Detección Facial en Vídeos Digitales

Facial Detection On Digital Videos

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE
CURSO 2019–2020

Arturo Barbero Pérez
Alejandro Cabezas Garríguez
José Morcuende Sierra

Directores

Luis Javier García Villalba
Esteban Alejandro Armas Vega

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2020

Agradecimientos

Arturo Barbero Pérez

Antes de comenzar, me gustaría parafrasear una expresión atribuida al filósofo Bernardo de Chartres, la cual dice así: «Somos como enanos sobre los hombros de gigantes, de modo que podemos ver las cosas a una mayor distancia. No en virtud de nuestra agudeza visual, o cualquier distinción física, sino porque nos elevan alto por su gran altura».

Para mí, aquellos *gigantes* a los que hace referencia, son todas las personas que, de una forma u otra, han influido en que hoy pueda ser como soy y, gracias a ello, me encuentre finalizando estos estudios. Por ello, me gustaría dedicar y agradecer el presente trabajo a los que yo considero mis *gigantes*:

A mis padres, por todo lo que me han dado durante toda la vida. Por haber estado al lado, tanto en los logros como, sobre todo, en los tropiezos. Por haberme transmitido desde siempre los valores del esfuerzo y la constancia y haber confiado en mí cuando yo no lo hacía.

A mi hermano, por todos los momentos que hemos pasado y por ser la persona que me dio a descubrir la informática desde pequeño, haciendo que me acabara interesando cada vez más por este mundo. Gracias a ello, ahora sé qué es lo que quiero hacer con mi vida.

A mis primos, en especial a Edu y Bea, por toda la ayuda y consejos que me han brindado a lo largo de los años. Porque, además de familia, son ejemplos y modelos a seguir tanto personal como profesionalmente. No podría estar más agradecido.

A David y Mariano, por estar prácticamente desde el principio de los tiempos y ser desde entonces un gran apoyo. Por todas las buenas experiencias que hemos pasado juntos y por haberme enseñado lo que es realmente el valor de la amistad y la lealtad.

A todos mis mastodontes, fieras y leyendas —ellos saben quiénes son—, por todas las risas, quedadas y el apoyo continuo en prácticas y exámenes. Gracias a ellos, ahora la frase «¡Feliz año!» tiene un significado aún más especial. Mi experiencia universitaria habría sido muy distinta sin vosotros.

A los profesores que he ido teniendo a lo largo de la carrera porque, además de haberme enseñado el campo de la Ingeniería Informática, me han hecho darme cuenta de que enfrentarse a problemas complejos por los que nunca antes has pasado es una de las mejores formas de crecer como persona.

Y por último, no podría olvidarme tanto de nuestros directores como de mis compañeros, así como de Ana Lucila y Alexandra. Gracias por darme la oportunidad de realizar este trabajo con todos vosotros y haberme acompañado en este reto. Estoy seguro de que, de no haber sido por vuestra ayuda, la realización de este proyecto se habría dificultado en gran medida.

Alejandro Cabezas Garríguez

En primer lugar, me gustaría agradecer a nuestros directores, Luis Javier y Esteban por habernos dado la oportunidad de investigar sobre estos campos que, personalmente, me han hecho amar aun más esta carrera y el mundo del software y la informática en general. Gracias por aceptar aquella reunión en mitad de verano y gracias por aceptarnos para realizar este proyecto. Además, creo que el trabajo que hemos realizado conjuntamente con Ana ha sido muy satisfactorio y productivo, y le agradezco a esta persona la labor de organización que ha tenido con nosotros a lo largo del desarrollo de todo el trabajo. También debo agradecer a Alexandra por la labor que realizó a principio de curso para enseñarnos los conceptos básicos con los que poder comenzar a investigar y a aprender por nosotros mismos. Sin duda alguna no habríamos podido comenzar con tanta soltura si ella no nos hubiese explicado conceptos tan básicos sobre Inteligencia Artificial y Redes Neuronales.

Por supuesto, también debo agradecerle la labor y el empeño puesto en este trabajo a mis compañeros de proyecto, José y Arturo. Tengo muy claro que sin ellos, el desarrollo de la idea habría sido imposible y la labor de investigación habría supuesto muchas más complicaciones y quebraderos de cabeza. Estoy seguro de que ha sido nuestro trabajo en equipo lo que nos he hecho poder avanzar de manera efectiva, siempre dando pasos firmes. En especial, quiero agradecer a Arturo el conocimiento que ha aportado a el proyecto, puesto que él conocía previamente conceptos del mundo de la Inteligencia Artificial. Gracias a él hemos podido aprender y aplicar conocimientos de una manera mucho más rápida y efectiva.

También quiero agradecer a la Universidad y a los profesores que me han enseñado todos los conceptos necesarios durante todos estos años de carrera. Gracias a ellos he podido conocer el mundo del software y de la informática en todo su esplendor y me han hecho darme cuenta de cual es mi lugar en la vida y qué es lo que quiero hacer con ella. Me han guiado con sus asignaturas a través de un camino difícil, lleno de retos, y me han hecho superarme y ofrecer lo mejor de mí.

Por último, me gustaría entrar en una parte de mi corazón que no suelo compartir con los demás porque creo que aún no estoy preparado para ello. Hoy, mientras termino lo que puede parecer un párrafo insignificante en comparación a todo lo que está por venir en esta memoria, cada palabra que escribo me trae a la mente todos los momentos que he pasado en mi vida. Me hacen darme cuenta de quienes han sido los apoyos, los cimientos, las bases... el todo de mi vida. Hola papá, hola mamá, estoy seguro de que en algún momento leeréis esto, estoy seguro de que os recordaré que reviséis esta parte, estoy seguro de que sentiré nervios por saber vuestra reacción. Creo que he dejado bastante claro lo que significáis para mi y lo que suponéis en mi vida, pero no pienso dejar pasar los días sin que os los recuerde: sois mi éxitos, mis fracasos, mis virtudes, mis defectos,

mi ser y estar, mi ayer, hoy y mañana, mi todo y mi nada. No puedo dejar este apartado sin acordarme de vosotros, aunque nunca dejaré de teneros en mi mente. A vosotros os debo en primer lugar mi vida, y en segundo lugar todo. Gracias a vosotros hoy puedo escribir esta memoria, hoy puedo cerrar este capítulo final para terminar esta carrera tan maravillosa. Gracias, soy mejor persona hoy en día por haber tomado como ejemplo a dos personas tan increíbles como vosotros.

Gracias Fran por ser mi enfermero, mi hermano mayor y mi amigo. Gracias a ti tengo claro que, no solo soy una persona sana de cuerpo, sino también de alma. Gracias por ser el escudo de mi mente, por salvarme de mi propio mar de ideas que a veces me ahoga y por liberarme de las cadenas que a veces me pongo a mi mismo y me impiden disfrutar.

Finalmente, me gustaría recordar unos versos de un poema escrito por Rudyard Kipling: «*Si puedes llenar cada implacable minuto, con sesenta segundos de combate bravío, tuya es la Tierra y sus codiciados frutos, y —lo que es más—: ¡serás un Hombre, hijo mío!*». Aún recuerdo cuando tú, papá, me mostraste este poema entre lágrimas y me hiciste prometer que estos versos fuesen el motor de mi vida, mientras enmarcabas el poema en mi habitación. Pues bién, así ha sido, y así será, puesto que no pararé ni un segundo en conseguir aquello que me propongo, y esto es gracias a vosotros.

Papá, mamá, Fran, abuela, abuelo, tito, tita, prima, primo... familia, desde lo más profundo de mi alma, os lo agradezco todo y os debo lo que está por llegar en mi vida. Simplemente, gracias.

José Morcuende Sierra

Estar escribiendo esto me da que pensar, esto supone el fin de una etapa. Supone abandonar una de las mejores fases, de la que mejores recuerdos tengo y mas apoyo he recibido, donde considero que he ganado verdaderos amigos.

En estos cuatro años ha habido tanto buenos momentos como malos, pero todos ellos forman parte de lo que llaman "*la vida universitaria*". Me gustaría empezar agradeciéndoselo mi padre y a mi madre, si no fuera por ellos no hubiera podido vivir esta experiencia, apoyándome año tras año, impulsándome a continuar y sacar lo máximo de mi, a no ponerme limitaciones, en general me han enseñado a confiar en mi mismo. Gracias a ellos he aprendido que intentarlo siempre merece la pena, que estas experiencias son realmente importante en la vida. Asimismo agradecérselo a mis tíos y a mi abuela por estar siempre pendientes, por llamarme cada vez que realizaba un examen o entregaba una tarea para preguntar que tal me había ido y darme ánimos fuese cual fuese el resultado.

También agradezco el haber conocido a Alejandro desde los inicios de de la carrera, gracias por todo el apoyo que me has ofrecido, tengo claro que sin tu ayuda hoy día no estaría donde estoy. No me olvido de Arturo, que a pesar de solo conocerle de vista por la facultad considero que me llevo una amistad más. Sin ellos todo esto no hubiera sido posible, ambos han sido primordiales para la realización y finalización de este trabajo, además de lo llevadero que han conseguido que sea este duro camino.

Otra parte importante es resto de las personas de la Universidad, el resto de amigos con los que he tenido la suerte de compartir clases y viajes. Asimismo, los profesores han tenido un papel muy relevante en estos cuatro años, han sido los encargados de enseñarnos poco a poco en que consiste de verdad esta carrera. No me olvido de nuestros directores, Luis Javier y Esteban, que nos dieron la oportunidad de realizar este trabajo de investigación, que a pesar de no estar nada acostumbrados, que junto con Ana Lucila nos han ayudado a completarlo, y más siendo en el campo de la Inteligencia Artificial, que personalmente no había visto nada nunca.

Por ultimo me gustaría agradecérselo a esa persona que conocí a mediados de agosto del año pasado, a la que en este tiempo he cogido mucho cariño, por todo lo que has hecho por mi, por todo el apoyo que me has ofrecido de manera continuada desde que te conozco, por preocuparte todos los días tanto de mi como de mis estudios, ni te imaginas lo mucho que te agradezco que me hayas acompañado en el final de etapa, simplemente gracias Lidia.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Configuraciones

Lista de Acrónimos

AFEW	*AFEW 6.0*
AI	*Artificial Intelligence*
ALSTM	*Association Long Short Term Memory*
ANN	*Artificial Neural Network*
C3D	*3D Convolutional Neural Networks*
CB	*Celebrity-1000*
CNN	*Convolutional Neural Networks*
CSV	*Comma-Separated Values*
CV	*Computer Vision*
DL	*Deep Learning*
FER	*FER-2013*
FPS	*Frames Per Second*
IOU	*Intersection over Union*
LFW	*Labeled Faces in the Wild*
LSTM	*Long Short Term Memory*

mAP	*Mean Average Precision*
ML	*Machine Learning*
NAN	*Neural Aggregation Network*
RL	*Reinforcement Learning*
RNN	*Recurrent Neural Networks*
SL	*Supervised Learning*
SSD	*Single Shot Detector*
SSH	*Secure Shell*
SVM	*Support Vector Machine*
TL	*Transfer Learning*
UL	*Unsupervised Learning*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*
YF	*YouTube Faces*
YO	*YouTube-Object*

Abstract

Today, neural networks are a fundamental pillar in the development of intelligent applications. However, this field is still in a phase of expansion and many advances are being made every day. Artificial intelligence is responsible for creating autonomous systems and for automating tasks that require a great investment of time and effort to be carried out by human beings, such as the identification of objects in both images and videos. On the other hand, digital forensic analysis has undergone a considerable advance in recent years thanks to the continuous technological evolution that is currently taking place. If artificial intelligence and forensic analysis are brought together, this work can be considerably improved, thus opening the door to future advances that will give way to functionalities that are still unknown.

This work is focused on trying to create optimal systems specialized in finding faces in videos with very unfavorable conditions, in order to find criminal behavior quickly and efficiently, thus facilitating the work of forensic analysis and thus get rid of the need to analyze the videos manually.

For this purpose, extensive research has been carried out both in the concepts that encompass the field of facial detection, as well as in previous work related to this topic, in order to understand in depth which are the current techniques in use that could serve for this work.

After carrying out extensive experimentation by testing six different models based on deep learning, the conclusion has been reached that two models should be proposed: one specialized in the detection of videos in premises capable of inferring with an accuracy of 74.83 %, and another adapted for real time operation with which an accuracy of 74.59 % is achieved.

Keywords: Forensics, Identification, Detection, Video, Neural Networks, Artificial Intelligence, Faces, Deep Learning

Resumen

Hoy en día, las redes neuronales constituyen un pilar fundamental en el desarrollo de aplicaciones inteligentes. No obstante, dicho campo se encuentra aún en fase de expansión y son muchos los avances que se consiguen diariamente. La inteligencia artificial es la responsable de crear sistemas autónomos y de automatizar tareas que suponen una gran inversión de tiempo y esfuerzo de ser realizadas por seres humanos, como la identificación de objetos tanto en imágenes como en vídeos. Por otro lado, el análisis forense digital ha sufrido un avance considerable en los últimos años gracias a la evolución tecnológica continua que se está viviendo actualmente. Si se juntan la inteligencia artificial y el análisis forense, se puede mejorar considerablemente dicha labor, y permitir así abrir las puertas a futuros avances que den paso a funcionalidades que aún están por conocer.

Este trabajo se centra en intentar crear sistemas óptimos especializados en encontrar rostros en vídeos con condiciones muy desfavorables, con el fin de encontrar comportamientos delictivos de manera rápida y eficiente, facilitando así la labor del análisis forense y desprenderse así de la necesidad de analizar los vídeos manualmente.

Para ello, se ha realizado una extensa investigación tanto en los conceptos que engloban el campo de la detección facial, como en trabajos previos relacionados con este tema, para comprender en profundidad cuáles son las técnicas actuales en uso que podrían servir para este trabajo.

Tras realizar una amplia experimentación probando con seis modelos distintos basados en aprendizaje profundo, se ha llegado a la conclusión de proponer dos modelos: uno especializado en la detección de vídeos en local capaz de inferir con una exactitud del 74,83 %, y otro adaptado para su funcionamiento en tiempo real con el que se alcanza una exactitud del 74,59 %.

Palabras Clave: Análisis Forense, Identificación, Detección, Vídeo, Redes Neuronales, Inteligencia Artificial, Rostros, Deep Learning.

Capítulo 1

Introducción

1.1. Motivación

Durante los últimos 70 años, los numerosos avances científicos han supuesto un crecimiento exponencial del mundo tecnológico. Concretamente, en el ámbito audiovisual, dicha evolución ha supuesto numerosos progresos que han repercutido positivamente en la calidad, en la velocidad de procesamiento y capacidad de almacenamiento de la imagen y del audio. Hasta hace poco, se requería una cámara especializada para grabar un vídeo y una innumerable cantidad de componentes extras para su correcta reproducción. Además, la calidad obtenida era mucho menor a la de hoy en día, suponiendo una pérdida de información importante a la hora de realizar un análisis forense sobre vídeos.

Con la llegada de la era digital, surgieron las cámaras digitales y esto consiguió arreglar los problemas comentados anteriormente. Con ellas, no era necesario equipamiento especial y la calidad obtenida era notablemente superior. Además, con el paso del tiempo surgieron nuevas formas de almacenamiento que permitían vídeos de mayor duración, haciendo que el análisis forense en vídeos fuese una tarea más eficiente y productiva.

Aún así, la calidad en los vídeos no era la deseada y, además, resultaba difícil la tarea de identificación cuando la calidad del entorno en el que se realizaba el vídeo estaba bajo condiciones muy desfavorables, como, por ejemplo, lugares oscuros. Gracias al avance del software y el surgimiento de los primeros programas de edición de imagen y vídeo, los metrajes podían ser modificados para mejorar ciertos aspectos de la composición, como el brillo, la saturación o el contraste, entre otros. El análisis forense se convirtió así en una tarea aún más productiva, pero aún había una problemática que dificultaba su evolución: exigía demasiado trabajo manual e intervención humana para el trabajo de identificación, y esto ralentizaba el proceso.

