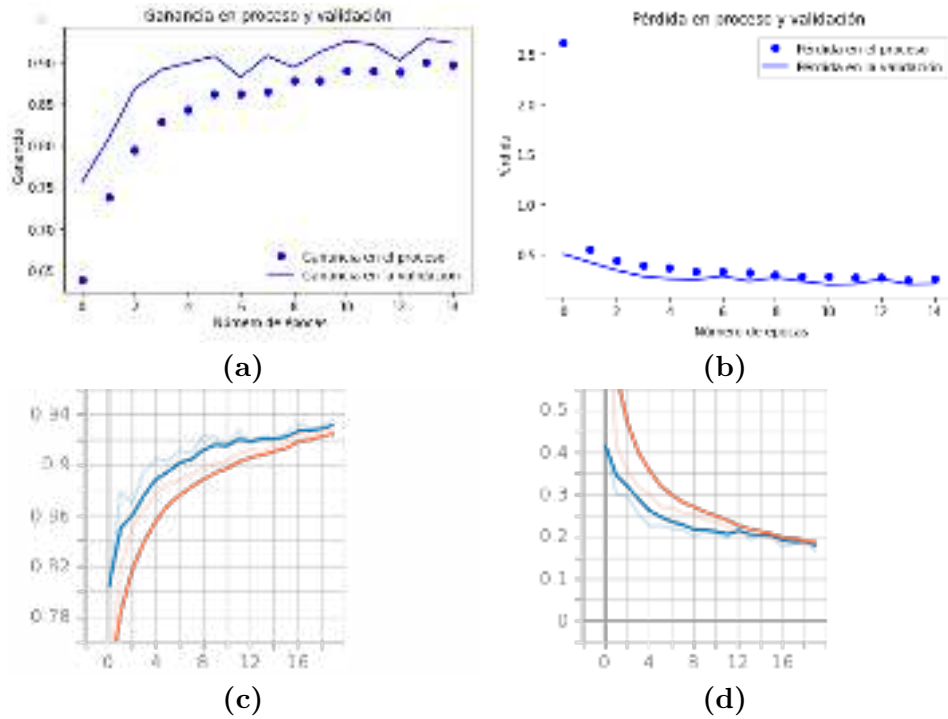


Figura 4.50: Gráficas de rendimiento de la red.

Haciendo referencia a la Figura 4.50, en (a) se presenta el comportamiento de la ganancia; y en (b) se presenta el comportamiento de la pérdida. Ambos cálculos ocurren durante el proceso sobre la partición de datos que le corresponde. Se puede ver que en el caso de la ganancia se parte de un valor cercano a 0.65, y en el segundo ciclo ésta ya está próxima a 0.80. A partir de allí se mantiene una curva suave con una sola oscilación notable en el octavo ciclo. El caso de la pérdida presenta una caída drástica desde aproximadamente 1.8 a 0.5 en el primer ciclo, y luego un descenso estable con un ligero pico en el ciclo octavo. En el caso de la ganancia, la separación entre las ganancias de proceso y validación es de sólo 0.024, mientras que la diferencia entre pérdidas de proceso y validación es de 0.0323. La curva de la gráfica de pérdida es engañosa debido a que parece que la diferencia es despreciable. Esto se explica porque, aunque el número de ciclos es el mismo, los rangos de valor inicial y final son diferentes (eje y). Las Figuras 4.50 (c) y (d) son las gráficas producidas por TensorBoard con base en una función especial de retrollamada que se pasa como parámetro de la función `fit()` de la red. En estas últimas, las líneas de color naranja representan el trazo de proceso, en tanto que las azules, la validación. Las líneas de color más tenue representan valores atípicos durante el proceso de convergencia.

El modelo ejecutado con función de pérdida de entropía cruzada produce al final los valores indicados en la Tabla 4.10 y da lugar a un siguiente paso de calcular los valores estimados sobre el conjunto de prueba. La herramienta empleada (TensorFlow) permite estimar todo el conjunto con una sola instrucción. El resultado, en forma de

un tensor de estimaciones y un tensor de valores reales, se pasa por otra función que entrega una matriz de comparación. La literatura se refiere a ésta como *confusion matrix*, o matriz de confusión. Con ella se establece una comparación entre el número de valores estimados contra valores reales. Cada vez que se hace una estimación se tienen dos casos posibles: el valor estimado es el correcto o el valor estimado es incorrecto. Las combinaciones de acierto y error dan lugar a un recuadro (o matriz) como el mostrado en la Figura 4.51. Este cuadro es creado con los datos del objeto `hist` mostrado en el código de la Sección 4.5.2.

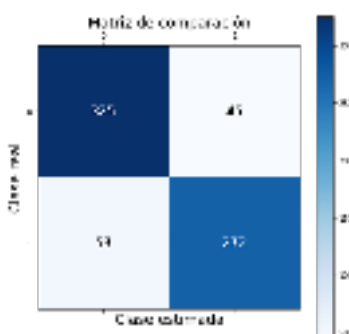


Figura 4.51: Matriz de comparación. Las clases son: 0 para hombres y 1 para mujeres.

Los datos de la Tabla 4.14 servirán para explicar la Figura 4.51, ya que en estas cantidades se basan los resultados de la prueba. Una prueba de este tipo permite tener una idea de qué tan bueno es un modelo para generalizar ante casos desconocidos.

Tabla 4.14: Detalle de las imágenes de prueba.

Partición	Cantidad
Imágenes para prueba	695
Imágenes de prueba de la clase 0	370
Imágenes de prueba de la clase 1	325

Haciendo referencia a la Figura 4.51, el primer recuadro de arriba a la izquierda indica los casos en que el modelo estimó que una imagen era de la clase 0 y en realidad era de esa clase. La literatura le da un nombre a esto: ‘cierto positivo’. El modelo estimó que 325 de las 370 imágenes de hombres eran de hombre y 45 de mujer, es decir, produjo un resultado incorrecto en 45 casos. De nuevo, la literatura tiene un nombre para esto y le llama ‘falso positivo’. Luego, en el recuadro de abajo a la izquierda se observa una celda que corresponde al número de veces que el modelo estimó incorrectamente la clase de mujeres, diciendo que eran hombres. A esto se le conoce como ‘falso negativo’. La Figura 4.51 dice que 53 imágenes de hombre fueron incorrectamente estimadas como de mujer. En la última celda se aprecia el caso en que se estimaron correctamente 272 imágenes como de mujer y efectivamente eran de mujer. A esto le llaman, confusamente, ‘cierto negativo’.

Esto parece confuso debido a que se basa en una herramienta de evaluación con antecedentes en la Segunda Guerra Mundial, durante el desarrollo de los sistemas

de radar que, efectivamente, identificaban naves enemigas como enemigas, es decir, acertaba en la posibilidad negativa en el sentido de amenaza.

Luego de esta matriz también se puede presentar un cálculo complementario que corresponde a una variedad de combinaciones entre esas cuatro posibilidades de ciertos y falsos. En la Tabla 4.12 se observa lo que se conoce como reporte de clasificación. Este reporte se crea con la función `clasification_report(clases, estimados)` de la biblioteca `sklearn`, donde `clases` es el arreglo Numpy de clases, y `estimados` es otro arreglo Numpy que se crea con la función `modelo.predict_classes(x)`, donde `x` es un lote de 32 elementos de la colección de prueba, tomados aleatoriamente.

Tabla 4.15: Reporte de clasificación.

	precision	recall	f1-score	support
0.0	0.89	0.88	0.87	371
1.0	0.84	0.88	0.86	325
accuracy			0.87	695
macro avg	0.87	0.87	0.87	695
weighted avg	0.87	0.87	0.87	695

La Tabla 4.15 muestra el reporte de clasificación generado. No se modificaron ni el formato ni los textos para preservar el reporte original. En éste se observa, sin encabezado, las clases 0 (hombres) y 1 (mujeres). La columna ‘*precision*’ dice que un porcentaje de 0.89% se estimó correctamente para la clase 0 (hombre) y 0.84% para la clase 1 (mujer). La columna ‘*support*’ hace referencia al número de elementos (imágenes) de cada clase. ‘*recall*’ y ‘*f1-score*’ son expresiones aritméticas que juegan con los datos con el ánimo de darles sentido. El primero se conoce en español como sensibilidad, y el otro no tiene traducción, pero es una relación entre precisión y sensibilidad. Aritméticamente representan las relaciones $tp/(tp + fn)$ y $2tp/(2tp + fp + fn)$, respectivamente, donde tf = acierto negativo, tp = acierto positivo, fp = error tipo uno, y fn = error tipo dos.

Se efectuó otra evaluación que también se hace sobre la colección de prueba y, en este caso se obtuvo:

```

escor = modelo.evaluate(Xp, verbose=0)
22/22 [=====] - 2s 76ms/step - loss: 0.3320 - accuracy: 0.8676
print(escor)
>>>[0.31692734360694885, 0.8589928150177002]
```

donde a la función `evaluate` se le pasa el tensor con la colección de prueba (X_p).

Finalmente, y haciendo uso de la herramienta `sklearn.metrics`, se calcula la precisión de las estimaciones:

```

from sklearn.metrics import accuracy_score as ac
print('La precisión evaluada es: ',ac(estimaciones,clases))
>>> La precisión evaluada es: 0.84375
```

En la Figura 4.52 se muestra el proceso de un lote aleatorio de la colección de prueba. Se observan 32 imágenes de mujeres y hombres. Para el programa, la clase 0 corresponde a hombres y la clase 1 a mujeres. Arriba de cada imagen se muestra un par de números en color verde o rojo. Cada identificador puede estar unido por un signo '=' cuando hay coincidencia, o por '!=' cuando hay diferencia. Cuando la estimación coincide con la clase real, el texto se presenta en verde, y cuando no hay coincidencia, en rojo. Para este lote la precisión fue de $1 - 5/32 = 0.84375$.



Figura 4.52: Estimación de clases sobre un lote aleatorio de la colección de prueba.

Es posible resumir este proceso con la siguiente lista de pasos:

- 1) Preparación.- Se establece la ruta a los datos, se leen, se acondicionan, se particionan y, de manera opcional, se extienden artificialmente.
- 2) Configuración.- Se define el modelo, sus parámetros, las funciones auxiliares requeridas y se compila. El modelo puede cambiarse en cualquier momento que no satisfaga un umbral de rendimiento mínimo, o presente alguna inestabilidad, como en los casos de: falta de convergencia, estancamiento o disparo de valores.
- 3) Se ejecuta el modelo.
- 4) Se evalúa, se ajusta y se da por bueno.

El proceso de encontrar la mejor función de pérdida no es independiente del contexto de la aplicación. En este ejercicio se trabajó en un caso de reconocimiento binario de clases de imágenes. Hay otros contextos tales como el de series de tiempo o de múltiples características presentes o no en el objeto de análisis. Por ejemplo, condiciones atmosféricas, precios de casas, enfermedades o flores.

Se considera que las funciones de pérdida en que interviene la distancia euclidiana (EMA y 3L) no fueron adecuadas en este caso binario, y que EC fue superior debido a que trabaja en distribuciones probabilísticas.

4.6. Clasificadores

4.6.1. Introducción

Para cumplir el Objetivo **iii)** se van a analizar tres tipos de clasificador muy utilizados en la academia. Aunque ya no se escriben trabajos científicos enfocados en destacar su importancia, se observa que se utilizan en muchos de ellos. Basta mirar los resultados de las referencias en el sitio Google Académico. En la Tabla 4.16 se muestran los resultados aproximados de buscar por autor o por algoritmo.

Tabla 4.16: Resultados de búsqueda por autor y algoritmo.

Autor	
Karl Pearson @pca	14,800
NS Altman @knn	189,000
Vladimir Vapnik @svm	16,000
Algoritmo	
PCA ^a	6.4 millones
k -NN	329,000
SVM	1.61 millones

^aPCA es uno de los algoritmos que usa `TensorBoard` para reducir las dimensiones de los datos y presentarlos en un hiperespacio de agrupación.

A diferencia de secciones anteriores, en este apartado se usará una colección de imágenes recopiladas en tiempo real con un mecanismo de ‘enrolamiento’. Éste es parte de un sistema de control de acceso llamado *Amaxayac* de uso personal. Se recopilaron entre 25 y 40 imágenes de 9 identidades¹⁰. La captura en este programa tiene dos características: por un lado, durante el enrolamiento presenta un recuadro para que el usuario coloque la imagen de su cara dentro de él. En este paso de captura, el usuario puede girar o inclinar su cara mientras dure el proceso. Por otro lado, la captura de la imagen en el paso de reconocimiento se hace sólo cuando los ojos de la cara están alineados horizontalmente. Para realizar la prueba de este algoritmo se empleó una parte separada de 5 imágenes por identidad y otras tomadas fuera del sistema para probar la habilidad del reconocedor ante cambios en las condiciones de captura de imágenes.

La Tabla 4.17 muestra la cantidad de imágenes de cada identidad, las cuales se nombrarán como números consecutivos.

Las imágenes capturadas durante el enrolamiento son alimentadas a cada algoritmo. Esto produce un conjunto de valores que se mantiene en memoria como en el caso de PCA y k -NN, o bien, se almacenan en un archivo de formato *Pickle*¹¹ para su utilización posterior, como en el caso de SVM. A esos valores se les denominará vectores de características. También es correcto llamarles representaciones vectoriales faciales.

¹⁰Una de ellas llegó a 237 debido a las pruebas.

¹¹Algoritmo de serialización de objetos tipo byte.

Tabla 4.17: Relación de imágenes por identidad.

Identidad	Enrolamiento	Prueba	Distintas
Uno	237	13	8
Dos	52	16	11
Tres	37	1	1
Cuatro	35	5	0
Cinco	35	5	0
Seis	35	5	5
Siete	28	18	13
Ocho	20	1	1
Nueve	10	1	1

Una vez que se tienen los vectores representativos de cada identidad, se generan los del conjunto de prueba. Posteriormente se pasa por el algoritmo de comparación de cada método. En cada caso se indicará el procedimiento que se aplique.

Ahora se tratará el comportamiento de los algoritmos PCA, k -NN y SVM sobre el conjunto de imágenes de la Tabla 4.17

4.6.2. PCA

El programa hecho para este caso lee las colecciones de datos de procesamiento y de prueba. En ambos casos se generan valores en memoria que son pasados a la biblioteca `scikit-learn`. Se utiliza `OpenCV` para leer las imágenes y producir arreglos compatibles con las funciones de `scikit-learn`. El método `model_selection` de `scikit-learn` se encarga de separar las imágenes en un conjunto de procesamiento y otro de prueba, pero en este caso se llevó al límite inferior el conjunto de prueba debido a que ya se cuenta con uno seleccionado a mano, a fin de no usar las mismas imágenes de enrolamiento o captura, porque todas presentan cierta uniformidad y no es deseable que haya una sobrerepresentación.

La función `PCA(n_components=k, whiten=True)` instancia un objeto usado para generar los valores deseados. Cabe recordar que PCA busca los componentes principales del conjunto de datos que mejor extraen la varianza de múltiples vectores ortogonales entre sí. Con los resultados de esta primera pasada sin parámetros, se extraen los componentes cuya varianza no exceda 95%. Con ello se calcula un valor k que indica el número de componentes a usar en una segunda pasada. En nuestro caso probamos para varianzas de 94 a 97% y se eligió 95%, lo cual generó un número de 22 componentes principales. La segunda pasada se llamó con esta línea de código:

```
pca = PCA(n_components=k, whiten=True)
```

El parámetro `n_components` es propiamente k , y el parámetro `whiten` se usa para que los vectores del elemento `n_components` sean multiplicados por la raíz cuadrada de `n_samples` (número de imágenes) y luego dividida entre los valores singulares

(eigenvalores) para asegurar resultados no correlacionados con las varianzas de los componentes unitarios [59]. Posteriormente se aplica una transformada de descomposición¹² [62] a los datos de entrada mediante:

```
transformada_x = pca.fit_transform(x_entrada)
```

donde `x_entrada` es el tensor con los datos de procesamiento, pero que desde un inicio se aparta aleatoriamente del total de imágenes de la colección.



Figura 4.53: Imágenes generadas con la transformada inversa de los datos originales.