Por otro lado, el campo de la Inteligencia Artificial (del inglés, *Artificial Intelligence* (AI)) aún estaba en proceso de investigación. No fue hasta 1958 que apareció la primera arquitectura de red neuronal, Perceptrón [Sim20]. Desde entonces, las Redes Neuronales y la AI comenzaron a crecer a gran velocidad, en gran parte gracias a la evolución del hardware que permitía crear máquinas más potentes capaces de soportar la construcción de arquitecturas más complejas y eficientes.

El desarrollo de software cambió por completo con la incorporación y desarrollo de las Redes Neuronales. Como su utilidad abarca cualquier cometido, muchas fueron las empresas que desarrollaron Redes Neuronales para la identificación de objetos en imágenes para implementar sistemas autónomos que reaccionaran y tomaran decisiones en tiempo real teniendo en cuenta el entorno en el que se encontraran.

A priori podría parecer que el uso de la AI en el análisis forense solucionaría todos los problemas. Dicha tarea se automatizaría por completo, además de realizarse análisis exhaustivos en cuestión de minutos y de manera masiva. La calidad de la imagen se convierte en un problema menos prioritario, porque los sistemas analizan las imágenes de manera diferente a la que el ojo humano lo hace, encontrando patrones y anomalías que son imperceptibles para cualquiera. Pero, aun así, surge un nuevo tipo de problema: la necesidad de una red puntera y extremadamente eficiente, para evitar identificaciones erróneas, y esto, para la identificación de objetos y más concretamente de rostros, es un trabajo muy complejo.

Por tanto, un buen sistema de identificación de rostros, el cual sea capaz de realizar detecciones muy fiables y acertadas, además de conseguir realizar identificaciones en vídeos de baja calidad, es crucial para poder incorporar la AI en el análisis forense de manera plena y poder así prescindir de la intervención humana, que supone una ralentización increíble del proceso. La búsqueda de dichos sistemas es el motor que impulsa a la realización de este trabajo.

1.2. Contexto

Este Trabajo Fin de Grado ha sido realizado dentro del Grupo de Análisis, Seguridad y Sistemas (Grupo GASS, https://gass.ucm.es/, Grupo 910623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación THEIA (Techniques for Integrity and Authentication of Multimedia Files of Mobile Devices) con referencia FEI-EU-19-04.

1.3. Objeto de la Investigación

Una imagen contiene numerosas características simples de ver a los ojos de un ser humano, como pueden ser las siluetas de los objetos o el color que estos tienen. Estos patrones de manera conjunta hacen posible la identificación objetos en una imagen y, por consiguiente, sacar conclusiones sobre ésta, como el número de personas que se encuentran en ella o el momento del día en que fue realizada. Sin embargo, se ha descubierto que una AI lo suficientemente entrenada puede identificar ciertos patrones en las imágenes que se quedan fuera del entendimiento humano, y estos son determinantes a la hora de identificar objetos o cualquier otra tarea que implique el análisis de una fotografía.

Además, son muchos los identificadores ya creados y probados dedicados a identificar cuerpos enteros, y no rostros específicamente, y esto puede ser una desventaja cuando en una imagen el cuerpo no se muestra completamente.

Este trabajo propone distintas técnicas, algoritmos y modelos de Aprendizaje Profundo (del inglés, *Deep Learning* (DL)) que intentan realizar una identificación de rostros eficiente en imágenes y vídeos en *streaming*.

1.4. Plan de Trabajo

El desarrollo de este trabajo se ha realizado en tres fases:

1. **Investigación**: Durante los primeros tres meses se llevó a cabo un periodo de adaptación al contexto del trabajo y de adquisición de conocimientos necesarios para comenzar con el posterior desarrollo. Al comienzo, se realizó una reunión general en la que se explicó el punto de partida del trabajo, cuales iban a ser los objetivos a lograr y cuales iban a ser los conocimientos necesarios. Tras esto, se acordaron numerosas reuniones con los directores para el seguimiento del progreso de la investigación y resolver las posibles dudas que surgieran. Otro motivo por el cual se concertaron dichas reuniones fue para recibir charlas sobre los conceptos básicos de los distintos campos que concierne este trabajo. En los siguientes capítulos se hablará sobre ellos. Algunos integrantes del equipo ya disponían de dichos conocimientos gracias a las asignaturas de AI y Aprendizaje Automático (del inglés, *Machine Learning* (ML)), impartidas en esta Facultad. En las reuniones también se explicaron numerosas herramientas que facilitarían la labor de investigación. Una de estas herramientas fue *Google Scholar*, la cual el equipo usaría para buscar artículos científicos sobre investigaciones anteriores realizadas en el mismo campo de estudio. Finalmente, una vez obtenidas varias conclusiones acerca de qué se quería y cómo se quería conseguir, el equipo comenzó a dar prioridad al desarrollo.

2. **Desarrollo**: una vez se adquirieron los conocimientos necesarios para poder empezar a crear una versión sólida del proyecto, se decidió dedicar menos tiempo a la investigación y comenzar con la codificación de la propuesta. Aunque la fase de investigación no cesó, más bien pasó a un segundo plano, realizándose investigaciones sobre conceptos puntuales surgidos durante el desarrollo. Es por ello por lo que durante esta fase se investigaron conceptos avanzados del lenguaje de programación de *Python* y de librerías como *Keras* y *Tensor Flow*. Durante esta fase se construyeron también los conjuntos de entrenamiento para el modelo, obtenidos a partir de las fuentes proporcionadas por los trabajos leídos durante la fase anterior.

3. **Experimentación**: En esta fase se realizó la evaluación del prototipo desarrollado y análisis de los resultados obtenidos. Se utilizaron distintas herramientas proporcionadas por las mismas librerías explicadas anteriormente, se compararon los resultados con los obtenidos en los distintos trabajos revisados y se realizó la configuración de los parámetros para el afinamiento del modelo. Por ejemplo, se decidió ajustar los conjuntos de datos de entrenamiento para analizar su influencia en la mejora de los resultados. Cabe destacar que la fase de desarrollo no cesó durante la fase de experimentación, puesto que el modelo era constantemente comparado con otros y modificado para su optimización.

4. **Documentación**: A mitad del desarrollo, el equipo comenzó con la fase que consistiría en la escritura y constante revisión de la memoria final del proyecto. Dicha fase transcurrió de manera paralela entre las etapas de desarrollo y experimentación. Para la escritura de la memoria, el equipo realizaba constantes actualizaciones del contenido, a medida que el desarrollo iba incluyendo alguna tecnología nueva, la experimentación concluía en diferentes conclusiones, o, incluso, se investigaban artículos nuevos. Tras una actualización, el equipo revisaba los cambios realizados en busca de erratas o posibles apartados que mejorar. Finalmente, esta fase concluyó

tras terminar el último hito de la etapa de experimentación, compilando todos los resultados que se obtuvieron de dicha fase y añadiéndolas al Trabajo.

En la Tabla 1.1 se muestran las actividades realizadas durante el proyecto. Asimismo, en la Figura 1.1 se puede apreciar con más detalles la organización que el equipo ha seguido durante el transcurso de este proyecto, así como la duración de cada fase y de las subtareas que las componen.

Tabla 1.1: Tabla de hitos seguidos por el equipo seguido de sus identificadores

Identificador	**Tarea**
1	Investigación
1.1	Introducción al contexto de trabajo
1.2	Aprendizaje individual de conceptos básicos
1.3	Introducción a herramientas y software
1.4	Investigación de artículos académicos
1.5	Extracción de conclusiones y puesta en común
2	Desarrollo
2.1	Construcción de conjuntos de entrenamiento
2.2	Creación del prototipo (arquitectura)
2.3	Entrenamiento de los modelos
3	Experimentación
3.1	Experimentos con vídeos
3.2	Optimización de parámetros y entradas
3.3	Extracción de métricas y conclusiones
4	Documentación
4.1	Desarrollo de la memoria

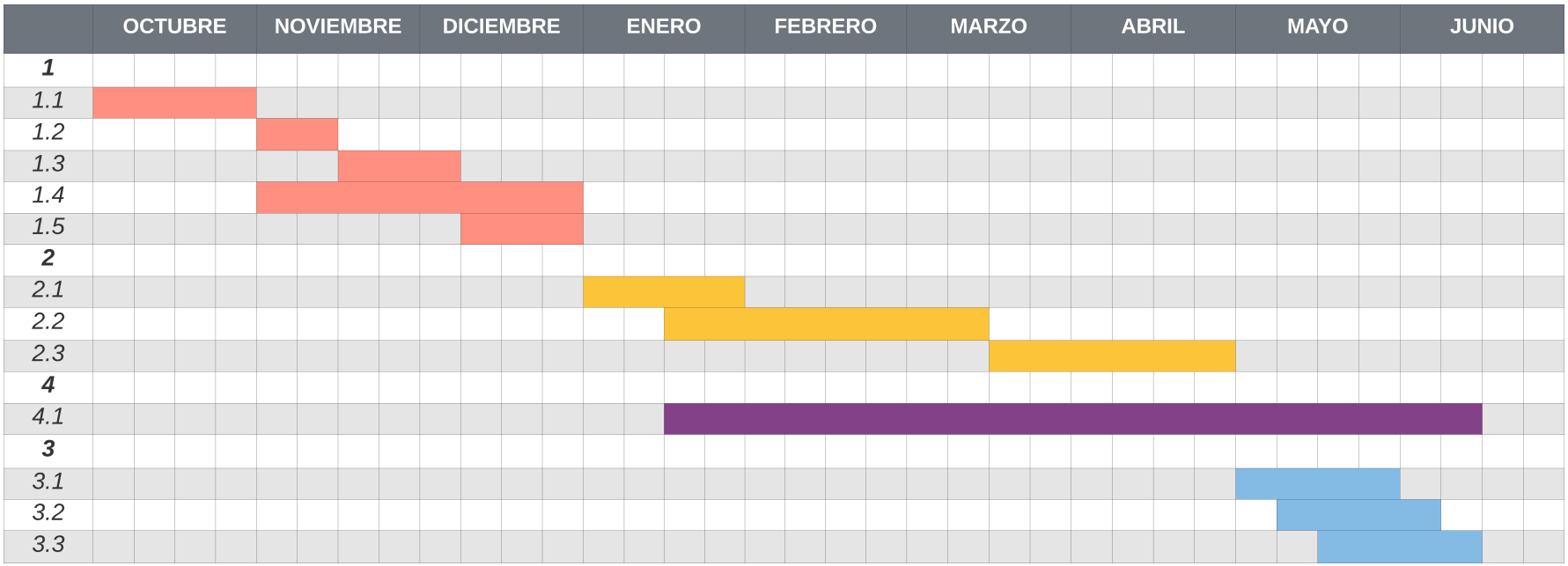


Figura 1.1: Diagrama de Gantt seguido en el desarrollo del trabajo

1.5. Estructura del Trabajo

El resto del trabajo está organizado en 9 capítulos con la estructura que se comenta a continuación:

El Capítulo 2 explica conceptos básicos tanto las bases de la AI como la detección de objetos. Concretamente, este capítulo centra ambos campos de estudio en el término que concierne a este trabajo, que es la detección de rostros en imágenes digitales y vídeos. Por tanto, en este capítulo se habla tanto de la historia que envuelve a la AI como de las técnicas utilizadas y los distintos modelos de Redes Neuronales.

El Capítulo 3 comienza mostrando un estado del arte en el que se presentan distintas técnicas e investigaciones realizadas sobre el campo de la identificación de rostros en imágenes, así como distintas arquitecturas y modelos presentados. Dichos trabajos representan la situación en la que se encuentra actualmente el campo de la detección de rostros en imágenes y vídeos. De cada artículo, se extrae tanto la idea y las técnicas utilizadas para el desarrollo de esta, como el conjunto de entrenamiento utilizado y las conclusiones a destacar en base a los resultados obtenidos. Dichos resultados se almacenan en distintas tablas para poder obtener comparativas entre los distintos trabajos. Finalmente se realiza una puesta en común de los distintos artículos para poder extraer así conclusiones acerca del estado actual del objeto de estudio en este trabajo, la identificación de caras en imágenes y vídeos digitales.

El Capítulo 4 presenta las contribuciones de este trabajo. Así pues, en primer lugar se presenta una extensa descripción de las tecnologías aplicadas para la creación de los modelos que posteriormente se pondrán a prueba en capítulos posteriores. También se explica detalladamente los distintos conjuntos de imágenes utilizados en el entrenamiento de los modelos y las distintas métricas utilizadas en las evaluaciones de la fase de experimentación.

El Capítulo 5 describe los experimentos realizados para evaluar la efectividad de los modelos propuestos a partir de las tecnologías explicadas en el Capítulo 4 y presenta los resultados obtenidos, así como la variación de dichos resultados en función de la optimización de la configuración inicial.

El Capítulo 6 relata las aportaciones personales de cada uno de los integrantes del equipo al desarrollo de este trabajo.

El Capítulo 7 muestra las principales conclusiones de este trabajo, las líneas futuras de investigación.

Por último, los Capítulos 8 y 9 son las traducciones al inglés de la Introducción y de las Conclusiones.

Capítulo 2

Marco Conceptual

En este capítulo se pretende explicar los conceptos, tendencias y las diferentes técnicas referentes a la AI y la detección de objetos, centrándonos sobre todo en el objetivo de detección de rostros en imágenes digitales y vídeos. Para ello, se explicará de dónde surge la AI y los hitos que se han conseguido previos a este trabajo, las técnicas más usadas a día de hoy en el campo de la AI, las bases sobre las que se cimienta el DL y las Redes Neuronales y, por último, cómo se ha aplicado todo este conocimiento para crear detectores capaces de encontrar un objeto, o rostro en este caso, de manera eficiente.

2.1. Historia de la Inteligencia Artificial

¿Se pueden crear máquinas inteligentes? Durante siglos, los grandes pensadores han utilizado esta pregunta como fuente de motivación para permitir el avance de la AI. Con el tiempo, el desarrollo de la AI ha dado lugar a que esta cuestión se empiece a relacionar cada vez más con el funcionamiento del cerebro, hasta tal punto que, actualmente, la investigación pasa por considerar el cerebro como un sistema computacional y la AI como un modelo que se encarga de emular el funcionamiento de este. Debido a esto, se podría decir que el fin último de la AI es conseguir que la máquina tenga una inteligencia similar a la humana, tratándose así de uno de los objetivos más ambiciosos que se han podido plantear en la ciencia.

Pese a que en los años 40 se publicaron algunos trabajos, la AI no gozó de una gran repercusión hasta el año 1950, con la difusión de un trabajo firmado por el matemático británico Alan Turing.

Alan Turing "considerado padre de la informática y precursor de la AI" dio sus primeros pasos en esta rama del conocimiento diseñando un programa que permitía a los usuarios jugar al ajedrez. Años después, escribió un artículo para la revista *Mind* [Tur50], en el que defendió el comportamiento inteligente que podrían llegar a tener las maquinas; y propuso un test, ahora conocido como el *Test de Turing*, que permitía determinar si se podía considerar inteligente a un computador o no. A día de hoy, se piensa que este test no es muy fiable y se han propuesto otras versiones que lo complementan.

A su vez, en los Ángeles, Belmont Farley y Wesley Clark realizaron en la *Western Joint Computer Conference* una presentación de cuatro trabajos: tres de ellos trataban sobre reconocimiento de patrones y el restante sobre máquinas para jugar al ajedrez. En uno de ellos, se podía ver cómo se estaba buscando la manera de relacionar el cerebro con una AI, y describieron la manera de reconocer patrones mediante el ajuste de los pesos de una red neuronal artificial.

Entre los años 1955 y 1956, Allen Newell y Herbert Simon, difundieron un programa conocido por el nombre *Logic Theorist*, que permitía demostrar teoremas de lógica proposicional que estaban contenidos en el libro *Principia Mathematica* [WR10]. A partir de aquí, la investigación en la AI comenzó a aumentar y se consiguieron distintos avances. En los años 60, en el *Stanford Research Institute*, Duda y Hart crearon con el lenguaje Fortran el primer sistema basado en redes neuronales que era capaz de aprender. Más tarde, se comenzaron a dar los primeros pasos en cuanto al procesamiento del lenguaje natural se refiere. [LdMM17]

Es importante destacar que, hasta 1980, no se volvieron a ver grandes avances en este campo. Aunque los ordenadores eran cada vez más potentes y económicamente menos costosos, aún se estaba lejos de conseguir un comportamiento verdaderamente inteligente por parte de las máquinas. Esta situación se agravó debido a una escasa financiación por parte de los gobiernos y al desarrollo de unos algoritmos que no eran muy eficientes en el ámbito de la AI.