Esto no es estrictamente necesario para el reconocimiento, pero sirve para ilustrar la buena ‘representatividad’ de los vectores calculados. En la Figura 4.53 se muestra el resultado de aplicar la transformada inversa a los resultados en la variable `transformada_x` mediante:

```
x_representativos = pca.inverse_transform(transformada_x)
```

También es posible mostrar las imágenes que corresponden a los componentes principales de más alto valor en el caso de este estudio, como se ilustra en la Figura 4.54.

Debido a que PCA no es en sí una herramienta de cálculo de cercanía entre sus valores, se utilizó k -NN para calcular las distancias entre los vectores de entrada y los de prueba mediante una simple ecuación de distancia:

```
distancia = ((x_entrada[i, :]-x1)**2).sum()
```

donde `x_entrada` ya es conocido, `x1` es el i ésimo elemento del conjunto de prueba, y `distancia` es una lista de valores de la cual se toma el más común (cabe recordar que k -NN es como un sistema de votación).

El número de componentes utilizado es 3, y el resultado analítico de la función `accuracy_score` de `scikit-learn` fue:

¹²Internamente hace una descomposición de valor singular para modelar los datos de procesamiento y estimar los datos no vistos en una sola función.

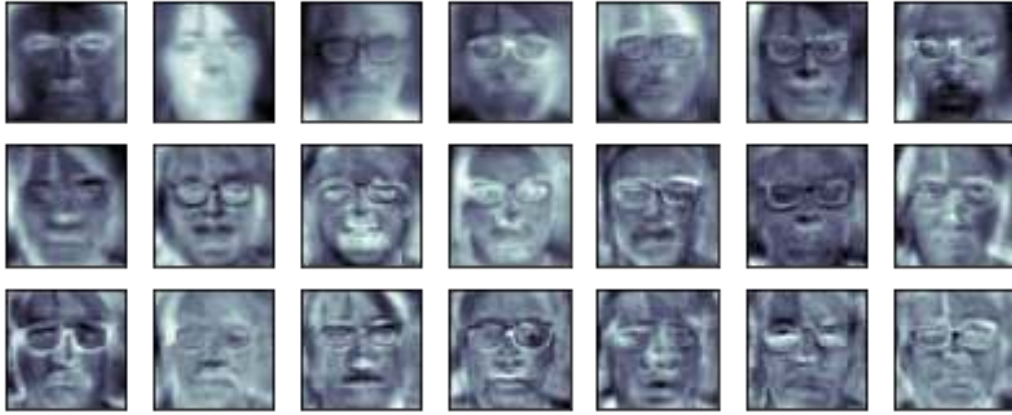


Figura 4.54: 21 de las 22 imágenes generadas a partir de los 22 eigenvectores con 95% de varianza.

(*'Accuracy score is'*, 0.49230769230769234)

A continuación se presentan los resultados visualmente más elocuentes. En la Figura 4.55 se puede observar que 20 de las 69 imágenes corresponden a una extracción aleatoria de 5 imágenes originalmente ‘enroladas’. Se introdujeron nuevas imágenes de prueba porque se observó una altísima precisión con imágenes del conjunto original. El texto de color verde representa un acierto, en tanto que el de color rojo representa error en la estimación.



Figura 4.55: Resultados de la evaluación del algoritmo PCA, con $k = 3$.

4.6.3. k -NN

k -NN es un algoritmo que toma prácticamente cualquier vector de características y compara la distancia que lo separa contra el resto de los vectores en el conjunto de entrada. Como se vio en PCA, las imágenes son pasadas por un proceso de cálculo de componentes principales, y esto deja un conjunto de datos más ligero. Por ejemplo, los datos de entrada de este trabajo son matrices de 100 x 100 píxeles que, al pasar por PCA dejan vectores de 22 dimensiones. Es posible aplicar cualquier algoritmo de reducción de dimensiones, pero en este caso se decidió tomar las matrices tal como vienen y hacer la evaluación en ese espacio de 100 x 100 dimensiones. El procesamiento es más tardado pero también se pueden usar imágenes de 28 x 28 píxeles, por ejemplo.

Igual que en PCA, se leen las colecciones de datos de entrada y de prueba, los valores son normalizados al rango 0.0 - 1.0, y se almacenan en variables en formato de arreglos NumPy.

Luego las matrices se aplanan y se pasan al algoritmo de comparación de distancias como en PCA, y los resultados se guardan en variables. Finalmente estas variables se utilizan para calcular la precisión del algoritmo y la representación visual de cada imagen de prueba.

Con este método la precisión calculada con `scikit-learn` fue:

```
('Accuracy score is', 0.49230769230769234).
```



Figura 4.56: Resultados del algoritmo k -NN, con $k = 3$.

En la Figura 4.56 se observa que cada imagen se presenta con un texto que indica el valor calculado y el valor real. Cuando estos valores coinciden, el texto se presenta en color verde. En caso contrario, se presenta en color rojo. También se puede ver que el algoritmo tiende a calcular más aciertos en imágenes extraídas de la colección original que con imágenes nuevas.

4.6.4. SVM

El programa empleado en este caso es muy diferente a los dos anteriores debido a que, por un lado, se utilizan funciones propias de OpenCV más que de `scikit-learn`, debido a que ofrecen mayor flexibilidad en el manejo de los datos de entrada y prueba.

Se utilizan las mismas imágenes que en los dos casos anteriores pero ahora no se hace una partición de datos de ejecución y datos de prueba como en el caso de `model_selection` de PCA, sino que se ocupa la totalidad de datos de entrada. Los datos de prueba son las mismas 65 imágenes originales para efectos de tener la misma base de comparación.

Se importa el modelo `openface_nn4_small12.v1.t7`, que es una versión de la red neuronal utilizada en [2] y hecha pública para su estudio. Aquí se toma cada una de las imágenes de entrada y se pasa por la red para generar un vector de 128 dimensiones. Cada vector es apilado junto con la clase (identidad) a la que corresponde. Cuando se termina el ciclo, cada representación y su identidad es almacenada secuencialmente en un archivo de formato *Pickle*. De forma resumida, un archivo *Pickle* es el ‘volcado’ de un objeto de memoria en un archivo de disco, de forma que al leerlo, las variables recuperan su estado original y no es necesario volver a generar los datos. A este archivo se le denominará ‘conjunto representativo de datos de entrada’.

Primero se realizó un proceso de generación de las representaciones y su almacenamiento en disco. Posteriormente, uno por uno, se fue generando el vector de características de cada imagen y se comparó contra el conjunto representativo de datos de entrada. OpenCV tiene una función que genera un objeto llamado ‘blob’, similar al ‘tensor’ de TensorFlow que es presentado a la red mediante la función `SetInput`. Posteriormente se pasa el ‘blob’ por la red y se obtiene el vector. El vector pasa por la función `predict_proba`, la cual entrega un arreglo de probabilidades de pertenencia a cada una de las clases en el conjunto representativo de entrada (en nuestro caso, nueve clases). Finalmente, la función `np.argmax` de la biblioteca NumPy, nos entrega la identidad con mayor probabilidad. Se recopiló la probabilidad de cada imagen y se calculó la precisión con la misma función de las dos secciones anteriores. En este caso el resultado fue:

```
('Accuracy score is', 0.7384615384615385)
```

En la Figura 4.57 se presentan los resultados del algoritmo SVM sobre la colección de datos de prueba.

Se ha visto que de los tres algoritmos de clasificación los mejores resultados se obtuvieron con SVM. De manera inesperada se encontró que tanto con los datos de PCA



Figura 4.57: Resultados del algoritmo SVM.

(vector de 22 dimensiones) y de k -NN con vector de 10 mil dimensiones, los resultados fueron idénticos para $k = 1, 2, 3, 5$ y 7 . En la Tabla 4.18 se resumen los resultados individuales mostrados arriba para cada algoritmo, con $k = 3$, en los casos de PCA y k -NN.

Tabla 4.18: 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)

Es posible afirmar que PCA es una buena herramienta de visualización, ya que al reducir dimensiones sin perder representatividad, permite reducir ampliamente el espacio muestral, representarlo y procesarlo. De k -NN se puede decir que es un método estadístico que utiliza distancia euclidiana para determinar la pertenencia de un objeto a una clase y que, aparentemente es independiente de las dimensiones del vector representativo que procese. Es importante que todo objeto tenga una clase que lo represente en el conjunto de datos de procesamiento, de otra forma se estima como 'desconocido'. Finalmente, SVM ha mostrado que es más tolerante a cambios drásticos en los datos de prueba. Es posible suponer que la generación de vectores representativos mediante la red neuronal juega un papel importante.

Capítulo 5

Evaluación de los resultados

5.1. Introducción

En el presente trabajo con el título 'Desarrollo de aplicaciones de reconocimiento facial con base en los resultados del análisis de tres familias de algoritmos' se establecieron al principio los objetivos específicos y se enumeró una lista de problemas a resolver. En esta sección se irán desarrollando los resultados en el mismo orden de las secciones anteriores.

Una parte fundamental de este trabajo consistió en reunir una colección de imágenes 'propia', en el sentido de que no se usó ninguna colección pública disponible para su estudio académico, debido a que la mayoría de aquellas se encuentra ampliamente documentada y tiene mecanismos de acceso que mucho simplifican su empleo. Una consulta en Internet sobre cualquiera de los 10 métodos aquí expuestos conduce a una colección de soluciones repetitivas que parecen copias una de otra. La intención del presente estudio no era repetir el trabajo de otros, como queda demostrado en las colecciones creadas y en la multitud de programas codificados por cuenta propia. Todo esto queda disponible para quien lo solicite, como forma de verificar lo aquí expuesto.

Una de las fuentes de este trabajo ha sido `tensorflow.org`. En este sitio se presentan ejemplos originales de gente destacada en el ramo y cuya solución es reproducida en múltiples sitios con algún cambio cosmetológico. No se encontró una referencia completa a la implementación desde cero con una colección propia, y por ello se prefirió hacerlo directamente para efectos de este trabajo. Definitivamente no es el caso copiar y pegar. Lo que se buscaba era un punto de partida para empezar. Por ello se hicieron varios programas para 'bajar' imágenes y revistas. Las revistas fueron 'deshojadas' y expuestas como imágenes. A todas las imágenes se les aplicó el método de Cascada de clasificadores débiles para detectar y recortar las caras. Las caras se repartieron en diferentes conjuntos y algunas se utilizaron para evaluar la detección, otras para evaluar funciones de pérdida, y otras más para evaluar clasificación.

Los diferentes métodos de cada categoría se pueden combinar para lograr el resultado fundamental del reconocimiento facial, pero si hay que apegarse a criterios de calidad, tiempo, costo y eficiencia, son preferibles los métodos de la Tabla 5.1.

5.2. Método propuesto

A continuación se presenta la Figura 5.1, misma que muestra un diagrama de bloques de un método que se le propone a quien se interese en la tarea del reconocimiento facial. Más abajo se describe cada bloque.

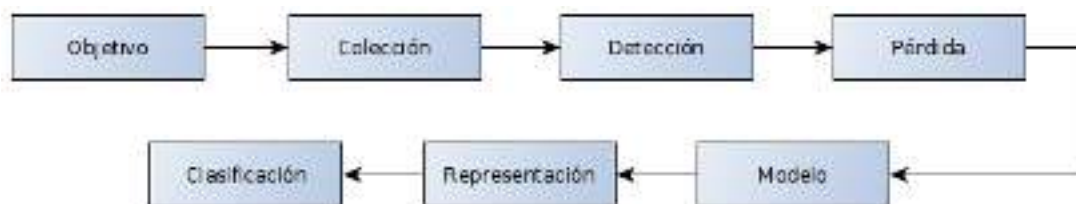


Figura 5.1: Diagrama de bloques del método propuesto para la tarea de reconocimiento facial.

- 1) Establecer claramente el objetivo. Puede ser permitir el acceso a un empleado o impedirlo a un intruso. Puede ser para pintarle bigotes a una cara en una imagen, para estimar la apariencia actual de un niño extraviado hace años o, para estimar el parentesco entre personas a partir de una colección de imágenes. También se puede modificar para dar alimento diferenciado a las mascotas. También, con modificaciones un poco más elaboradas, el objetivo puede enfocarse en realidad aumentada de carácter industrial. Cada caso es especial.
- 2) Seleccionar la colección de imágenes o recopilar la propia. Para darle alimento diferenciado a mascotas, necesitamos una colección propia. Para encimar el rostro de alguna persona pública en el rostro de otra, se pueden recabar imágenes de esas personas en Internet.
- 3) Seleccionar el algoritmo de detección a emplear o diseñar el propio. Hay muchos trabajos muy robustos de detección, pero si tiene una idea que probar, se cuenta con todo lo necesario para llevarla a la práctica. Los detectores que se emplearon aquí están ampliamente documentados y son un excelente punto de partida. La velocidad y la portabilidad son factores a considerar. Además, hay que aprovechar lo que en este trabajo ha quedado expuesto.
- 4) Seleccionar la función de pérdida a emplear con el objetivo de obtener una buena convergencia de la red, entendiendo la convergencia como una en la que la pérdida sea la mínima, aunque no cero, y la ganancia sea máxima, aunque no uno. Las investigaciones demuestran que no hay límite ni en el diseño ni en la implementación de una función de pérdida, aunque éste no sea un paso fácil. Si bien hay otros factores, éstos se pueden 'sintonizar' en el proceso.
- 5) Seleccionar la red neuronal que mejor procese los datos con que se trabaja. Puede tomarse una ya existente, modificarla o diseñar una desde cero. Tampoco

hay límites al diseño o la implementación. Sólo hay que tratar de que el modelo represente correctamente el problema que se trata de resolver. Se puede observar cierta tendencia a tomar las redes que ya demostraron excelente desempeño y hacerles modificaciones o hacerlas parte de otras más complejas.

- 6) De la mano con el diseño de la función de pérdida y el modelo de red, está el diseño de la representación vectorial y el hiperespacio de destino. Por ejemplo, es posible tomar la representación tal cual o reducir sus dimensiones. Como ocurrió en el presente estudio, pasar de imágenes de 180 x 180 píxeles a 28 x 28 píxeles, bajó drásticamente el tiempo de procesamiento y permitió utilizar equipo de cómputo de bajas prestaciones.
- 7) Seleccionar un algoritmo de clasificación que mejor agrupe las clases o hacer el propio. En el caso de reconocimiento facial, llevado al extremo de identificar correctamente a cada ciudadano, como en el caso de China, se requieren métodos no estudiados en la academia, pero que allí están, latentes, y sólo se trata de descubrir la mejor forma de representar numéricamente una cara y luego identificarla.

De acuerdo a los resultados, la mejor solución deberá emplear los algoritmos de la Tabla 5.1. Se agregan comentarios adicionales después de la tabla.

Tabla 5.1: Algoritmos que se sugiere emplear en la tarea de reconocimiento facial.

Categoría	Método	Observaciones
Detección	Cascada de clasificadores	Rápido, eficiente, controlable
Función de pérdida	Entropía cruzada	Mejor convergencia
Clasificación	Máquinas de soporte vectorial	La mejor precisión obtenida en este trabajo

En el caso de la detección de rostros, se observó que el más rápido y eficiente fue el de Cascada de clasificadores débiles. Aunque tiene más detecciones incorrectas, éstas se pueden limitar en una aplicación de control de acceso debido a que se tiene la decisión sobre el lugar de captura del flujo de video: es un lugar fijo, en interiores, con iluminación homogénea (sin sombras ni reflejos), con un recuadro de captura fijo y condiciones internas de alineación de ojos. Por el lado de los otros tres mecanismos, fue posible notar que el redimensionamiento de las imágenes necesarias para aumentar la eficiencia consume mucho tiempo de procesamiento y que se hace prohibitivo en cierto tipo de implementaciones, como el de las aplicaciones móviles y en microcontroladores de bajo costo.

En lo que respecta a funciones de pérdida, se observó que no cualquiera es adecuada para el procesamiento de imágenes. Aunque se puede pensar que la distancia euclidiana aplica en cualquier hiperespacio, también debe considerarse el número de clases a manejar. Si hay más de dos clases, la clasificación con función de pérdida binaria (binary cross entropy), así como la función de activación sigmoide, no son

adecuadas. Para estos casos es más adecuada una función `sparse_categorical_crossentropy` y una función de activación `softmax`. Los datos obtenidos durante el desarrollo de esta sección así lo dejan ver. La mejor función de pérdida para el conjunto de datos seleccionado, fue la de Entropía cruzada categórica dispersa.