A partir de los años 90, empezaron a surgir diversas aportaciones de valor para posibilitar el progreso en este área. Concretamente, un hito muy sonado es el que se consiguió en 1997, cuando una AI de IBM consiguió ganar al campeón del mundo de ajedrez [FH99]. Años más tarde, se consiguió crear un robot que interpretaba emociones y, a su vez, era capaz de expresarlas [Bre20]. Otro punto de inflexión se produjo en 2006 [HOT06], cuando G. E. Hinton, S. Osindero, y Yee-Whye Teh dieron a conocer mediante una publicación lo que más tarde se conoció como *Aprendizaje Profundo*. En términos generales, el trabajo explicaba arquitecturas de redes neuronales con múltiples capas, que permitían que estas aprendieran de una forma más aguda.

A estos diez últimos años se los conoce como la época dorada de la AI, debido a los grandes avances que se están produciendo, impulsados también por una mejora de los algoritmos y una mayor capacidad de cómputo. En 2012, Google se encargó de crear una red neuronal artificial que contaba con 16.000 procesadores a la que le suministraron 10 millones de miniaturas de vídeos de YouTube al azar [Cla20]. Como resultado, el sistema aprendió a reconocer rostros con una precisión del 81,7 %, partes del cuerpo humano con una precisión de 76,7 %, e identificaba gatos el 74,8 % de las veces. Por otro lado, en 2014, un bot superó por primera vez el famoso *Test de Turing*. Este sistema consiguió engañar a 30 de los 150 jueces a los que fue sometido el programa durante el test, haciéndose pasar por un niño de 13 años y manteniendo una conversación con ellos.

En la actualidad, la AI se está viendo muy favorecida por el uso del *Big Data*, gracias al cual se recogen una cantidad inmensa de datos y se procesan para poder extraer conocimiento posteriormente de ellos. [Sch97]. El beneficio de esto es que se está empezando a aplicar a diversos campos, como pueden ser la medicina, robótica, seguridad, transporte, comunicación, finanzas, sociedad, etc. Además, es evidente que grandes empresas como Amazon, Google, Apple, IBM y Microsoft, están realizando una gran inversión y generando un valioso desarrollo en este área, que acabará dando lugar a un mayor progreso con el paso de los años.

2.2. Técnicas Utilizadas en Inteligencia Artificial

A lo largo de los últimos 20 años, los investigadores se han centrado mayormente en diferentes conceptos para poder definir el término de AI. Esto ha día de hoy ha cambiado. Ahora se quiere que las máquinas se comporten de manera racional mediante la capacidad de actuar, sentir, pensar y razonar, por lo que se tiene que enfocar la AI de distintas formas.

El lado positivo de esto reside en que, todas estas formas de entender lo que es una AI, tienen algo en común: se basan en encontrar teorías y metodologías que puedan ayudar a las máquinas a entender el mundo y reaccionar a las situaciones de la misma forma que lo haría un humano. Estas teorías se pueden clasificar en distintos marcos de trabajo según el enfoque y el problema que resuelvan en el mundo real. Esta clasificación se muestra en la Figura 2.1.

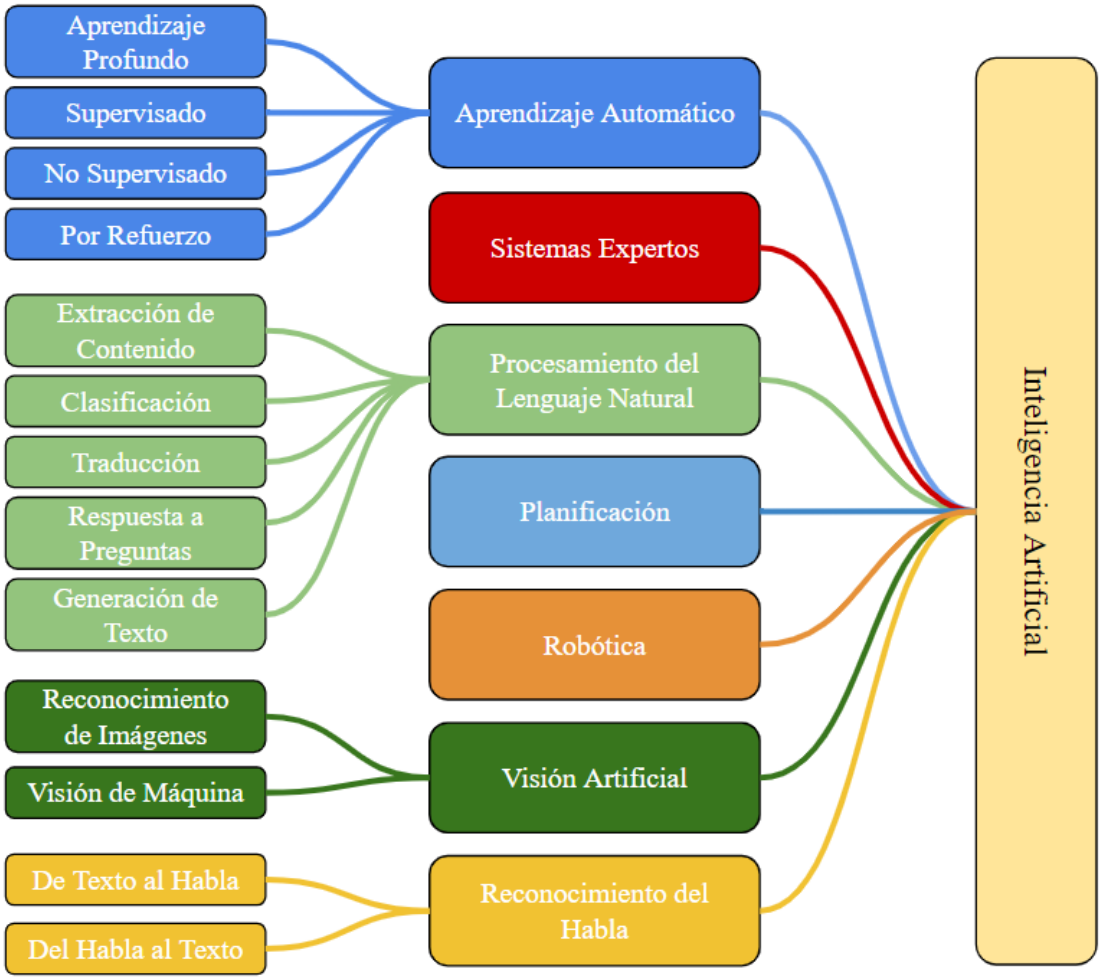


Figura 2.1: Principales aplicaciones de la Inteligencia Artificial

- **Aprendizaje Automático**: Posiblemente esta sea la disciplina más usada actualmente para crear comportamientos inteligentes. El objetivo es diseñar y desarrollar modelos que aprendan de datos ya existentes y realizar predicciones con nuevos datos. La mayor restricción de estos algoritmos es que están profundamente

limitados por los datos que se le suministren. De esta forma, si tenemos un conjunto de datos pequeño, el modelo resultante será bastante impreciso también. Este campo tiene distintas vertientes a su vez, como los que se pueden ver en la Figura 2.1. Además, cuenta con algunas técnicas como el Aprendizaje por Transferencia (del inglés, *Transfer Learning* (TL)), que nos confiere la habilidad de modificar modelos que ya han aprendido para ajustarlo a nuestras necesidades.

- **Sistemas Expertos**: Son sistemas que toman decisiones o proporcionan consejos usando técnicas de AI. En general, se utiliza en campos como las finanzas, marketing, medicina, etc. y extraen el conocimiento de una bases de datos para dar los consejos previamente dicho.
- **Procesamiento del Lenguaje Natural**: Este campo se encarga de procesar y entender textos. Generalmente, este tipo de técnicas se utilizan en motores de búsqueda, de manera que al usuario solo se le muestran los mejores resultados posibles en relación con su búsqueda realizada.
- **Planificación**: Este campo se ocupa de brindar el máximo rendimiento con unos costos mínimos en una planificación. Comienzan con unos determinados hechos y una especificación del objetivo a conseguir y generan el plan más óptimo para lograr los resultados.
- **Robótica**: En la robótica se combinan varios conceptos de la AI, ya que los robots tienen distintos sensores que, dependiendo de la situación, les permiten actuar de una manera u otra. Son capaces de adaptarse a nuevas situaciones pudiendo ver objetos, medir temperaturas, movimientos, etc.
- **Visión Artificial**: Como su propio nombre indica, hay sistemas que facilitan la tarea de otorgar la capacidad de visión a un computador, mediante el tratamiento de imágenes y vídeos. Estos algoritmos comprenden el contenido que se le suministra y extraen información de ello. Actualmente, la investigación se centra en aplicarlo a conducción autónoma y la realidad aumentada entre otros.
- **Reconocimiento del Habla**: Estos sistemas se encargan de oír y reconocer el habla. Día a día, se utilizan comúnmente en asistentes personales que son capaces de entender y realizar acciones específicas en base a lo que se les ha pedido.

Una vez se ha entendido cuáles son las distintas aplicaciones existentes en la AI, se va a profundizar aún más en aquellas técnicas que son necesarias para el objetivo de la detección de rostros en vídeos.

2.2.1. Aprendizaje Automático

Como se explica en [SF18], el ML es el campo de estudio que está caracterizado por ser un software que aprende de experiencias previas. En este tipo de programas lo que se intenta conseguir que estos algoritmos aprendan patrones que se hallan en los datos previos y produzcan unos resultados inteligentes para datos nuevos de entrada. Otro nombre por el que se conoce al ML es Aprendizaje Inductivo debido a que el código trata de inferir estructuras a través de los datos.

En ocasiones, modelar el comportamiento que tiene seguir un determinado programa puede ser una tarea bastante compleja, debido a que no es posible hacerlo mediante una secuencia de pasos claramente definida y en el algoritmo está involucrado algún tipo de patrón que denota o requiere cierta inteligencia. Para estos casos, el ML puede ser la herramienta correcta a utilizar.

Lo que este tipo de algoritmos proporciona es un conjunto de herramientas para escribir código sin definir todos y cada uno de los detalles del algoritmo. El programador se encarga de ajustar algunos parámetros y el resto se infieren mediante un proceso de aprendizaje llamado *entrenamiento*. Este proceso permite encontrar automáticamente los parámetros que mejor definen nuestro modelo. Generalmente, el rendimiento del modelo se puede mejorar mediante la disponibilidad de una mayor cantidad de ejemplos o datos que deben suministrarse como entrada del sistema en la fase de entrenamiento.

Se va a suponer el caso de que se está tratando de cocinar una receta. Si nunca antes se ha cocinado nada, puede llevar días descubrir cuál es la mejor combinación de ingredientes para hacer que el plato sea de buen gusto. Es esta creación de recetas la que da la capacidad de recordar rápidamente cómo hacer nuestro plato. Del mismo modo, el ML comparte esta idea. Como muestra la Figura 2.2, se puede decir que un algoritmo de ML se puede describir en dos etapas: *entrenamiento* y *predicción*.

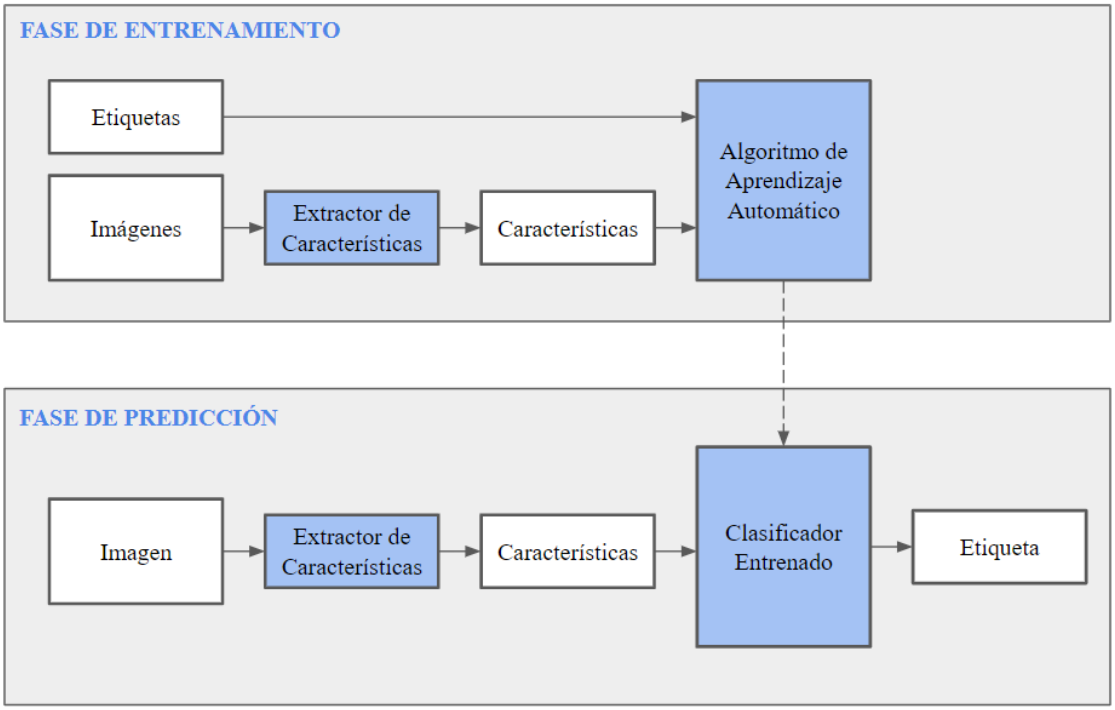


Figura 2.2: Fases de un algoritmo de Aprendizaje Automático

En la fase de entrenamiento, el objetivo que se persigue es describir los datos mediante el llamado *vector de características*, capaz de representar objetos del mundo real en una lista de atributos. Estos vectores son los que el algoritmo de aprendizaje utiliza para realizar su entrenamiento y obtener nuestro modelo. Esta fase es la que consume la mayor parte del tiempo, pudiendo durar horas, días o incluso semanas en completarse.

Por otro lado, se tiene la fase de predicción en la que el modelo previamente creado recibe un vector de características nuevo para él, es decir, un dato con el que no ha sido entrenado. Esta fase lleva mucho menos tiempo que la de aprendizaje —pudiendo ser incluso en tiempo real— y se encarga finalmente de hacer la predicción deseada.

Como es de esperar, el ML cuenta con distintos tipos de algoritmos de aprendizaje que pueden clasificarse a menudo en varios grupos. Los más importantes son los siguientes: *Aprendizaje Supervisado*, *Aprendizaje No Supervisado*, *Aprendizaje Por Refuerzo* y *Aprendizaje Profundo*.

Debido a la importancia de este último en la consecución de nuestro objetivo, se dedicará más adelante un apartado entero donde se explicará con más detalle.

- **Aprendizaje Supervisado**: El Aprendizaje Supervisado (del inglés *Supervised Learning* (SL)), se refiere a aquellos algoritmos que utilizan datos que deben ser etiquetados previamente por un humano antes de poder introducirse en el algoritmo. El objetivo que se persigue es poder etiquetar un nuevo dato basándose en las características encontradas en los datos de entrenamiento. La Figura 2.2 precisamente muestra estos tipos de algoritmos. Suponiendo que se quiere predecir los ingresos de una persona en base a su edad, localización, educación, etc. Para ello, se tendría que tener un conjunto de datos con esta información de las personas y etiquetar a cada una de ellas con los ingresos que recibe. De esta manera, se le diría al algoritmo qué parámetros corresponden con qué ingresos y el algoritmo aprendería a realizar ese mapeado. Las principales técnicas que se utilizan en ML son las siguientes: *Regresión Lineal*, *Regresión Logística*, *K-Vecinos Más Cercanos*, *Árboles de Decisión*, *Redes Neuronales Artificiales* y *Máquinas de Vectores de Soporte*

- **Aprendizaje No Supervisado**: El Aprendizaje No Supervisado (del inglés *Unsupervised Learning* (UL)), se refiere a aquellos algoritmos que no necesitan datos etiquetados para poder crear un modelo de aprendizaje. En cierta manera, es justo lo contrario a lo que hemos comentado en el apartado previo. Debido a que no hay etiquetas, se deben extraer patrones directamente de los datos. En este caso, si suponemos que existe una aplicación que se aprovecha de la variedad de usuarios que la utilicen y se quiere clasificar a estos para saber con qué tipo de personas se cuentan. Aquí no se sabría exactamente qué criterio seguir para poder separarlas. Por lo tanto, en esta situación entraría en juego el UL para clasificar de manera óptima a estas personas en distintos grupos. Las principales técnicas que se utilizan en UL son las siguientes: *Agrupación* y *Reducción de la Dimensión*

- **Aprendizaje Por Refuerzo**: El Aprendizaje Por Refuerzo (del inglés *Reinforcement Learning* (RL)), se refiere a aquellos algoritmos que, a diferencia del SL y del UL que están relacionados con el etiquetado, entrenan con la información que reciben del entorno cuando este reacciona a ciertas acciones. El objetivo es aprender qué combinación de acciones produce los resultados más favorables.

  Este tipo de algoritmos se suelen principalmente para crear la AI de videojuegos o en el campo de la robótica.

2.2.2. Aprendizaje Profundo

El DL es el subcampo del ML que pertenece a la familia de algoritmos de *Redes Neuronales Artificiales* y profundiza en ellos tanto funcional como estructuralmente. Estas se inspiran en el funcionamiento del cerebro humano y, como tal, tienen unos nodos conectados unos a otros y divididos en capas, los cuales se pasan información entre sí de manera que, cada capa de la red, se encarga de aprender unas determinadas características de la entrada en cuestión. Debido a la relevancia del funcionamiento de estas redes en este trabajo, se explicarán en secciones posteriores con más detalle.

Como se puede ver en la Figura 2.3, a diferencia de los algoritmos de ML donde la extracción de características de los datos la realiza explícitamente un humano, en DL pasa a ser responsabilidad de la arquitectura del propio algoritmo. Como consecuencia, la complejidad de la arquitectura aumenta considerablemente dado que es necesario poder tener más capas que procesen esa información.

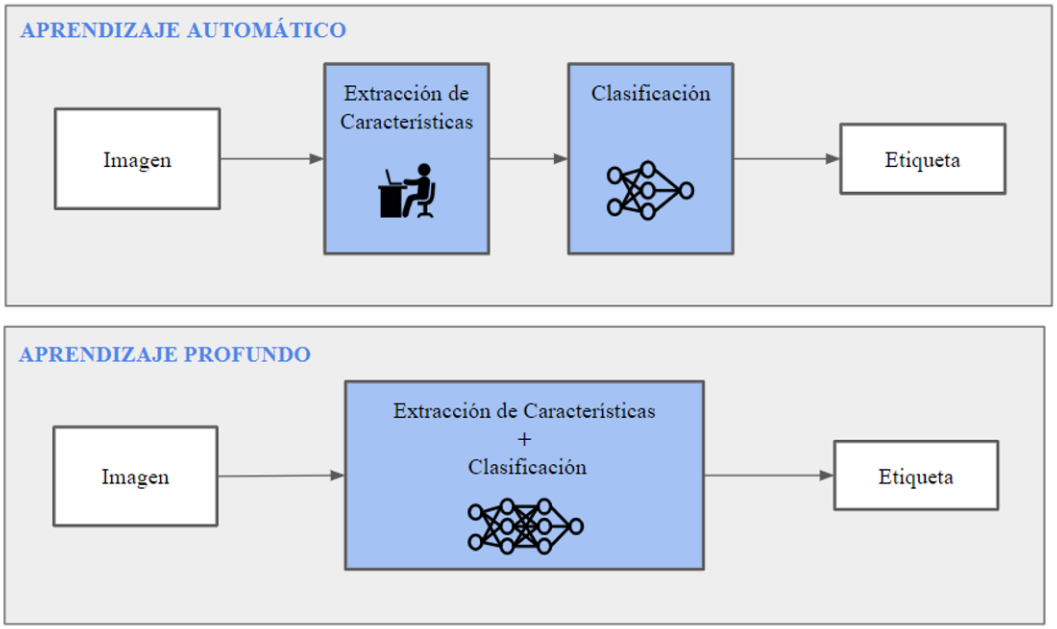


Figura 2.3: Diferencia entre algoritmo de Aprendizaje Automático y Aprendizaje Profundo

Pese a que los algoritmos de DL consiguen mejores resultados que un humano en la mayoría de los casos, no siempre es necesaria su utilización; esto dependerá de la complejidad del problema que estemos tratando de resolver. Al tratarse de algoritmos mucho más complejos que los relativos al ML, la capacidad computacional que pueden llegar a requerir puede ser inmensa en comparación con estos y, por tanto, es de extrema relevancia saber en qué momento tenemos que utilizar ML y DL. Dominando esto, se podrían reducir costes tanto en tiempo de ejecución como económicos, en caso de ser necesario utilizar plataformas de computación en la nube para ello.

En el caso de este trabajo, si se quisieran extraer las características que definan una imagen en concreto de forma manual, sería una tarea demasiado dificultosa, por lo que la utilización de DL se vuelve casi necesaria. Por tanto, mediante DL no solo se detectarán los rostros en una imagen dada, sino que además se hará sin necesidad de decirle a la red neuronal qué características definen un rostro.

2.2.3. Aprendizaje Por Transferencia

Como se expone en [Bro20b], el TL es una técnica usada en ML en la que un modelo que ya ha sido desarrollado y entrenado se reutiliza como base para crear otro modelo que realice una tarea similar.

Comúnmente, este método se utiliza en DL reutilizando modelos preentrenados para tareas de Visión Artificial y Procesamiento del Lenguaje Natural debido a la masiva cantidad de datos, recursos computacionales y tiempo que necesitan los modelos de DL para producir un algoritmo que funcione verdaderamente bien.

Para poder reutilizar el modelo sin perder el grado de aprendizaje que han alcanzado las capas en su último entrenamiento, se debe entrenar únicamente la capa de salida, la cuál deberemos adaptarla a nuestras necesidades según el tipo de tarea que queramos realizar.

Durante el proceso para realizar el TL, puede que se utilice todo o parte del modelo preentrenado dependiendo de la técnica que se utilice y el rendimiento que se quiera llegar a alcanzar. La técnica llamada *Afinamiento* (del inglés, *Fine-Tune*) es más complicada de realizar y con ella, no solo se reemplaza la última capa, sino que también se reentrenan algunas de las capas anteriores de manera selectiva. [Sar20]

El TL, por tanto, es una optimización que puede permitir ahorrar tiempo y obtener mejores resultados, aunque no tiene por qué ser así. Hasta que no se desarrolle y evalúe el modelo creado no se podrá saber si se ha desarrollado un beneficio o no, en cuanto a usar TL se refiere.

Como se puede ver en la Figura 2.4, hay tres formas por las que se puede averiguar si el modelo ha mejorado utilizando TL.

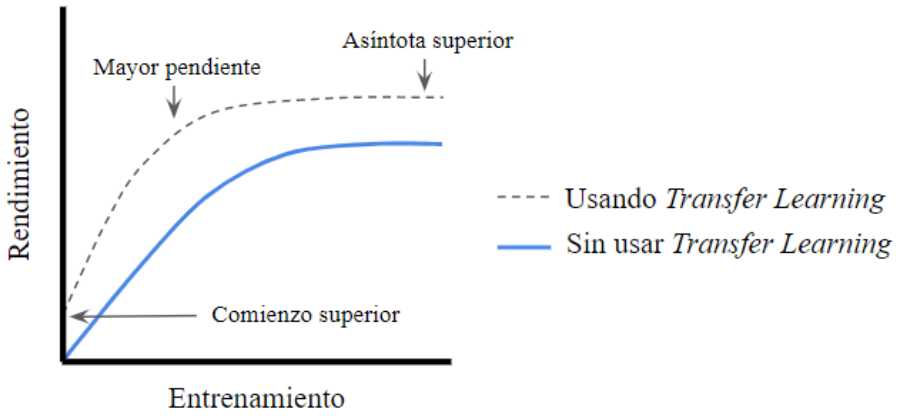


Figura 2.4: Signos de mejora usando Aprendizaje Por Transferencia

La primera es comprobando si el rendimiento inicial con el que se ha comenzado a entrenar el algoritmo es mayor de lo que sería de otra forma. La segunda opción que existe sería comprobar su pendiente. Cuanto más pronunciada sea la tasa de mejora durante el entrenamiento, mejor será el modelo. Por último, se podría comprobar que la convergencia final del modelo entrenado es superior en cuanto a rendimiento se refiere.

2.3. Visión Artificial

La Visión Artificial (del ingles *Computer Vision* (CV)) es el campo de la AI que otorga la propiedades de la visión humana a un computador, pudiendo tratarse de un dron, *smartphone*, *scanner*, etc. con varias cámaras integradas. El computador en cuestión debe ser el responsable de procesar y extraer información de las imágenes digitales que genere dicha cámara. [Sha18]

Existen diversas técnicas de DL que permiten poder solucionar los problemas que surgen a menudo en el campo de la CV:

- **Clasificación**: Es el proceso mediante el cual se etiqueta una imagen teniendo en cuenta su contenido visual, es decir, se trata de averiguar cuál es el concepto que aparece en ella. Por ejemplo, un algoritmo de clasificación de imágenes podría determinar si aparece una figura humana, un animal, una cara, etc.
- **Detección y Segmentación**: Se denomina *Detección* a la tarea de encontrar un determinado objeto o figura en una imagen y enmarcarlo con una caja delimitadora con su correspondiente etiqueta. Sin embargo, la *Segmentación* es diferente. Esta se encarga de hacer una clasificación por píxeles, bordeando la figura de aquello que queremos encontrar, en lugar de enmarcarlo con una caja delimitadora. Esta técnica se suele utilizar sobre todo en el procesamiento de imágenes médicas o imágenes obtenidas de satélites.
- **Aprendizaje Por Similitud**: En esta técnica se busca aprender de dos imágenes que son similares dándole importancia al significado semántico que reside entre ellas dos. Son algoritmos que se suelen utilizar a menudo en la identificación facial.
- **Subtitulado de Imágenes**: Como su propio nombre indica, se trata del proceso de describir la tarea que está sucediendo en la imagen.
- **Modelos Generativos**: Son aquellos modelos encargados de generar otras imágenes teniendo otras previamente como referencia.

2.4. Modelo Neuronal

A lo largo de los años, el DL se ha visto muy influenciado por el funcionamiento de nuestro cerebro y se ha inspirado en este para crear modelos computacionales y matemáticos que permitan simular su comportamiento. Para ello, es necesaria la creación de un modelo que reproduzca las neuronas biológicas y, de esta manera, se permita el paso de información de unas a otras. El modelo que se utiliza a día de hoy es el conocido como *Perceptrón* o *Neurona Artificial*.

Como se puede ver en la Figura 2.5, la neurona artificial funciona de tal manera que recibe un vector de entrada X, el cual se utiliza para realizar una suma ponderada con los pesos W asignados entre las entradas y el cuerpo de la neurona. Estos pesos son una representación de la importancia dada a cada entrada en concreto y su valor puede cambiar durante la fase de entrenamiento.

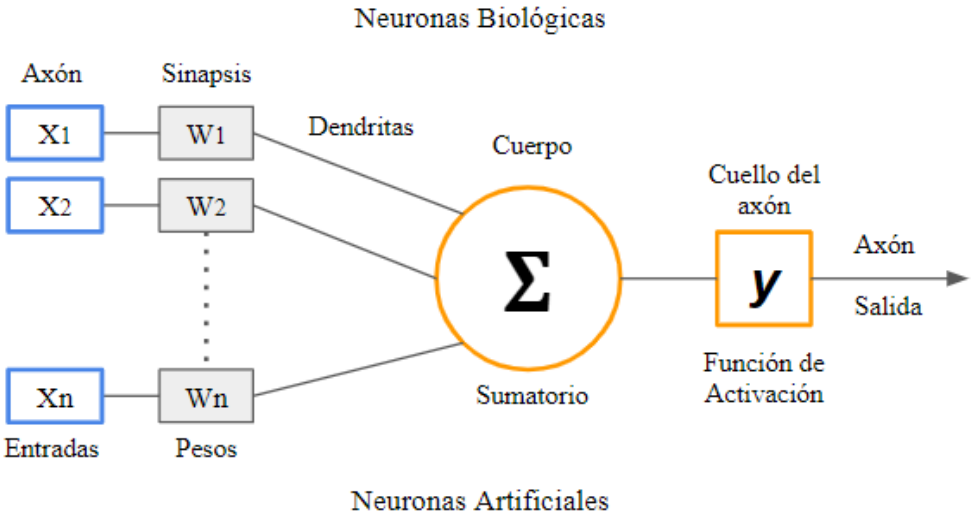


Figura 2.5: Comparación entre una Neurona Biológica y una Neurona Artificial

Se podría reflejar este cálculo mediante la siguiente expresión matemática:

$$S = \sum_{i=1}^{N} x_i * w_i \tag{2.1}$$

Para que la neurona pueda activarse durante el entrenamiento, el resultado de esta debe superar un cierto umbral, el cual viene determinado por la función de activación usada en la neurona. La salida S se pasa por dicha función y de esta forma el resultado que se obtiene es un valor no lineal, permitiendo así que las redes profundas puedan aprender funciones complejas.

Existen diversas funciones de activación a día de hoy. No obstante, las más conocidas y utilizadas son las siguientes:

- **Sigmoide**: Se considera que es una función de paso suavizado, es decir, derivable. Resulta muy útil cuando queremos calcular probabilidades y se suele utilizar comúnmente en problemas de clasificación binaria —por ejemplo, para determinar si una determinada imagen es un rostro o no— ya que transforma los valores de entrada en un rango de 0 a 1. Su fórmula matemática es:

$$f(x) = \frac{1}{1 - e^{-1}} \tag{2.2}$$

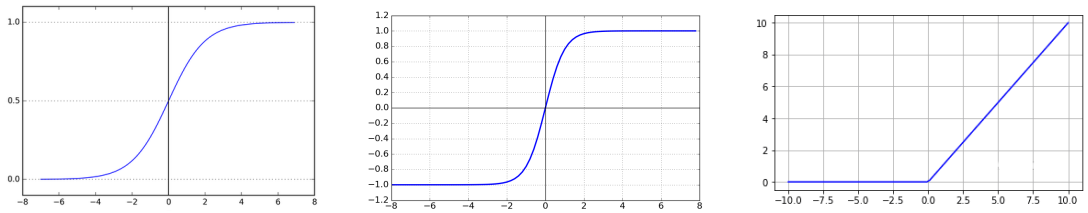
- **Tangente Hiperbólica**: Resulta muy útil cuando se quiere elegir entre una opción o la contraria, ya que transforma los valores de entrada en un rango de -1 a 1. Tiene una pendiente más acentuada que la Sigmoide y, por tanto, tiene menos problemas de desvanecimiento de gradiente, es decir, el valor de pesos varía más rápido y podría permitir un entrenamiento más rápido. Se puede expresar como:

$$f(x) = \frac{1}{1 - e^{-2x}} - 1 \tag{2.3}$$

- **ReLU**: La función ReLU es la que menos coste computacional exige para ejecutarla. Debido a que se mapea la entrada X al *max(0, x)*, hay veces que esta no se activa y, por tanto, permite que el entrenamiento se produzca de una manera más rápida. En general, funciona bien para una gran cantidad de problemas. Su expresión matemática es la siguiente:

$$f(x) = max(0, x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.4}$$

La Figura 2.6 muestra cada una de estas funciones gráficamente.