En cuanto a la clasificación, se analizaron los tres esquemas propuestos y se apreció que PCA es una técnica de reducción de dimensiones que en sí no contiene un mecanismo de clasificación. Por eso se utilizó otro mecanismo que sí lo permite. En este caso, k -NN. Posteriormente se analizó k -NN pero sin un proceso previo de los datos crudos. Se pasó el arreglo de 10 mil elementos por imagen al algoritmo y se encontró que para el mismo conjunto de datos, se obtiene el mismo resultado que en PCA para valores de $k = 1, 3, 5$, y 7 . Finalmente se analizó SVM y aquí se logró el mejor resultado. Por un lado, el algoritmo es en sí un algoritmo de clasificación, y por otro, el análisis de un vector de características de 128 dimensiones lo hacen un método idóneo para aplicaciones de aceptable precisión y respuesta tiempo real.

5.3. Omisiones

En la evaluación de algoritmos de detección facial faltó considerar si el tiempo de procesamiento era lineal o no lineal. Los resultados obtenidos en las diferentes pruebas conducen a pensar que es lineal pero no fue comprobado.

En el caso de la evaluación de funciones de pérdida, no se trabajó en más de dos clases debido a la dificultad de reunir suficientes imágenes de la misma persona. Fue más viable reunir más de 5,200 imágenes por clase considerando únicamente mujeres y hombres.

En cuanto a los algoritmos de clasificación no se tenía noción de que se obtendrían iguales resultados en PCA y k -NN. Fue algo que se encontró luego de escribir este documento y ya se estaba *codificando*¹ la presentación para el examen a puerta cerrada. No obstante fue posible hacer pruebas que constataron lo que ya se reportó.

5.4. Restricciones

El presente estudio tiene restricciones de reproducibilidad debido a los cambios frecuentes en las funciones empleadas. Aunque en lo fundamental la teoría no cambia, sí lo hacen las herramientas en las que se implementan tales teorías. De esta forma sucede que una biblioteca como `scikit-learn` tiene funciones de redes neuronales, pero también las tiene `pyTorch` o `TensorFlow`, y los formatos internos de cada una no son compatibles entre sí. Un caso que podría considerarse ‘severo’ es el uso

¹Recordemos que `LATEX` es un lenguaje de programación para el formateo de documentos.

de la biblioteca `gast`², que cuando se usa la versión 0.33 con `Tensorflow 2.3`, produce ganancias de 0.65%, mientras que usando la versión 0.4 con `TensorFlow 2.4`, obtiene 0.93% para exactamente el mismo conjunto de parámetros, según se lista en la Tabla 5.2.

Tabla 5.2: Variación de resultados al usar diferentes versiones de bibliotecas.

Concepto	Biblioteca	
Versión	gast 0.33 @ tf 2.3	gast 0.4 @ tf 2.4
Ganancia (%)	0.65	0.93
Función de pérdida	SparseCategoricalCrossentropy	
Tasa de convergencia	Adam 0.001	
Modelo de red	Convolutacional de 18 capas	
Épocas	15	

5.5. Mejora de los algoritmos o de su práctica

Para la detección se encontró la mejor combinación de parámetros de la función de detección, y que no son necesarios algunos pasos de redimensionamiento propuestos en otras fuentes.

Para las funciones de pérdida se encontró que reduciendo las dimensiones de entrada se mejora el desempeño y se reduce dramáticamente el tiempo de procesamiento. También se encontró que un monitoreo del proceso, y la identificación de oscilaciones en los valores de ganancia y pérdida, su estancamiento o disparo, son señales de que el proceso debe detenerse e intentar otra aproximación.

Para el proceso de clasificación se encontró que el enrolamiento incremental disminuye el tiempo de proceso; que la captura de imágenes cuando los ojos están alineados y se presenta un recuadro de confinamiento de la cara durante el enrolamiento, permiten una mejor agrupación de identidades.

El uso del recuadro que marca la cara detectada es ventajoso por las siguientes razones: 1. Se omite inspeccionar el resto de la cabeza y el fondo. 2. Disminuye el tiempo de procesamiento en cualquiera de los 10 algoritmos. 3. En el caso de representaciones vectoriales, éstas se remiten a un área con datos relevantes y evita distractores tales como oclusiones parciales, sombreros y objetos en el fondo.

²Esta biblioteca es necesaria para la optimización del árbol sintáctico (grafo) de la red y es llamada por `TensorFlow`.

5.6. Ejemplo de aplicaciones que se pueden desarrollar

La Figura 5.2 muestra la interfaz gráfica de la aplicación Amaxayac y la interfaz de línea de comando de un programa que establece un ciclo permanente de detección-búsqueda-reconocimiento para decidir si se permite el acceso o no. En cualquier caso, se registra una incidencia. La interfaz gráfica, como se indica a la derecha de cada botón, permite llevar a cabo el ciclo completo de operaciones de sistema de control de acceso. En éste se registran los datos generales y se capturan y editan imágenes de cada identidad. Posteriormente se generan los vectores representativos, se pasan por una red de clasificación y se registran en una base de datos contra la cual se compara el nuevo vector que se genere luego de una detección facial. De manera automática, el programa permite o no el acceso y registra la incidencia. Se dejó el botón para funcionar en modo manual.

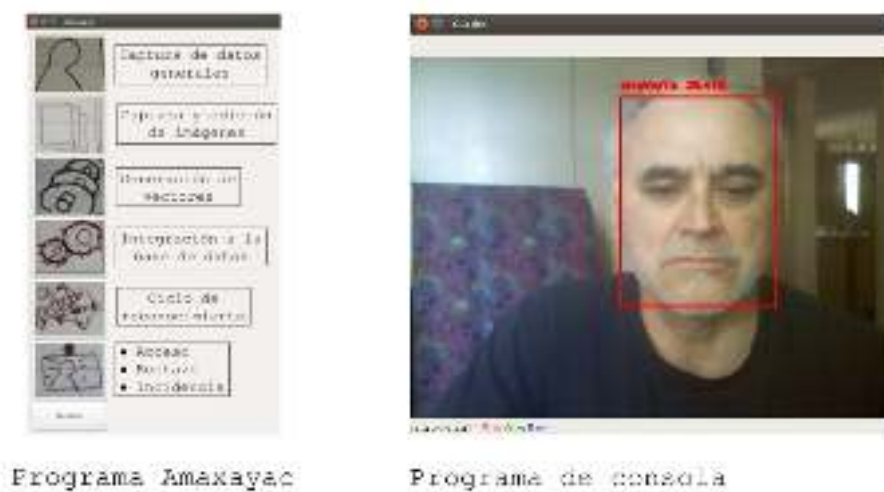


Figura 5.2: Ejemplos de aplicaciones logradas en este trabajo.

5.7. Comparación con otras soluciones

A continuación se presentan dos comparaciones. La primera es respecto a una solución de índole académica, y otra en el terreno comercial. Ambas tienen distinta complejidad y alcance y por eso se considera conveniente incluirlas.

5.7.1. Solución académica

La primera comparación es con una implementación hecha en la Universidad Carnegie Mellon. La siguiente es una consulta en el grupo *CMU-OpenFace* de la pla-

taforma *Google Groups* que muestra una solución doméstica basada en OpenFace, al igual que en el presente estudio. El texto se tradujo del inglés y sólo se transcribe la pregunta en el idioma original.

Pregunta: Quiero construir e instalar un sistema de asistencia (laboral) basado en reconocimiento facial para escuelas. Por favor díganme cuáles son los requerimientos y, de ser posible el costo del proyecto.

On Wednesday, April 24, 2019 at 9:30:26 AM UTC-7, binnachri...@gmail.com wrote:

I want to build and install the face recognition attendance system for schools, please what are the requirements, and possibly the cost of the project.