(a) Sigmoide (b) Tangente Hiperbólica (c) ReLU

Figura 2.6: Funciones de activación

2.5. Redes Neuronales Artificiales

Se puede definir una Red Neuronal Artificial (del inglés *Artificial Neural Network* (ANN)) como un conjunto de funciones de activación y neuronas conectadas unas a otras para formar capas ocultas que mapeen los datos de entrada con una salida, la cual produce un resultado de menor dimensión en comparación con el que tenía al entrar en la red. [Sha18] La ANN está estructurada de tal forma que cada capa tiene una serie de pesos los cuales se pueden ajustar para conseguir el objetivo deseado en el problema en cuestión que se esté tratando de resolver. Comúnmente, las redes más complejas —aquellas pertenecientes al DL— ofrecen un mejor rendimiento a costa de una mayor cantidad de cálculos y, en último término, mayores requisitos computacionales. Lo que realmente caracteriza a estas redes es su capacidad de aprendizaje a partir de un conjunto de datos, de los que extrae patrones y se ajusta a ellos mediante la creación de un modelo que permite generalizar para casos nuevos de entrada. A este proceso también se le puede llamar *entrenamiento de la red* y será explicado con más detalle en secciones posteriores.

Actualmente existen diversos tipos de ANN, cada una de ellas enfocada a distintos problemas que pueden surgir en el campo de la AI. Sin embargo, esta investigación se ha centrado sobre todo en indagar en aquellas que podrían ofreces mejores resultados para cumplir el cometido de detectar rostros en vídeo.

2.5.1. Redes Neuronales Convolucionales

En los últimos años, las Redes Neuronales Convolucionales (del inglés, *Convolutional Neural Networks* (CNN)) se han visto popularizadas debido a los grandes resultados que se pueden llegar a conseguir con ellas en el tratamiento de imágenes, impactando de esta manera al área de la CV.

Una manera no muy recomendada para entrenar un modelo de DL que utilice imágenes sería entrenarlo con ANN convencionales. En este trabajo, no se podrían usar estas ya que se necesitaría una entrada por cada píxel y resultaría en una red de un gran tamaño, incrementando así los requisitos de cómputo necesarios para poder entrenarlas.

Lo que diferencia a las CNN es que suponen que la entrada es una imagen, lo que permite programar distintas propiedades en la arquitectura para reconocer elementos en específico en una imagen [Tor18].

Los humanos, en general, pueden reconocer una cara ya que esta tiene una serie de características las cuales deben identificarse previamente, como por ejemplo: las líneas, bordes, texturas o formas. Una vez se han detectado estas, podemos determinar lo que es un rostro.

De manera similar funcionan las capas de una CNN. Cada una de ellas, aprende un nivel de abstracción, extrayendo distintas características de la imagen. Además, cada capa utiliza los datos de la anterior para extraer más información, por lo que el conocimiento cada vez es mayor. Cuanto más compleja sea la estructura a identificar, mayores capas debe tener la red. Podemos ver este proceso en la Figura 2.7 [LGRYTN11].

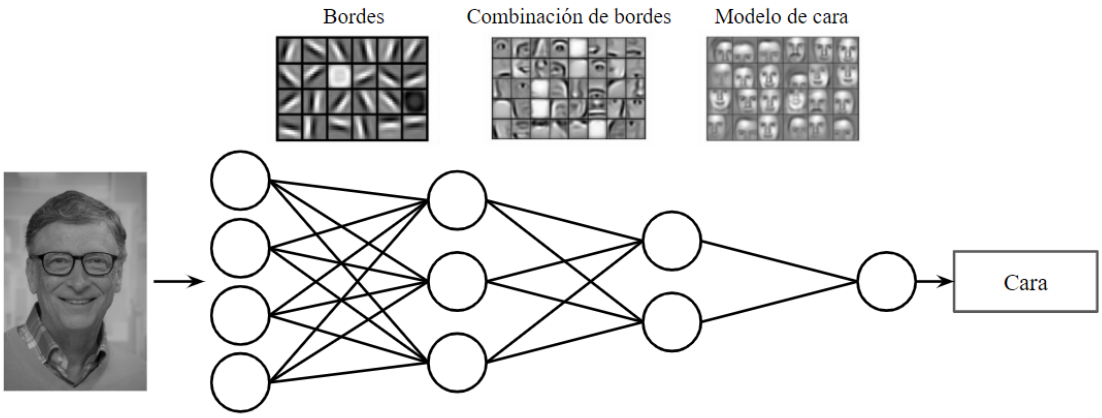


Figura 2.7: Aprendizaje por capas de una Red Neuronal Convolucional

Las CNN tienen una serie de componentes básicos que se van repitiendo a lo largo de la arquitectura hasta que se consigue cumplir con el objetivo deseado. Estos componentes son dos capas de neuronas llamadas: *Convolución* y *Pooling*.

2.5.1.1. Capa de Convolución

La capa de convolución es una operación que, a diferencia de las capas densamente conectadas donde se aprenden patrones globales de las entradas, se encargan de encontrar patrones locales en pequeñas ventanas de dos dimensiones.

Se va a suponer el ejemplo de una imagen de 28x28 píxeles perteneciente a un rostro y una ventana de tamaño 5x5. Durante la operación de convolución lo que se hace es recorrer la imagen de izquierda a derecha y de arriba a abajo, de manera que, mediante el deslizamiento de la ventana por esta capa de entrada, se asigna el procesamiento de cada ventana a una neurona de la capa siguiente. Este deslizamiento se realiza mediante el llamado *stride*, que configura la longitud de los píxeles a avanzar con cada paso.

Cada una de las neuronas de la capa siguiente, deben conectarse con las 25 neuronas (píxeles en este ejemplo) que le corresponden de la ventana asignada en la capa anterior y, para ello, se usa un valor de sesgo b y una matriz de pesos W de tamaño 5x5 conocidos como *filtro* o *kernel*. A la neurona que procesa esta información se le asigna el valor correspondiente al producto escalar entre el filtro y los 25 píxeles (5x5).

Se debe utilizar un mismo filtro para todas las neuronas de la capa siguiente, ya que este solo permite detectar una característica concreta de la imagen. Por tanto, se deben usar múltiples filtros —dependiendo de la cantidad de características que se quieran detectar— para realizar el reconocimiento de imágenes de manera completa. La capa de convolución queda representada en la Figura 2.8 para su mayor entendimiento.

2.5.1.2. Capa de Pooling

Además de la ya mencionada capa convolucional, las CNN tienen la conocida operación de *Pooling*, realizada justo después de esta. En la capa de *pooling* se condensa la información que se ha obtenido con la convolución y se crea una versión más compacta de la misma.

Para realizar esta simplificación, se debe definir una nueva ventana más pequeña que la utilizada en la capa anterior (por ejemplo, 2x2). Acto seguido, se tiene que utilizar una de las técnicas conocidas como *max-pooling* o *average-pooling*, las cuales se quedan con el mayor valor o la media de todos los valores contenidos en la nueva ventana respectivamente.

Debido a que la capa de convolución alberga diversos filtros, hay que aplicar una de las técnicas previas a cada uno de ellos. La capa de *pooling* acabará teniendo tantos filtros de *pooling* como convolucionales. De nuevo, en la Figura 2.8, se puede ver un esquema de cómo funcionan dichas capas.

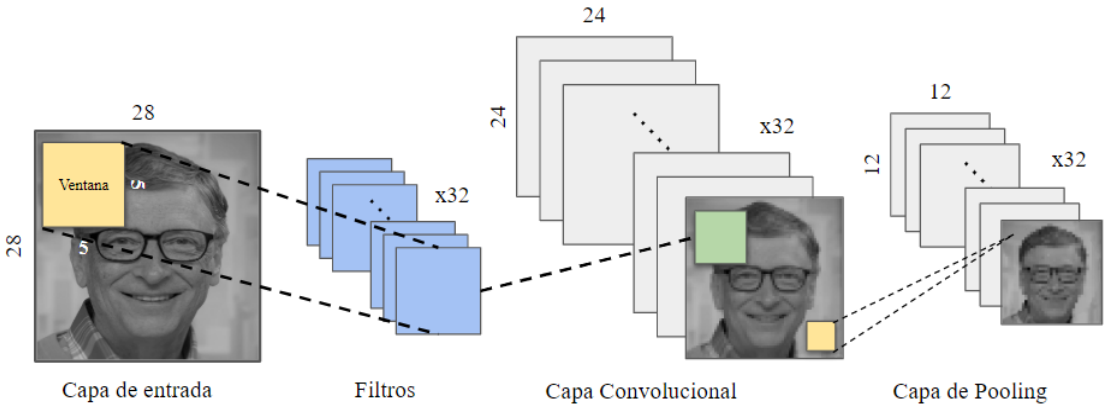


Figura 2.8: Representación de las capas de Convolución y *Pooling*

Dentro de la comunidad de DL, hay diversas arquitecturas de CNN que tienen nombre propio debido al gran rendimiento que ofrecen. Algunos ejemplos son: *Xception*, *InceptionV3*, *VGG16*, *ResNet50*, *InceptionResNetV2*, *MobileNet*, etc.

Muchas de ellas, se pueden encontrar preentrenadas en distintos *frameworks* de desarrollo utilizados en DL, por lo que se vuelven muy interesantes a la hora de querer practicar TL.

2.5.2. Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (del inglés *Recurrent Neural Networks* (RNN)) se comportan de manera diferente a las ANN tradicionales. Estas se encargan de añadir al modelo información secuencial, es decir, tienen en cuenta también el contexto en el que se desarrolla aquello que queremos predecir.

El planteamiento es similar: las RNN tienen también unos pesos que se siguen consiguiendo mediante el entrenamiento, diferenciándose en que, con este enfoque, ahora se introduce un peso de transición que debe ser transmitido al siguiente estado de la red. Esto indica que para poder predecir algo, no solo hay que tener en cuenta la entrada de la red, sino también el resultado de la predicción anterior.

Esta característica dota a las redes de una memoria que podría ser de gran ventaja para tareas relacionadas con el reconocimiento del habla —donde hay un contexto claramente— o detección de patrones en el tiempo.

2.5.3. Memoria a Largo y Corto Plazo

Las RNN desgraciadamente parten con una desventaja, y se trata de su incapacidad para recordar patrones durante largos periodos de tiempo.

Para solucionar esto, surgieron las Memorias a Largo y Corto Plazo (del inglés *Long Short Term Memory* (LSTM)) que, como queda explicado en [Ola20], son un tipo de RNN que funciona, para muchas tareas, mucho mejor que la versión estándar.

En general, la arquitectura de todos los tipos de RNN tiene una forma de cadena donde se van pasando la información de módulo en módulo. En su versión más simple, la estructura se basa en una sola capa de *tanh (tangente hiperbólica)*. Sin embargo, las LSTM tienen cuatro capas que interactúan entre sí de manera muy especial. En la Figura 2.9 se puede ver la diferencia entre esas dos arquitecturas.

El diagrama anterior queda representado de la siguiente manera:

- Cada línea transporta un vector completo, comunicando así distintos nodos.
- Los círculos de color rosa dentro de cada nodo, corresponden con operaciones como la suma de vectores, multiplicación, etc.
- Los cuadrados amarillos denotan capas de redes de neuronas, pudiendo distinguir entre las sigmoidales σ y la *tanh*.
- Cuando dos líneas se fusionan, denota una concatenación. Mientras que si hay una bifurcación significa que el contenido se copia en diferentes ubicaciones.

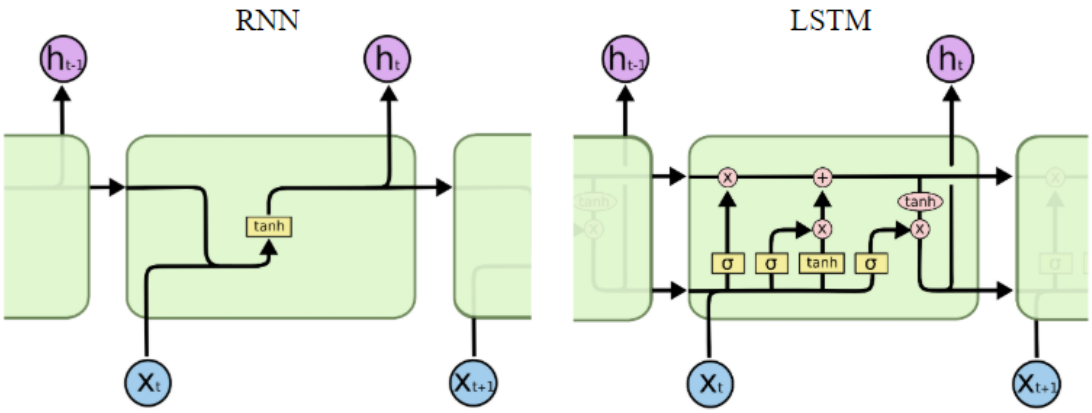


Figura 2.9: Diferencia entre una RNN y una LSTM

La LSTM está fuertemente caracterizada por su capacidad para eliminar o agregar información al estado de la celda anterior. Esto se regula con unas estructuras llamadas puertas, que están formadas por las capas y operaciones previamente mencionadas.

Para decidir qué información hay que pasar a nodos posteriores, se deben asegurar los siguientes pasos:

1. Decidir qué información se va a eliminar del estado anterior. Esta decisión se toma en una capa llamada *forget gate*.
2. Asegurarse de qué información se va a añadir al estado actual del nodo. Esto se produce en dos partes. La primera se realiza en una capa llamada *input gate* y la segunda en una capa de *tanh*. Una vez hecho eso, se debe combinar los dos para dar lugar a la actualización del estado.
3. Decidir qué se va a generar como salida. Esta se basará en el estado actual.

En términos generales, se puede afirmar que la mayoría de los grandes resultados obtenidos hasta la fecha con RNN, son en gran medida gracias a este tipo de redes. Esto las hace un tipo de red bastante interesante que deben ser tenidas en cuenta cuando el objetivo sea preservar las decisiones que se han tomado en el tiempo.

2.6. Entrenamiento de las Redes Neuronales

Durante el entrenamiento de una ANN, se produce el ajuste de los pesos que conectan las capas. Es aquí, en esta etapa, donde se aprenden qué características son importantes para describir y entrenar nuestro modelo. Una vez se ha creado el modelo de aprendizaje, e incluso durante su entrenamiento, se puede hacer una monitorización mediante las llamadas *curvas de aprendizaje*, que permiten diagnosticar el rendimiento de nuestra ANN. A continuación se explicará con más detalle los conceptos mencionados.

2.6.1. Proceso de Entrenamiento

Como deja claro el autor Jordi Torres [Tor18], el proceso de entrenamiento queda marcado por un constante "ir y venir" por las capas de la ANN. El hecho de ir hacia delante se conoce como *forward propagation*, mientras que su opuesto es comúnmente conocido por *backpropagation*.

El primer paso se produce cuando los datos de entrenamiento se exponen a toda la red y estos la cruzan para acabar calculando las predicciones. Durante esta propagación, cada neurona aplica su función de activación al dato que reciben y se lo pasan a la neurona siguiente. Una vez se ha recorrido, se usa la función de *loss* o error, para estimar y medir cómo de bueno o malo ha sido nuestro resultado en comparación con el que debería dar, el cual queda establecido por la etiqueta que se ofrece con cada dato de entrenamiento.

El segundo paso se realiza cuando ya tenemos calculado el error. Este se propaga hacia atrás, partiendo desde la capa de salida y pasando por todas y cada una de las neuronas que tiene la red y que han contribuido a las predicciones realizadas. No obstante, cada neurona solo recibe un error proporcional a la contribución que han realizado para producir la salida.

Esta última propagación es la que permite finalmente ajustar todos los pesos existentes en las conexiones que unen las neuronas. El objetivo que se pretende conseguir de esta forma, es que el error se aproxime cada vez más a cero para realizar mejores predicciones cada vez que se vuelva a utilizar la red.

Para minimizar el *loss* hay diversas técnicas, pero la más conocida es el *descenso de gradiente*. Esta se ocupa de utilizar la derivada de la función de error para encontrar el mínimo global mediante la variación de los pesos en pequeños incrementos. En general, esto se realiza en unos lotes de datos llamados *batches* durante una serie de iteraciones conocidas como *epochs*.

2.6.2. Sesgo y Varianza

El sesgo y varianza (del inglés, *underfitting* y *overfitting* respectivamente) son dos conceptos contrapuestos que constituyen uno de los problemas fundamentales del ML, ya que lidian con la optimización y la generalización del modelo [F.18].