Respuesta de Carnegie Mellon University: Si aún no tienes una aplicación que maneje empleados, turnos, descansos, historial, reportes, etc. tendrás que conseguir uno o programarlo. Es probable que requieras un servidor o PC para ejecutarlo. Luego necesitarás terminales con cámara y algún tipo de capacidad de retroalimentación (una pantalla) y acceso de datos; tantas como lugares por los que chequen los empleados. Como ejemplo, tenemos un sistema ERP^a, para implementar la recopilación de datos de entradas y salidas y tiempo laborado. Configuré un servidor dedicado con OpenFace. Modifiqué nuestro ERP para permitir transferencias entre éste y las terminales checadoras. Para las terminales diseñé una interfaz táctil (pantalla táctil de 10”) usando Python en una RPi (Raspberry Pi) en un gabinete a la medida. El empleado toca el reloj en forma de botón en la terminal. La terminal toma una corta serie de imágenes y las envía al servidor OpenFace. Este las procesa, e identifica al empleado (y su número de empleado) y lo envía de regreso a la terminal. La terminal envía entonces el Id del empleado y la hora al servidor ERP. Un programa en el servidor ERP acepta o rechaza el chequeo (en base a varios controles) y envía los resultados de regreso a la terminal que muestra la información correspondiente en la pantalla. Las terminales y el servidor están programados para permitir agregar caras, así como para desplegar información de accesos del empleado durante la semana. ¿Costos? Depende de tu experiencia en hardware, programación y diseño de sistemas, así como del número de terminales. Nuestras terminales cuestan como \$550 dólares en puras partes. Si las hiciéramos con pantallas de 7”, el costo caería significativamente como a \$250 dólares. El servidor OpenFace que estamos usando es un sistema dedicado y costó como mil dólares. Este servidor debe ser capaz de soportar más de 50 terminales. Hay mucha programación en medio. En realidad la parte de OpenFace es mínima. La mayor parte del código es con las terminales y la interfaz en pantalla. La programación de la interfaz con el ERP puede ser o simple o muy complicada. Si tienes que programar tu propia aplicación y base de datos para entradas y salidas, el tiempo de desarrollo puede ser bastante extenso. Dicho lo anterior, puede hacerse. Toma un rato extenso entender de sistemas, hardware y programación para arrancar. Si tienes esas habilidades o tienes ayuda en las áreas en las que no, éste sería un proyecto interesante.

Fin de la conversación.

^aSiglas en inglés que se refieren a un ‘Sistema de planificación de recursos empresariales.’

A continuación, en la Tabla 5.3, se muestran algunos rubros destacados de la solución académica y el presente trabajo.

Tabla 5.3: Comparación entre dos soluciones de índole doméstico.

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

La Tabla 5.3 dice que un servidor dedicado puede no ser necesario si no se tiene un ERP. El reconocimiento en la terminal es una ventaja, pues no requiere interactuar con el servidor excepto para enviar la incidencia. Por último, la facilidad de decidir cuáles imágenes son adecuadas para el enrolamiento es una característica que mejora el diseño y el desempeño.

5.7.2. Solución comercial

Ahora se presenta la Tabla 5.4, que compara rubros de una implementación comercial típica y el enfoque del presente estudio.

Tabla 5.4: Comparación entre una solución comercial y la aquí propuesta.

Concepto	Comercial	Amaxayac
Número de caras	10 mil	No evaluado
Tiempo de respuesta	No más de 5s en positivo verdadero	De instantáneo a un segundo para estimar frecuencia.
Falsos positivos	No reporta. No permite el acceso.	4 en promedio, calcula mayor frecuencia.
Falsos negativos	Frecuentes con oclusiones diversas.	No reporta falsos negativos
Antifalsificación	Sí	No
Regionalización	Muchos tipos de cara	Muchos tipos de cara
Tecnología	No indica	Red neuronal.
Base de datos	MySQL. Reportes excel. No directo.	MySQL para incidencias , archivos binarios para identidades.
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

Aunque la solución propuesta en este trabajo dista mucho de una implementación comercial como la que aquí se muestra, principalmente debido a la gran cantidad de identidades posibles, se cree que en entornos de hasta 500 identidades podría tener un desempeño aceptable. Algo que los sistemas comerciales no muestran es la multiplicidad de identidades a las que puede pertenecer la cara bajo reconocimiento. Tampoco permiten el reporte de falsos positivos, pues es mejor no dejar pasar a alguien autorizado que dejar pasar, por error atribuible al sistema, a alguien no autorizado.

Una forma de mostrar la conveniencia de la elaboración propia de soluciones se ilustra con la Figura 5.3, en la que se observa una aplicación *Web* para consultar una tabla de incidencias originalmente entregada en formato de Excel. Luego de un re-trabajo intenso (e innecesario en otro caso) se logra observar un tablero informativo que dice más que los miles de renglones de los archivos excel. El re-trabajo es un paso costoso, ya que teniendo acceso libre a los datos crudos, es posible emitir cualquier tipo de reporte bajo demanda con sólo pulsar un botón y con la velocidad de respuesta típica de las aplicaciones *Web*.

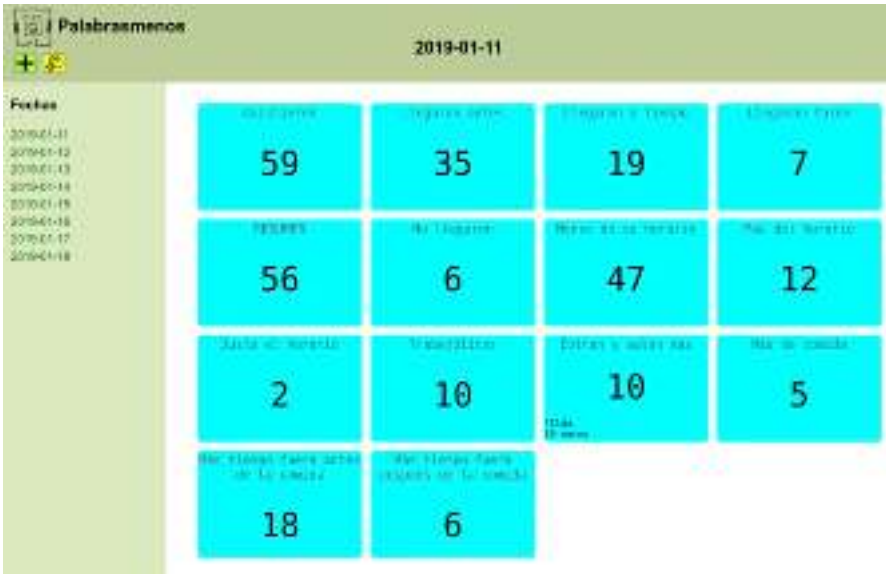


Figura 5.3: Tablero de incidencias durante el día.

5.8. Evaluación global

Como se pudo ver, se encontraron los mejores algoritmos de cada objetivo haciendo una evaluación en términos de eficiencia, velocidad de respuesta, dimensiones y plataforma de destino de la aplicación. A continuación se presenta la Tabla 5.5, la cual resume lo que se logró al trabajar en los Objetivos específicos de esta tesis. Los recuadros de color pistache indican con qué algoritmo se cumplió el objetivo.

Tabla 5.5: Resumen de resultados asociados a los Objetivos específicos.

Objetivo específico i)				
Encontrar el mejor detector facial en términos de eficiencia y velocidad, de los cuatro analizados aquí.				
Algoritmo	Comentario	Resultados *		
Haar	Función de OpenCV altamente configurable.	C	T (s)	V (i/s)
		1,931	47	51.63
HOG	Mejora marginal pero mayor tiempo.	C	T (s)	V (i/s)
		1,975	353	6.87
dlib	Mejor rendimiento si se usa GPU.	C	T (s)	V (i/s)
		2,376	1,408	1.72
OpenCV	Aunque es el segundo más rápido, no detecta tantas caras como el anterior.	C	T (s)	V (i/s)
		2,067	269	9.02
* En esta columna de resultados, el número total de imágenes procesadas fue siempre 2,427. Los encabezados son: C = número de caras detectadas, T = tiempo en segundos, y V = número de imágenes procesadas por segundo.				