La optimización se ocupa de ajustar el modelo a los datos de entrenamiento proporcionados, con el objetivo de conseguir el mejor rendimiento posible, mientras que la generalización hace referencia a cómo de bien o mal se realiza la predicción sobre datos que nunca se han visto.

Un modelo de ML puede encontrarse en distintos durante su entrenamiento. Puede hallarse en un estado de *underfitting*, es decir, no ha podido aprender de los datos correctamente y aún es incapaz de generalizar de manera adecuada. Por tanto, esto indica que puede optimizarse aún más. Por otro lado, también puede encontrarse en un estado de *overfitting*, o dicho de otro modo, ha aprendido a ajustar tan bien a los datos de entrenamiento que le resulta imposible generalizar bien sobre datos nuevos. Esto indica que está demasiado optimizado.

El objetivo es encontrar el balance entre estos dos extremos y, para ello, se pueden usar la generación *curvas de aprendizaje* para diagnosticar su rendimiento, las cuales muestran cómo ha ido avanzando el error con los datos de entrenamiento y validación.

Un modelo que sufra sesgo, puede ser identificado únicamente por su curva de entrenamiento. Se podrá detectar si esta permanece plana, sigue disminuyendo hasta el final del entrenamiento o marca un error demasiado alto. En la Figura 2.10, se muestran algunos ejemplos [Bro20a].

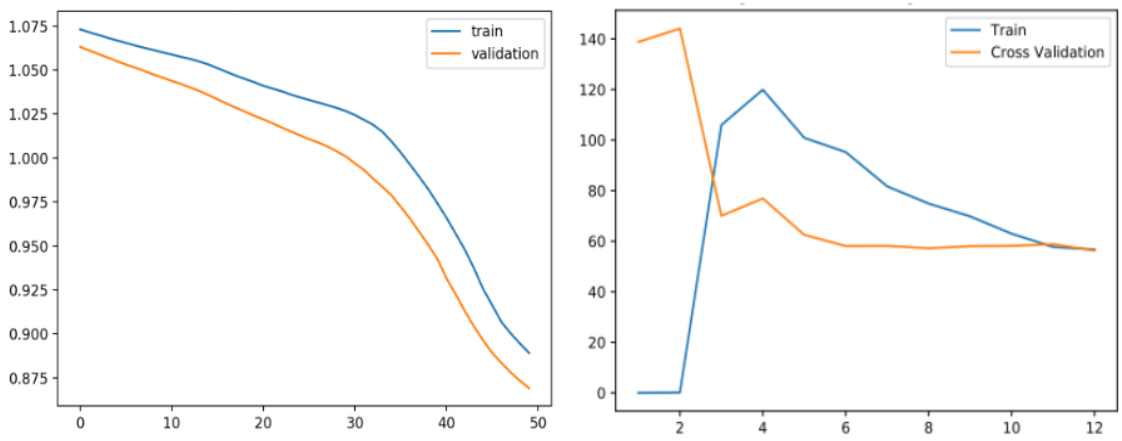


Figura 2.10: Ejemplos de Curvas de Aprendizaje con Sesgo

Por el contrario, un modelo que sufra varianza, se debe identificar con las dos curvas: tanto la de entrenamiento como la de validación. Este estado se caracteriza por tener una curva de entrenamiento con bajo error y con tendencia a disminuir con cada experiencia, y por curva de validación que decrece hasta un punto y más tarde comienza a crecer.

La situación ideal que se busca reside en tener una curva de entrenamiento optimizada en coste y una de validación que se adapte a ella durante todas las iteraciones.

En la Figura 2.11 se pueden ver unos ejemplos [Bro20a].

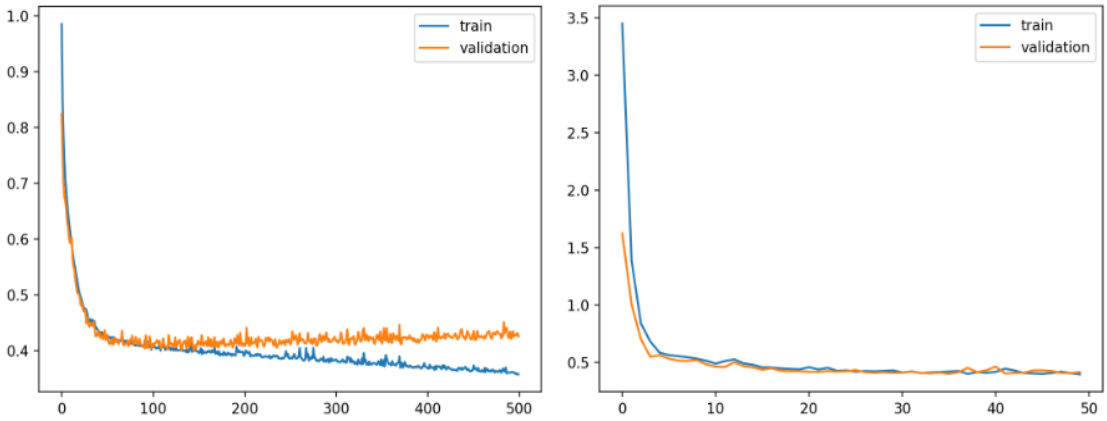


Figura 2.11: Ejemplo de Varianza y de un Aprendizaje Ideal

Con el objetivo de obtener el mejor rendimiento y evitar las situaciones extremas que se acaban de explicar, se pueden realizar una serie de técnicas que permiten abordarlo de una manera apropiada. Por ello, se citarán algunas de las más utilizadas a día de hoy, según artículos científicos publicados recientemente [Sil20].

Para solucionar el sesgo:

- Incrementar la complejidad del modelo.
- Aumentar el número o tamaño de parámetros que este utiliza.
- Entrenar el modelo durante un mayor tiempo.

Para corregir la varianza:

- Utilizar regularización, lo cual lleva al modelo a tener ecuaciones menos complejas.
- Probar con el método de validación cruzada.
- Parar el entrenamiento antes de que la curva de validación comience a ascender.
- Utilizar la técnica de *dropout*, según la cual se ignoran neuronas durante el entrenamiento.
- Añadir más datos de entrenamiento.

2.7. Detección Facial

Dos términos que suelen confundirse muy a menudo son la detección facial y el reconocimiento facial. La detección facial es una tecnología que forma parte de una más general: la detección de objetos. Esta se encarga de encontrar objetos en vídeos e imágenes digitales y tiene un coste computacional mucho mayor que aquellas que permiten clasificar objetos únicamente, ya que no solo se predice la clase a la que pertenece una determinada imagen, sino que además se encarga de señalar en qué coordenadas se encuentra dicha predicción. Sin embargo, el reconocimiento facial es una tecnología que, además de realizar la tarea de detección, identifica quién es la persona que ha detectado.

Con el tiempo, algunas técnicas de CV se han vuelto cada vez más rudimentarias y se han visto sustituidas por ML y ANN que permiten realizar la misma función de una manera más eficiente y eficaz. A día de hoy, existen diversos métodos para realizar esta tarea y se dividen en el número de etapas en el que consiguen realizar la detección. Por ello, se tienen *detectores de dos etapas* y *detectores de una etapa*.

2.7.1. Detectores de Dos Etapas

Como queda expuesto en [JZL+19], los detectores de dos etapas, tienen una alta precisión de localización y de reconocimiento de objetos. Utilizan una ANN llamada *Region Proposal Network* para realizar una búsqueda de regiones de interés que más tarde deben pasarse a una etapa de clasificación y delimitación de las coordenadas que identifican el objeto.

2.7.1.1. Red Neuronal Convolucional Basada en Regiones

La Red Neuronal Convolucional Basada en Regiones (del inglés, *Region-Based Convolutional Neural Network*) conocida más comúnmente por R-CNN, fue propuesta por Ross Girshick [GDDM14] como uno de los primeros trabajos en demostrar que una CNN podría elevar el rendimiento en tareas de detección de objetos.

La R-CNN se compone de tres módulos que interactúan de manera secuencial:

- El primero, se encarga de generar y extraer 2000 regiones llamadas propuestas de región. Estas son unos cuadros que delimitan los potenciales candidatos a ser detectados.
- El segundo, recoge las regiones producidas en el módulo anterior y se encarga de extraer las características de las imágenes mediante una CNN, como se explicó en apartados previos. La red, produce un vector de salida de 4096 elementos que describe el contenido de la imagen.
- El último módulo, utiliza el vector del paso anterior para introducirlo en una *Support Vector Machine* que se ocupa de clasificar la presencia del objeto deseado en la región propuesta. Además de ello, se predicen también cuatro valores con las coordenadas en las que se sitúa el objeto.

Una de las principales desventajas que presenta esta aproximación reside en que requiere que cada región sea pasada por una CNN, haciéndolo de esta manera un algoritmo lento, puesto que tiene 2000 regiones para analizar. Se puede ver la arquitectura en la Figura 2.12

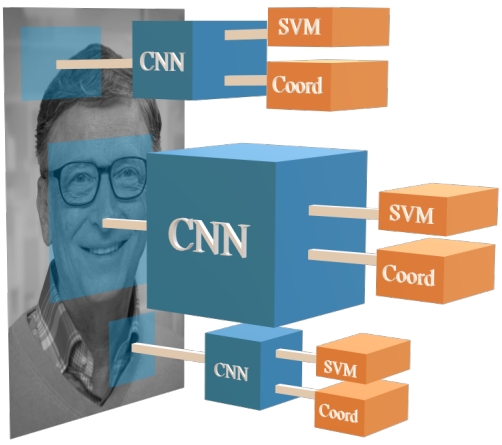


Figura 2.12: Arquitectura del Modelo R-CNN

2.7.1.2. Red Neuronal Convolucional Rápida Basada en Regiones

Ross Girshick al ver que tuvo éxito en su implementación de la R-CNN, se decidió a implementar una mejora del modelo, centrándose sobre todo en hacerla con un mayor rendimiento y más rápida. Para ello, propuso una *Fast R-CNN*, es decir, una Red Neuronal Convolucional Basada en Regiones (del inglés, *Fast Region-Based Convolutional Neural Network*) [Gir15].

En su implementación, tomó como base el modelo anterior pero ahora, en lugar de pasar cada propuesta por una CNN, genera un mapa de características convolucional pasándole la imagen entera una sola CNN. A partir de este mapa, se trata de identificar las propuestas de región, se agrupan en una capa y posteriormente se pasan a otra capa totalmente conectada. Finalmente, acaba usándose *Softmax* para predecir la clase y, como en el modelo anterior, se vuelven a predecir las coordenadas en las que se sitúa la región.

Como puede verse en la Figura 2.13, se puede intuir que esta aproximación es mucho más rápida que la anterior, ya que no tiene que utilizar una CNN por cada región que se ha propuesto, es decir, no utiliza 2000 CNN.

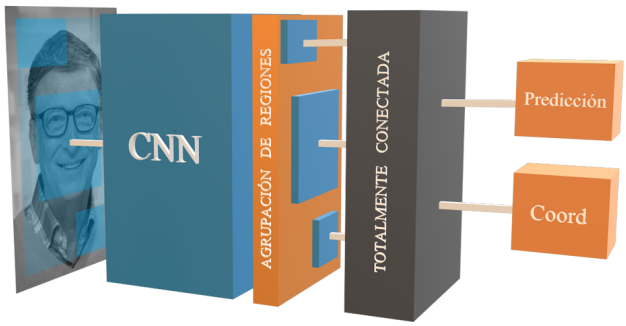


Figura 2.13: Arquitectura del Modelo *Fast R-CNN*

2.7.1.3. Red Neuronal Convolucional Más Rápida Basada en Regiones

Hace apenas cuatro años, Shaoqing Ren consiguió mejorar aún más la velocidad de entrenamiento del *Fast R-CNN* y propuso el *Faster R-CNN* o, dicho de otro modo, la Red Neuronal Convolucional Más Rápida Basada en Regiones (del inglés, *Faster Region-Based Convolutional Neural Network*) [RHGS15].

En las dos propuestas anteriores, se tiene que realizar una búsqueda para poder conocer cuáles son las regiones propuestas. Esto hace que sea un proceso lento y provoca un gran impacto en el rendimiento. Por ello, en este modelo y, al igual que en *Fast R-CNN*, se alimenta una única CNN con la imagen y se utiliza otra red que permite predecir las propuestas. Estas se vuelven a agrupar en una nueva capa y posteriormente se clasifican y predicen las coordenadas del objeto. Se puede ver su arquitectura en la Figura 2.14.

Hasta el día de hoy, el *Faster R-CNN* es el modelo más rápido y eficaz en cuanto a detección de objetos se refiere. Por ello, es una opción a tener muy en cuenta en el desarrollo de este trabajo.

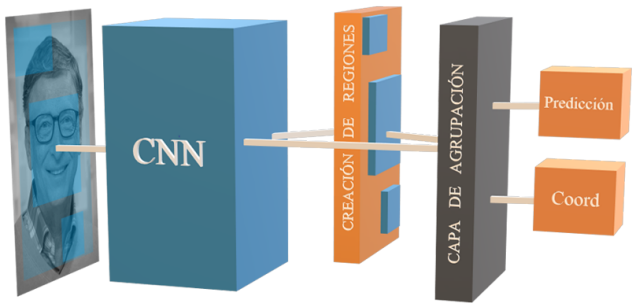


Figura 2.14: Arquitectura del Modelo *Faster R-CNN*

2.7.2. Detectores de Una Etapa

A diferencia de los detectores anteriores, los de una etapa son buenos cuando se quiere conseguir una alta velocidad de inferencia, por lo que serían convenientes en tareas que se deban realizar en tiempo real ya que son bastante más rápidos que los comentados previamente.

Funcionan de tal manera que se les proporciona una imagen a la red y proponen cuadros para realizar las predicciones, evitándose así la etapa de búsqueda de regiones de interés.

2.7.2.1. Solo Miras Una Vez

El modelo Solo Miras Una Vez o YOLO (del inglés, *You Only Look Once*), fue propuesto por Josheph Redmon y Ross Girshick en 2015 [RDGF16].

Esta aproximación de modelo de detección funciona de manera muy distinta a todos los propuestos anteriormente. En YOLO, se coge una imagen y se utiliza una sola CNN. Concretamente, lo que se hace es dividir la imagen en una cuadrícula de tamaño NxN y cada cuadro resultante se vuelve a dividir en M cuadros delimitadores.

Para cada uno de estos, la CNN predice la clase a la que pertenece devolviendo una probabilidad y unos valores de desviación asignados a cada cuadro delimitador. Si esta probabilidad supera un cierto umbral, se tendrá en cuenta el cuadro y formará parte del cuadro delimitador total que señalará la detección. Su arquitectura se puede ver en la Figura 2.15.

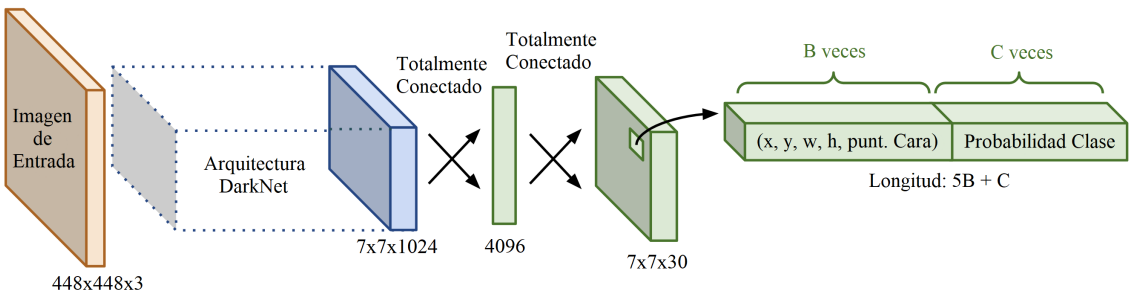


Figura 2.15: Arquitectura del Modelo YOLO

La desventaja que ofrece YOLO se basa principalmente en que no es capaz de detectar objetos de manera precisa cuando estos son demasiado pequeños, por lo que no sería una buena opción para detectar rostros si estos se encuentran alejados de la cámara que los captura.