Objetivo específico ii)				
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.				
Algoritmo	Comentario	Resultados		
Distancia euclidiana	Es muy rápida, fácil de implementar y viene en cualquier plataforma de redes neuronales.	Pérdida	Ganancia	
		0.4966	0.4968	
Entropía cruzada	Es muy rápida, fácil de implementar y viene en cualquier plataforma de redes neuronales, pero también fue la más efectiva en este trabajo.	Pérdida	Ganancia	
		0.2298	0.9031	
Pérdida por tripleta	No muy utilizada por la dificultad para formar ‘tripletas duras’ (identidades distintas muy parecidas o misma identidad poco parecida).	Pérdida	Ganancia	
		0.5273	0.4986	

Objetivo específico iii)				
Descubrir el mejor algoritmo de clasificación de representaciones vectoriales faciales de los tres analizados aquí.				
Algoritmo	Comentario	Resultados		
PCA	Grupo de técnicas estadísticas aplicadas a un conjunto de datos para determinar sus valores y vectores propios. El conjunto de datos reducido usa k-NN para su clasificación.	Precisión		
		0.49230769230769234		
k-NN	Mide la distancia euclidiana de un punto a clasificar respecto al resto de puntos. El nuevo punto pertenecerá a la clase con la que tenga k vecinos más cercanos. k siempre es un número impar para evitar empates en un rango típico de 1 a 7.	Precisión		
		0.49230769230769234		
SVM	Algoritmo especialmente diseñado para encontrar el hiperplano óptimo en conjuntos de datos linealmente separables.	Precisión		
		0.7384615384615385		

Tabla 5.3: Continuación y final.

Objetivo específico iv)			
Mostrar que se pueden hacer colecciones de imágenes y herramientas propias para estas investigaciones y para reproducir resultados de otros estudios.			
Resultados			
Concepto	Número de imágenes	Número de clases	Número aproximado de programas
Detección	2,427	10	22
Pérdida	11,451	2	10
Clasificación	622	9	10
Colecciones	14,530	21	19

5.9. Problemas resueltos

La tabla siguiente muestra el problema planteado y la solución encontrada.

Tabla 5.6: Solución a los problemas planteados en la Sección 1.2

#	Problema	Solución
1	Los sistemas de control de acceso y videovigilancia son cerrados.	Aquí se hace un sistema abierto de principio a fin y se muestra información para que el investigador decida cómo hacerlo.
2	No hay forma de acceder directamente a sus registros.	Se muestra un tablero informativo (Fig. 5.3) como ejemplo de lo que se puede lograr.
3	No hay forma de agregar funcionalidad	El investigador decide qué agregar.
4	No son transparentes en sus resultados (por ejemplo, razones por las que se niega el acceso a alguien debidamente registrado).	El usuario puede saber si se le está confundiendo con otra identidad y el investigador puede corregir el problema.
5	No ofrecen un canal de comunicación directo con el usuario.	Se establece la comunicación faltante.
6	La información relevante para reproducir un sistema de control de acceso desde cero es inaccesible o inexistente.	Aquí se expone todo lo necesario.
7	Hay información que no ha sido incluida en la documentación de funciones relevantes en esta materia.	A lo largo del trabajo se han reportado resultados no documentados de casi todas ellas.
8	No se conocen detalles sobre la implementación tales como dimensiones de una aplicación, plataformas de despliegue, lenguajes de programación, bases de datos, colecciones de imágenes, velocidad de respuesta, precisión, complejidad y otros factores).	A lo largo del trabajo se han hecho propuestas. En la Figura 5.1, hay un diagrama de ejemplo.
9	No es claro que haya otras aplicaciones en campos tales como: entretenimiento, sistemas de recomendación personalizados y realidad aumentada.	También se ha incluido información al respecto a lo largo del trabajo.

Con base en lo anteriormente expuesto, es posible asegurar que la Hipótesis de trabajo ha sido satisfecha.

Capítulo 6

Conclusiones

6.1. Introducción

Este ha sido un trabajo extenso en el que se aprendió una variedad de temas que originalmente no se habían considerado, pero que sobre la marcha fue necesario adquirir para poder presentar resultados convincentes y verificables. Es importante reconocer y agradecer a esa gran red de investigadores y desarrolladores que generosamente abren al público sus hallazgos y herramientas. Entre ellos deben mencionarse principalmente a Amos [2], Sagonas [60], Schorff [35] y Kazemi [61]. También a los desarrolladores de Linux, Python, NumPy, OpenCV, dlib, scikit-learn, Matplotlib, Keras, Tensorflow, Jupyter, Anaconda, wxWidgets y L^AT_EX, entre otros muchos.

6.2. Conclusiones

En este trabajo se cumplió con la Hipótesis porque fue posible tener control sobre todo el entorno de análisis. En cuanto a los objetivos específicos, se logró satisfacer cada uno de ellos. Pudimos mostrar que las Cascadas de clasificadores débiles, es más ágil. Hay que recordar que el algoritmo con resultados más próximos pudo encontrar 44 caras adicionales, pero en un tiempo 7 veces mayor. Fue posible mostrar que la función de pérdida conocida como Entropía cruzada presentó un desempeño mayor a 90 % al trabajar con la red convolucional aquí expuesta. Cabe destacar que dicha red tiene un tamaño compacto, y que sólo fueron necesarios 15 ciclos de procesamiento, de forma que en cuestión de minutos puede procesar 14,530 imágenes repartidas en dos clases con una computadora rudimentaria. En cuanto a la mejor técnica de clasificación de rostros, se encontró que SVM se desempeña mejor que las otras dos técnicas analizadas. En la sección de Evaluación global se puede consultar la Tabla 5.5, misma que resume los resultados de los Objetivos específicos, y la Tabla 5.6, con la lista de problemas resueltos.

Gracias al trabajo de recopilación de imágenes provenientes de Internet, revistas en formato .pdf y capturadas de un flujo de video, se logró filtrar decenas de miles de caras contenidas en ellas para terminar con un conjunto de 14,530 imágenes con la cara

de una persona.

Este trabajo mostró que se puede detectar, codificar (crear representaciones vectoriales) y clasificar (por lo tanto, reconocer) caras en tiempo real, utilizando exclusivamente herramientas de código abierto, con lo que se tiene acceso a la máxima flexibilidad en el desarrollo del cualquier estudio.

6.3. Recomendaciones

Se recomienda a los lectores de este trabajo que quieran adentrarse en el tema de reconocimiento facial, que establezcan objetivos muy precisos y que siempre usen las herramientas de última liberación más estable, debido a que en el transcurso de una investigación éstas pueden variar y algunas funciones pueden retirarse por obsolescencia. No siempre hay un documento que diga cuál función desapareció y cuál la sustituyó. No es recomendable seguir ningún ejemplo que use versiones de TensorFlow anteriores a la 2.3, debido a que su ejecución debe hacerse con las mismas versiones de los ejemplos. Se encontraron mejoras importantes en versiones más recientes, y por ello recomendamos que se inicie cualquier estudio a partir de aquellas. Eso evitará perder el tiempo con versiones obsoletas. Como alternativa, se recomienda practicar con PyTorch, que a últimas fechas es un muy cercano competidor de TensorFlow.

Se recomienda [12], que es un congreso que recibe y publica miles de artículos que se pueden consultar de manera gratuita, a diferencia de los de IEEE, en los que cada artículo cuesta sobre 30 dólares

6.4. Trabajo futuro

No es la intención continuar el estudio de reconocimiento facial debido a que ya fue posible crear un sistema propio (Amaxayac, citado en el glosario y en la sección de clasificación mediante SVM), pero sí del proceso de imágenes en general.

La meta inmediata que se tiene, es iniciar una investigación sobre navegación aérea metropolitana de aeronaves no tripuladas, utilizando visión por computadora mediante la o las cámaras a bordo, entre otros de los sensores de esos equipos, e integrarla a soluciones conocidas como ‘Ciudades inteligentes’.

Los lectores de este trabajo podrán notar que en las características no documentadas de varias funciones se puede encontrar una oportunidad de estudio. Por el lado de creación de colecciones de imágenes propias, también se abre un camino amplio, ya que no abundan colecciones en las que la etnicidad sea considerada un factor clave ni en su creación ni en su análisis. Aquí también se enumeraron algunos trabajos que se están investigando en congresos, tales como el de visión por computadora y reconocimiento de patrones, en los que se pueden encontrar muchísimas oportunidades de estudio. Finalmente, se puede investigar qué ocurre cuando se usa la función de pérdida por tripleta en colecciones de 500 o más identidades, con al menos 50 imágenes en diferentes condiciones por identidad. La mera recopilación de la colección de imágenes es, en sí, un reto. ■

Bibliografía

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [2] Brandon Amos, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science*, 6(2), 2016.
- [3] Vladimir N. Vapnik (auth.). *The Nature of Statistical Learning Theory*. Springer New York, 1st edition, 1995.
- [4] Intelligent behaviour understanding group. Facial Point Annotations. <https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/>, 2013. visitada: 2021-06-23.
- [5] Jonathan T. Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [6] W. W. Bledsoe. The model method in facial recognition. *Technical Report PRI 15, Panoramic Research, Inc.*, 1964. visitada: 2021-06-23.
- [7] Qiuyu Chen, Wei Zhang, Jun Yu, and Jianping Fan. Embedding complementary deep networks for image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [8] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.
- [9] Francois Chollet. Image classification from scratch. https://keras.io/examples/vision/image_classification_from_scratch/, 2020. visitada: 2021-06-23.
- [10] François Chollet et al. Keras: The python deep learning library. *Astrophysics Source Code Library*, pages ascl–1806, 2018. visitada: 2021-06-23.

- [11] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.
- [12] The computer vision foundation. Computer vision and pattern recognition conference. <https://www.thecvf.com/>, 2021. visitada: 2021-06-23.
- [13] Wikipedia contributors. Entropy (information theory)—Wikipedia the free encyclopedia. [https://en.wikipedia.org/wiki/Entropy_\(information_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory)), 2021. visitada: 2021-06-23.
- [14] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.
- [15] Franklin C Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212, 1984.
- [16] CV-Tricks. Running deep learning models in opencv. <https://cv-tricks.com/how-to/running-deep-learning-models-in-opencv/>, 2019. visitada: 2021-06-23.
- [17] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.
- [18] Equipo de stackexchange. Why normalize images by subtracting images mean... <https://stats.stackexchange.com/questions/211436/why-normalize-images-by-subtracting-datasets-image-mean-instead-of-the-current>, 2017. visitada: 2021-06-23.
- [19] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [20] Dlib.net. Face recognition in python. http://dlib.net/face_detector.py.html. visitada: 2021-06-23.
- [21] Evelyn Fix and Joseph L Hodges Jr. Discriminatory analysis-nonparametric discrimination: Small sample performance. Technical report, CALIFORNIA UNIV BERKELEY, 1952.
- [22] Guillermo Gallego, Mathias Gehrig, and Davide Scaramuzza. Focus is all you need: Loss functions for event-based vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [23] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Conference on computer vision and pattern recognition (CVPR)*, 2012.

- [24] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [25] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.
- [26] OpenCV group. Cascade classifier. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. visitada: 2021-06-23.
- [27] Isabelle Guyon, B Boser, and Vladimir Vapnik. Automatic capacity tuning of very large vc-dimension classifiers. In *Advances in neural information processing systems*, pages 147–155, 1993.
- [28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [29] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. *CoRR*, abs/1812.01187, 2018.
- [30] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.
- [31] Yuge Huang, Yuhan Wang, Ying Tai, Xiaoming Liu, Pengcheng Shen, Shaoxin Li, Jilin Li, and Feiyue Huang. Curricularface: Adaptive curriculum learning loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [32] The Tonight Show Starring JF. Wf and cs talk about their rivalry. <https://www.youtube.com/watch?v=ESWHyBOK2iQ>, 2014. visitada: 2021-06-23.
- [33] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.
- [34] Takeo Kanade. Picture processing system by computer complex and recognition of human faces. 1974.
- [35] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1867–1874, 2014.
- [36] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4873–4882, 2016.

- [37] Davis E. King. Cnn based face detector using dlib. http://dlib.net/cnn_face_detector.py.html. visitada: 2021-06-23.
- [38] Davis E King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009.
- [39] Davis E King. Max-margin object detection. *arXiv preprint arXiv:1502.00046*, 2015.
- [40] Davis E King. Dlib deep neural network module. http://dlib.net/dnn_mmod_ex.cpp.html, 2017.
- [41] Davis E King. Use dlib for face recognnition. http://dlib.net/face_recognition.py.html, 2017.
- [42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2017.
- [43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [44] G. Lugosi L, Devroye. L. Györfi. *A probabilistic Theory of Pattern Recognition*. Springer, 1991.
- [45] Yann LeCun and Yoshua Bengio. Pattern recognition and neural networks. *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, 1995.
- [46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [47] Jian Li, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Jilin Li, and Feiyue Huang. Dsfd: Dual shot face detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [48] Yann Lifchitz, Yannis Avrithis, Sylvaine Picard, and Andrei Bursuc. Dense classification and implanting for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [49] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [50] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. *CoRR*, abs/1704.08063, 2017.

- [51] Robert K. McConnell Arlington Mass. Method of and apparatus for pattern recognition. <https://patents.google.com/patent/US4567610A/en>, 1986. U. S. Patent 4567610.
- [52] Nikolaus Mayer, Eddy Ilg, Philip Häusser, and Philipp Fischer. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. <https://www.arxiv-vanity.com/papers/1512.02134/>, 2015. visitada: 2021-06-23.
- [53] Xiang Ming, Fangyun Wei, Ting Zhang, Dong Chen, and Fang Wen. Group sampling for scale invariant face detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [54] Mahyar Najibi, Bharat Singh, and Larry S. Davis. Fa-rpn: Floating region proposals for face detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [55] Andrdrew Ng. Deeplearning ai – ai for everyone. <https://www.deeplearning.ai/program/ai-for-everyone>, 2021. visitada: 2021-06-23.
- [56] OpenCV. Cascade classifier. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html, 2021. visitada: 2021-06-23.
- [57] OpenCV. Open computer vision. <https://opencv.org>, 2021. visitada: 2021-06-23.
- [58] Karl Pearson. Principal components analysis. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 6(2):559, 1901.
- [59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [60] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 397–403, 2013.
- [61] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. 2015.
- [62] Scikit-learn. 6. dataset transformations. https://scikit-learn.org/stable/data_transforms.html, 2013. visitada: 2021-06-23.
- [63] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- [64] Abita Sharma. Convolutional neural networks in python with keras. https://www.datacamp.com/community/tutorials/convolutional_neural_networks-python, 2017. visitada: 2021-06-23.

- [65] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [66] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.
- [67] TensorFlow. Introduction to graphs and tf.function. https://www.tensorflow.org/guide/intro_to_graphs, 2015. visitada: 2021-06-23.
- [68] TensorFlow. Tensorboard: TensorFlow’s visualization toolkit. <https://www.tensorflow.org/tensorboard>, 2015. visitada: 2021-06-23.
- [69] TensorFlow. Fashion-mnist with tf.keras. <https://blog.tensorflow.org/2018/04/fashion-mnist-with-tfkeras.html>, 2018. visitada: 2021-06-23.
- [70] Koutroumbas Konstantinos. Theodoridis, Sergios. *Pattern Recognition, Second Edition*. Elsevier, Academic Press, 2003.
- [71] S. Kotz V. Vapnik. *Estimation of Dependences Based on Empirical Data: Empirical Inference Science*. Information Science and Statistics. Springer, 2nd edition edition, 2006.
- [72] 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.
- [73] Ramsri Vlogs. Viola jones face detection and tracking explained. <https://www.youtube.com/watch?v=WfdYYNamHZ8>, 2013. visitada: 2021-06-23.
- [74] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 5265–5274. IEEE Computer Society, 2018.
- [75] Qiangchang Wang, Tianyi Wu, He Zheng, and Guodong Guo. Hierarchical pyramid diverse attention networks for face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [76] Patrick Henry Winston. Learning: Support Vector Machines, Ford Professor of Artificial Intelligence and Computer Science at the Massachusetts Institute of Technology. https://www.youtube.com/watch?v=_PwhiWxHK8o, 2010. visitada: 2021-06-23.
- [77] Zhi Zhang, Tong He, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of freebies for training object detection neural networks. *CoRR*, abs/1902.04103, 2019.

- [78] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1491–1498. IEEE, 2006.