2.7.2.2. Detector de Un Solo Vistazo

El Detector de Un Solo Vistazo (del inglés, *Single Shot Detector* (SSD)), fue una propuesta de C. Szegedy que lanzó en 2016 [LAE+16a] y con la que alcanzó grandes resultados en cuanto a precisión y rendimiento en tareas de detección se refiere, alcanzando una precisión media del 74 % en conjuntos de imágenes como COCO o PascalVOC.

Se puede ver en la Figura 2.16 que la arquitectura de la SSD se componen principalmente de la CNN conocida por VGG-16. Esto se debe a su gran rendimiento en cuanto a clasificación de imágenes de alta calidad se refiere. A esta CNN le siguen diversas capas más que se encargan de extraer una mayor cantidad de características en múltiples escalas.

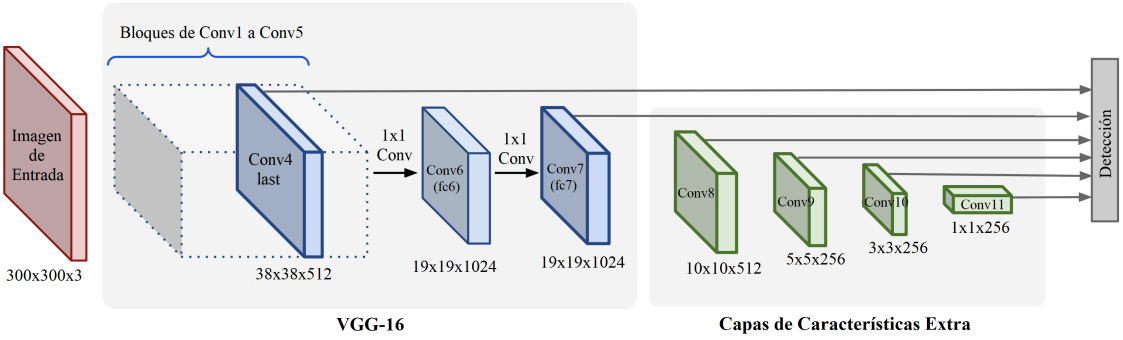


Figura 2.16: Arquitectura del Modelo SSD

Debido a esta última peculiaridad de la red, se considera que el SSD podría ser capaz de detectar rostros con una mayor eficacia en objetos pequeños, a diferencia de YOLO que no era capaz.

Capítulo 3

Estado del Arte

En este capítulo se pretende ilustrar como está la situación actual referente a la detección de rostros en imágenes digitales y vídeos en función de los trabajos leídos.

De cada uno de ellos se desarrollará qué es lo que quieren conseguir, la técnica utilizada para llevarlo a cabo y los conjuntos de datos que utilizan, para posteriormente ver los resultados que han obtenido.

En la actualidad hay mucha información referente a la detección de objetos tanto en imágenes [LWZ11, PVZ15, DAHG+15] como en vídeos [DAHG+15, MMdRM16, KOLW16, FLLL16, YRZ+17, LLT17]. Además [LWZ11, PVZ15, YRZ+17] se centran en la detección de rostros.

En todos ellos se aprecia el gran potencial que tienen las redes CNN para este cometido que además se pueden concatenar con otras como las LSTM [DAHG+15] o las Redes Neuronales Convolucionales 3D (del inglés, *3D Convolutional Neural Networks* (C3D)) [FLLL16] para obtener mejoras notorias en la detección.

Se ha decidido explicar estos artículos [PVZ15, FLLL16, YRZ+17, LLT17] ya que cada uno se centra en una arquitectura diferente en la red para completar la detección.

3.1. Red de Agregación Neuronal para el Reconocimiento Facial en Vídeos

En [YRZ+17] se propone una Red de Agregación Neuronal (del inglés, *Neural Aggregation Network* (NAN)), que puede ser entrenada mediante SL para reconocer caras en vídeos o en conjuntos de imágenes. Los autores explican que el problema de los vídeos es que es difícil conseguir una representación acertada del rostro debido al constante movimiento de los objetos y el ruido que genera estos, una aproximación para resolverlo es dividir el vídeo en fotogramas y pasar cada uno a redes que ya saben reconocer rostros pero esto es demasiado costoso.

Esta red está compuesta por dos módulos que pueden ser entrenados por separado. El primero es un extractor de características a nivel de fotograma que usa un modelo de CNN profundo. El otro es un módulo de agregación que fusiona los vectores de características de los fotogramas del vídeo. En general la red toma de entrada un conjunto de imágenes con caras y devuelve un vector con las características para la tarea de reconocimiento facial.

Uno de sus objetivos es diseñar un mejor esquema para las ponderaciones para conseguirlo proponen que el módulo de agregación debe ser capaz de procesar un número diferente de imágenes, que la agregación se debe realizar de forma independiente al orden de entrada y que el módulo debe adaptarse a la entrada y tener parámetros que se puedan entrenar a través del SL. Para ello emplean bloques de atención inspirados en el mecanismo de atención de la memoria descritos en [GWD14, SWF15, VBK15].

Un bloque de atención se encarga de leer los vectores de las características del primer módulo y generar pesos lineales para cada uno.

Solo usando un bloque de atención, se observa como los pesos son elevados en las imágenes con rostros de más calidad, como las de alta resolución y fondos simples, y es más reducido en las imágenes con la cara desenfocada, parcialmente oculta o con una exposición inadecuada. Para mejorar todavía más el rendimiento deciden usar dos bloques de atención en cascada, de esta forma se establecen los pesos de las imágenes por cada identidad para que los pesos no queden influenciados por las imágenes de las demás identidades.

El entrenamiento se decide realizar por separado para exprimir al máximo las capacidades de cada uno de los módulos. Para la CNN del extractor usan tres millones de imágenes con rostros de cincuenta mil identidades diferentes. Las caras son detectadas usando el método de JDA [CRW+14], y alineadas con el método LBF [RCWS14]. El modulo de agregación se entrena sobre las características extraídas con la CNN.

Una vez esta entrenadas ambas, se obtiene como resultado un $95.72 \pm 0.64\,\%$ de exactitud para el conjunto de imágenes de *YouTube Faces* (YF) [WHM11] donde comparado con otros modelos como DeepFace-single [TYRW14], DeepID2+ [SWT15] y Wen *et al* [WZLQ16] obtiene mejores resultados, aunque no supera el $97.3\,\%$ de FaceNet [SKP15].

En la Tabla 3.1 y en la Tabla 3.2 se pueden ver los resultados con el conjunto de imágenes *Celebrity-1000* (CB) [LZLY14], donde comparado con los métodos de MTJSR [LZLY14] y Eigen-PEP [LHS+14] obtiene unos valores notablemente más elevados tanto en el conjunto cerrado como abierto.

Tabla 3.1: Resultados de NAN sobre el conjunto de datos de CB con conjunto cerrado

	Número de Sujetos			
Método	**100**	**200**	**500**	**1000**
NAN - VideoAggr	88.04	82.95	82.27	76.24
NAN - SubjectAggr	90.44	83.33	82.27	77.17

Tabla 3.2: Resultados de NAN sobre el conjunto de datos de CB con conjunto abierto

	Número de Sujetos			
Método	**100**	**200**	**400**	**800**
NAN - SubjectAggr	88.76	85.21	82.74	79.87

El conjunto de imágenes de YF [WHM11] está diseñado para la verificación de rostros en vídeos. Contiene 3425 vídeos con 1595 personas diferentes. La longitud de los vídeos varia entre 48 y 6,070 fotogramas con una duración de promedio de 181,3 fotogramas.

El conjunto de imágenes de CB [LZLY14] está diseñado para la identificación de rostros en vídeos, contiene 159.726 secuencias de vídeo de 1.000 personas diferentes. La longitud de los vídeos es de 2.4M de fotogramas en total, con 15 fotogramas por secuencia. Tiene dos tipos de protocolos, uno de conjunto abierto y otro de conjunto cerrado. Se pueden encontrar los detalles de estos protocolos en [LZLY14].

3.2. Reconocimiento Profundo de Rostros

En [PVZ15], los autores tiene dos metas: investigar cual es la mejor arquitectura y CNN para reconocer y verificar caras; y establecer un procedimiento para crear conjuntos de datos a gran escala.

En cuanto a las arquitecturas se centran en *DeepFace* [TYRW14]. Este método usa una CNN profunda entrenada con 4 millones de imágenes, También utiliza una arquitectura de red siamesa, donde se aplica la misma CNN a pares de caras para obtener descriptores que luego se comparan usando la distancia euclidiana. Además, las imágenes son pre-procesadas. Ese proceso consiste en "recolocar" las imágenes, situando las caras en una pose canónica usando un modelo 3D.

La estrategia que proponen consta de varias etapas para recopilar de manera efectiva un gran conjunto de datos faciales que contenga cientos de imágenes.

Lo primero es obtener una gran lista de nombres de los que potencialmente se pudieran obtener información, para ello se eligen celebridades, políticos, etc, en general personajes públicos. La lista inicial la obtienen de *IMDB* con un gran número de nombres de celebridades. Esos nombres los cruzaron con *Freebase knowledge graph* [BEP+08], que es una base de datos con información de muchísimas celebridades. Tras todo esto obtuvieron un total de 5000 nombres de personas para los que se consiguieron 200 imágenes por persona simplemente usando el buscador de imágenes de *Google*. Posteriormente realizaron un filtrado, eliminando las personalidades cuyo conjunto de 200 fotos no superaba el 90 % de pureza (llamando pureza a la heterogeneidad de las imágenes del conjunto). Esto redujo la lista de candidatos a 3250. Por último, eliminaron las celebridades que aparecían en otros conjuntos de datos para tener datos nuevos con los que entrenar a la red. Tras este filtrado, terminaron con 2622 nombres de celebridades.

El segundo paso es obtener más imágenes de esas personas. Para ello buscaron los nombres en el buscador de imágenes de *Google* y en el de *Bing*, que les proporcionaron como resultado 2000 imágenes por cada persona.

El tercer paso es mejorar la pureza del nuevo conjunto de imágenes. Para ello por cada celebridad se entrena una red SVM lineal usando en descriptor Fisher Vector Faces [SPVZ13, PSVZ14] que es capaz de filtrar automáticamente las mejores imágenes, en las que las 1000 mejores permanecen y el resto de desechan.

El cuarto paso es eliminar duplicados ya que al usar dos motores de búsqueda puede haber imágenes iguales. También en esta fase se eliminan las imágenes que solo se diferencian por el balance de color o con texto superpuesto. Esto lo realizaron calculando el descriptor VLAD [JPD+11, AZ13] para cada imagen.

Para el último paso se tienen 2622 identidades y unas 1000 imágenes por cada uno. La idea de esta fase es aumentar la pureza del conjunto de datos. Para ello hacen uso de una CNN multidireccional con la arquitectura de *AlexNet* [KSH12] que determina que el conjunto de imágenes de una identidad es buena si la pureza es mayor al 95 %.

Tras todo esto, el numero total de imágenes es 982803 con aproximadamente el 95 % de ellas con rostros frontales y el otro 5 % de perfil.

Sugieren otras alternativas a este proceso como cambiar la fuente de *Freebase knowledge graph* con las de *DBPedia (Structured WikiPedia)* o *Google Knowledge Graph*. También se pueden recolectar imágenes de *Wikimedia Commons*, *IMDB* o de otros buscadores como *Baidu* o *Yandex*.

Para la etapa de entrenamiento, todas las imágenes se reescalaron a un ancho y alto de 256 píxeles, además, se voltearon las imágenes con una probabilidad del 50 %, sin embargo, no realizó ningún aumento del canal de color como se describe en [SZ14] y [KSH12].

Se usó la técnica de *triplet loss* para la clasificación de imágenes, donde el esquema de entrenamiento es similar al de [SKP15].

Una vez está entrenada la red se obtiene como resultado un 98.95 % de exactitud para el conjunto de imágenes de *Labeled Faces in the Wild* (LFW) [HMBLM08] que frente otros modelos como Fisher Vector Faces [SVZ14] DeepFace [TYRW14], Fusion [TYRW15], DeepID-2,3, FaceNet [SKP15] y FaceNet [SKP15] + Alineado, solo es superado por este último que otorga un 99.63 % de exactitud aunque requiere de 200M de imágenes para el entrenamiento, frente a las solo 2.6M del DeepFaceRecognition [PVZ15] que es el que menos requiere de todos los comparados.

El conjunto de imágenes de LFW [HMBLM08] está diseñado para estudiar el problema del reconocimiento facial sin restricciones. Contiene 13233 imágenes de rostros recopiladas de la web etiquetadas con el nombre de la persona. En total se tienen 5749 identidades distintas, de 1680 de ellas se tienen dos o más imágenes.

Además de este también se comparan los resultados con el conjunto de imágenes de YF [WHM11]. En esta ocasión se obtiene como resultado un 91.6 % de exactitud que al ser inferior a varios de los otros modelos comparados, Video Fisher Vector Faces [PSVZ14], DeepID-2,3 y FaceNet [SKP15] + Alineado, se le añade aprendizaje incrustado con su técnica *triplet loss*, lo que permite aumentar la exactitud a un 97.3 % obteniendo el mejor resultado de los comparados. También hay diferencias en el valor de 100 %- EER (tasa de error igual). Se puede ver esta diferencia en la Tabla 3.3.

Tabla 3.3: Resultados de *DeepFace* sobre el conjunto de datos YF

Método	**Imágenes**	**100 % - EER**	**Exactitud (%)**
DeepFaceRecognition	2.6M	92.8	91.6
DeepFaceRecognition+Embedding learning	2.6M	97.4	97.3

3.3. Reconocimiento de Emociones en Vídeos Usando CNN-RNN y Redes Híbridas C3D

En [FLLL16], los autores proponen un sistema para reconocer emociones en vídeo. El módulo principal es una red híbrida que combina RNN con C3D. La RNN se encarga coger las características extraídas con por una CNN sobre los fotogramas de un vídeo como entrada y codifica el movimiento, mientras tanto, la C3D lo modela.

Para reconocer dichas emociones, anteriormente se utilizaban combinaciones de CNN con RNN. Actualmente, las C3D están haciendo grandes avances en el tratamiento de varias tareas de análisis sobre vídeos y dando resultados más eficientes en las pruebas con vídeos [TBF+15], por ello se propone hacer una red híbrida con ambas.

Las RNN llevan mucho tiempo siendo usadas para en reconocimiento manuscrito y oral [GLF+08, SSB14], pero tienen ciertos problemas para recordar estados a largo plazo, por lo que proponen sustituirlas por LSTM [DAHG+15].

Las C3D se pueden entender como CNN de tres canales, en las que las operaciones no solo se realizan espacio-temporalmente, es decir, sobre el fotograma, si no que también añaden la dimensión temporal [TBF+15].

Para mejorar aún más el rendimiento de esta red híbrida, entrenan una Máquina de vectores de soporte (del ingles *Support Vector Machine* (SVM)) para clasificar el sonido de los vídeos. Esto puede traer una mejora de afinidad de aproximadamente un 3 % [YSMC15], además, el sonido puede ser imprescindible para reconocer mejor algunas expresiones.

Puesto que este proyecto fue realizado durante una competición, los participantes solo disponen de los conjuntos de imágenes proporcionados por el certamen. Ese año fueron el *FER-2013* (FER) [CCG+13], *AFEW 6.0* (AFEW) [DGLG12, DGJ+16] y HAPpy PEople Images [DGG15, DGJ+16] por ello los resultados se basan en las mezclas de estos.

El conjunto de imágenes de FER [CCG+13] está compuesto de 35685 imágenes de 48x48 píxeles en escala de grises, categorizadas en función de la emoción que muestran. El conjunto de imágenes de AFEW está compuesto de 1750 trozos de vídeos, divididos en tres partes: 774 para entrenamiento, 383 para validación y 593 para pruebas.

Las imágenes son preprocesadas, se extrae la cara del resto de la imagen con el detector Viola-Jones [LWZ11] y se alinea de acuerdo a los rasgos faciales principales. Además, para los fotogramas sin rostro detectados se les aplica un filtro basado en una CNN para determinar si el detector ha cometido un error y tener o no en cuenta esa imagen.

La red C3D está formada por 8 capas convoluciones, 5 funciones de *max-pooling* y 2 capas completamente conectadas, seguidas de una capa de salida *softmax* dispuestas como se muestra en la Figura 3.1, el resto de parámetros son similares a los descritos en [TBF+15].

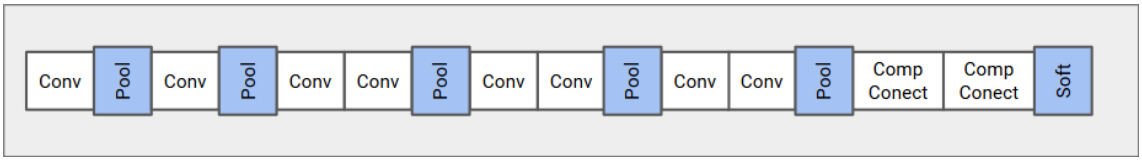


Figura 3.1: Arquitectura del Modelo C3D

Para el entrenamiento de la RNN se toman como entrada en cada iteración, 16 características aleatorias. El resultado final de la predicción es la combinación de las predicciones obtenidas de entrenar de forma separada la LSTM, la C3D y la SVM de audio. Este resultado se divide en 7 predicciones, una por cada emoción de las que se detectan en este *paper*: enfadado, disgusto, miedo, felicidad, tristeza, sorpresa y neutral.

Para seleccionar una buena base para el modelo se entrenan las siguientes arquitecturas de CNN: CaffeNet [JSD+14], GoogLeNet [SLJ+15], VGG [SZ14], y Residual Network [HZRS16]. También se prueba el modelo de Deep-face VGG (VGG-FACE) [PVZ15] entrenado solo con imágenes de caras, dando el mejor resultado 70.74, quedando el resto entre 62 y 69.

En el caso de la red C3D establecen dos tipos de imágenes, uno el explicado anteriormente en el párrafo del preprocesamiento y otro el que proporcionado en AFEW6.0 (Face-AFEW). Además se prueba con distintas configuraciones, variando el número de C3D y de CNN-RNN. Los resultados con las imágenes preprocesadas son ligeramente peores que las descritas por Face-AFEW, por ello para mejorar los resultados se decide unir ambas. Los resultados para las diferentes configuraciones se pueden observar en la Tabla 3.4.

Tabla 3.4: Resultados de las distintas configuraciones para la C3D

Método	**Exactitud (%)**
Face-ours (1 C3D)	38.97
Face-ours (1 CNN-RNN + 1 C3D)	42.82
Face-ours + Face-AFEW (2 CNN-RNNs + 2 C3Ds)	48.30

3.4. Detección de Objetos en Vídeos Usando una LSTM Asociativa

La detección de objetos en vídeo es una herramienta fundamental para muchas aplicaciones. Para esta tarea lo mas común es utilizar una LSTM [HS97], pero tienen el problema de que no pueden modelar los objetos entre fotogramas consecutivos. Por ello en [LLT17], los autores proponen una arquitectura de LSTM que si sea capaz de realizarlo, denominada *Association Long Short Term Memory* (ALSTM).

A diferencia de la LSTM [HS97] proponen hacer una regresión directa con las localizaciones y categorías de los objetos, y mientras tanto, producir asociaciones entre las características para representar los distintos objetos detectados. Estas características de asociación son una representación de dichos objetos detectados que capturan la información espacio-temporalmente, además, se optimizan para evitar que dos detecciones estén asociadas con el mismo objeto. Esta mejor asociación mejora el flujo de información a través de los objetos detectados del vídeo, haciendo que la LSTM genere características de mejor calidad, además, el error de regresión del objeto y el error de asociación lo optimizan de forma conjunta.

Hay dos flujos principales de detección de objetos basados en CNN. La mayoría de los métodos utilizan propuestas de bajo nivel para primero generar los cuadrados candidatos, después se clasifica cada región de interés con los modelos de clasificación del estado del arte.

Se tienen en cuenta que en trabajos anteriores se usan RNN con características convolucionales [TLBN16, NZH+17] para realizar tareas de refinamiento, sin embargo, solo concatenan las características de alto nivel con entradas las entradas de la RNN. Por el contrario, proponen implementar la asociación definiendo explícitamente el error de asociación.

La ALSTM recibe como entrada los fotogramas del vídeo, teniendo en cuenta los antiguos. Se usa el detector SSD [LAE+16b] para extraer los cuadros donde se encuentran los objetos. La salida incluye los cuadrados detectados, su nivel de exactitud y además, se indican las asociaciones de las nuevas características. Para acelerar el proceso de entrenamiento, normalizan la salida de cada capa con la normalización *Batch* [CBL+16].

Para la red, dichos autores utilizan un modelo SSD [LAE+16b] de veinte clases preentrenado. Para los resultados utilizan la métrica *Mean Average Precision* (mAP) [Hui18], que calcula el valor de la precisión media para el valor de exhaustividad entre 0 a 1.

Comparan el método propuesto por el conjunto de imágenes de *YouTube-Object* (YO) [PLC+12, TLBN16] con los modelos: VOP [TBHN16], Unsupervised [KCL+15], YOLO [RDGF16] y context [TLBN16]. Ellos proponen tres métodos: El *Baseline_1* que usa solo un SSD [LAE+16b] para detectar los objetos de los fotogramas, sin una regresión en la LSTM [HS97]; el *Baseline_2* añade un post-procesado usando un rastreador basado en puntos clave que refuerza el seguimiento de los objetos a largo plazo y por último la ALSTM. En los resultados se puede observar como el *Baseline_1* otorga peores resultados en las distintas detecciones, mientras que el *Baseline_2* y la ALSTM obtienen resultados mejores que el resto de modelos comparados y más parejos entre si, además la ALSTM es la que consigue valores más elevados con un mAP [Hui18] de 72.14. El resto de valores para las diferentes detecciones se pueden observar en la Tabla 3.5.

Tabla 3.5: Resultados por categoría de la ALSTM sobre el conjunto de datos de YO

Método	mAP (%)	Avión	Pájaro	Barco	Coche	Gato	Vaca	Perro	Caballo	Moto	Tren
Baseline_1	66.21	74.89	85.03	60.11	77.63	61.22	77.56	56.91	80.18	40.67	54.83
Baseline_2	70.43	77.14	91.02	63.34	81.70	63.47	79.38	59.18	83.56	42.33	59.80
ALSTM	72.14	78.92	90.94	65.87	84.76	65.22	81.39	61.86	83.27	43.92	61.25

El conjunto de imágenes de YO [PLC+12, TLBN16] esta compuesto vídeos de 10 subclases diferentes. Contiene 6087 fotogramas anotados, de los cuales, 4306 son para entrenamiento y 1781 son para las pruebas.

A continuación en la Tabla 3.6 se muestran una puesta en común de los proyectos de los trabajos revisados anteriormente. De cada uno de ellos se muestra: La referencia a su respectivo artículo, la arquitectura de la red o el algoritmo de ML que utiliza, el porcentaje de acierto, el conjunto de imágenes que se utilizó y otras características relevantes como pueden ser el número de identidades diferentes utilizadas, si poseen o no preprocesamiento en las imágenes o si añaden alguna técnica a la red.

Tabla 3.6: Puesta en Común de los trabajos del estado del arte

Referencia	Arquitectura	Exactitud (%)	Conjunto de imágenes	Características adicionales
[YRZ+17]	1 CNN y 2 Attention blocks	95.72 ± 0.64	YF [WHM11]	-
		90.44	CB [LZLY14] cerrado	Con 100 Sujetos
		88.76	CB [LZLY14] abierto	Con 100 Sujetos
[PVZ15]	1 CNN profunda	98.95	LFW [HMBLM08]	-
		97.3	YF [WHM11]	Con técnica *triplet loss*
[FLLL16]	1 C3D	38.97	Mezcla de FER [CCG+13],	-
	1 CNN-RNN y 1 C3D	42.82	AFEW [DGLG12, DGJ+16] y	-
	2 CNN-RNNs y 2 C3Ds	48.30	HAPpy PEople Images [DGG15, DGJ+16]	Con Face-AFEW
[LLT17]	Solo SSD sin regresion LSTM	66.21	YO [TLBN16, PLC+12]	-
	Añadiendo postprocesado	70.43		-
	LSTM asociativa	72.14		-

Capítulo 4

Técnica de Detección de Rostros en Vídeos

En este capítulo se explica en profundidad las características de cada una de las partes de las que se compone el trabajo: las tecnologías usadas, la colección de los datos y su preprocesamiento, los modelos propiamente dichos y el proceso que conlleva a la creación de estos y, por último, las métricas usadas durante su experimentación.

El trabajo que se ha realizado ha sido la creación de dos modelos para la detección de rostros en imágenes en movimiento. Dichos modelos están configurados y especializados para dos finalidades: los vídeos reproducidos en local, es decir, con una duración finita y un tamaño definido, y los vídeos en tiempo real, también conocidos como vídeos en *streaming*, como pueden ser las grabaciones en directo de una cámara de seguridad o una cámara web. El trabajo se puede encontrar en el siguiente enlace: `https://gitlab.fdi.ucm.es/tfg2020/facedetection_tfg/`

4.1. Preprocesado de los Datos

Para la generación del modelo, era necesario que las imágenes de los conjuntos tuvieran un formato especial conocido como *TFRecord*. Esto se debe a que se decidió utilizar *TensorFlow* por las ventajas que suponía su uso, como se detalla en la Sección 4.4.2. Los ficheros *TFRecords* no son más que un archivo binario, el cual es leído secuencialmente por *TensorFlow* para entrenar un modelo de DL.

Debido a la dificultad para encontrar conjuntos de imágenes con sus datos en este formato, se optó por investigar cual era el procedimiento para convertir archivos de etiquetado más comunes en dicho formato. La solución final fue transformar las etiquetas del conjunto almacenadas en *eXtensible Markup Language* (XML) a *TFRecord*, transformándolas a formato *Comma-Separated Values* (CSV) como proceso intermedio. Para uno de los conjuntos de imágenes, esta solución se realizó manualmente para cada una de las muestras y para el otro se aplicó el mismo proceso pero de manera automática. Cada caso se explica con más detalle en su sección correspondiente. Los conjuntos de imágenes utilizados son:

- **Conjunto de imágenes I**: Labeled Faces in the Wild [HMBLM08].
- **Conjunto de imágenes II**: Wider Face Dataset [YLLT16].

4.1.1. Conjunto de imágenes I

LFW [HMBLM08] es un conjunto de imágenes publico diseñado para verificar si una pareja de imágenes pertenecen a la misma identidad. Está compuesto por 13233 imágenes de 250x250 de caras almacenadas en 5749 carpetas con sus identidades.

A pesar de que el propósito de este trabajo no era el de la identificación sino el de la detección, y aunque el propio servicio web que ofrece acceso al repositorio de imágenes no recomendaba el uso de este para tareas de detección por la escasez de variedad en lo que a edad de los sujetos presentes en las imágenes se refiere, el se decidió usarlo debido a, no solo los buenos resultados que se pueden observar en [PVZ15], sino también al tamaño del conjunto, dado que este hecho daba la oportunidad de transformar las imágenes al formato requerido por *TensorFlow* sin invertir mucho tiempo en ello.

El proceso de adaptación del conjunto de imágenes sigue los siguientes pasos:

1. El primer paso consistió en una investigación acerca de cómo crear los *TFRecords* que necesitaba *TensorFlow* a partir de un conjunto de imágenes sin etiquetar. Tras la investigación, se encontró un repositorio con un conjunto de imágenes etiquetadas que estaba compuesto por dos *scripts* en *Python*. El primero se encargaba de convertir las etiquetas de formato XML a CSV y el segundo transformaba de formato CSV a *TFRecord*. Modificando estos dos *scripts* y cambiando algunas configuraciones para adaptarlo a lo que se requería, el equipó encontró el método para realizar la transformación con imágenes etiquetadas.

2. El siguiente paso que se realizó fue el de etiquetado de las imágenes. La herramienta que permitió realizar esta tarea y generar imágenes etiquetadas en formato XML fue *labelimg* [lab17]. Se decidió usar esta herramienta ya que, además de contar con interfaz gráfica, su configuración permitía establecer una etiqueta por defecto, de tal forma que al seleccionar la región deseada, esta se auto-etiquetara, acelerando el proceso. Se realizó un script de Batch que centralizaba las imágenes en un mismo directorio para posteriormente importarlo a *labelimg* y permitir así la navegación entre las imágenes. Una vez dentro de la aplicación, se seleccionaban la o las caras que estaban contenidas en cada imagen, se etiquetaban con la etiqueta por defecto mencionada anteriormente y, por último, se guardaban, de tal forma que la aplicación generaba el archivo XML con la información del etiquetado, necesaria para su posterior conversión.

3. Por último, se aplicaron distintos procedimientos para transformar las imágenes etiquetadas en el anterior paso al formato de *TFRecord* y para separar todas imágenes en los subconjuntos de pruebas y validación. Para realizar dicha labor, primero se agruparon todos los archivos XML generados en el anterior paso en los directorios mencionados anteriormente y se usó el procedimiento comentado en el primer paso para generar los *TFRecords* correspondientes.

4.1.2. Conjunto de imágenes II

Wider Face [YLLT16] es un conjunto de imágenes diseñado para la detección de rostros. Utiliza las imágenes publicas del conjunto de *WIDER* [XZLT15] que contienen rostros en distintas distancias, poses, con partes tapadas o pintadas, con distintas expresiones y con distintos niveles de luminosidad.

En total se tienen 32203 imágenes con 393703 rostros etiquetados, de los cuales un 40 % son para entrenamiento (∼12876 imágenes), un 50 % para pruebas (∼16101 imágenes) y un 10 % para validación (∼3226 imágenes).

A la hora de buscar como pasar este conjunto de imágenes al requerido por *Tensorflow*, se encontró en un repositorio un *script* para realizar esta tarea sobre este conjunto en específico [wid20] de manera automática. Gracias a esto, solo fue necesario instalar dicho repositorio y ejecutar, para obtener así los *TFRecord* necesarios.

4.2. Creación de los Modelos de Detección

Se han creado dos modelos para realizar la detección de los rostros, una para cada finalidad de uso, explicadas anteriormente.

El primer modelo ha sido optimizado para la detección de rostros en vídeos de manera local, es decir, con vídeos con duración limitada. Se trata, por tanto, de una red más enfocada en la precisión que en la velocidad de detección. Se decidió diseñar esta red puesto que para realizar la detección en un vídeo, es necesario un procesamiento previo completo de este antes de obtener los resultados finales, por lo que no es demasiado relevante, en este caso, la velocidad de detección del modelo por fotograma.

Por otro lado, en este trabajo se ha desarrollado una segunda opción optimizada para su uso en vídeos en tiempo real, conocidos como vídeos en *streaming*. Dado que para el procesamiento de un vídeo en tiempo real es crucial el tiempo en el que se realiza dicha detección, se ha diseñado este modelo con un enfoque mucho más directo en la velocidad de procesamiento de fotogramas que la anterior, manteniendo también un buen rendimiento en cuanto a precisión se refiere.

Ambos modelos han demostrado durante la fase de experimentación ser capaces de detectar varios rostros simultáneamente y en distintas condiciones, tales como rostros a distintas distancias, poses poco comunes, expresiones variopintas, condiciones de luz desfavorables, e incluso rostros maquillados y/o tapados con distintos atuendos, como mascarillas.

Para cada una de los modelos, se explicará de qué están formados, cuál es el proceso que siguen para realizar la detección y la configuración que se ha utilizado para poder conseguir el objetivo.

4.2.1. Modelo para la Detección en Vídeos Reproducidos Localmente

Dado que el objetivo con este modelo es asegurar la efectividad frente a la velocidad de detección, se ha utilizado la combinación formada por un detector *Faster R-CNN* y el extractor de características *Inception-ResNet*.

La arquitectura interna del primer modelo desarrollado consta de tres partes bien diferenciadas, como se puede ver en la Figura 2.14: La extracción de características, la propuesta de rostros y por último una capa donde se agrupan esas propuestas y se realiza la predicción.

La primera hace referencia a la CNN que se encarga de la extracción de características. En este caso se ha usado la red neuronal llamada *Inception-ResNet*, desarrollada por Google, cuya estructura interna puede verse en la Figura 4.1.

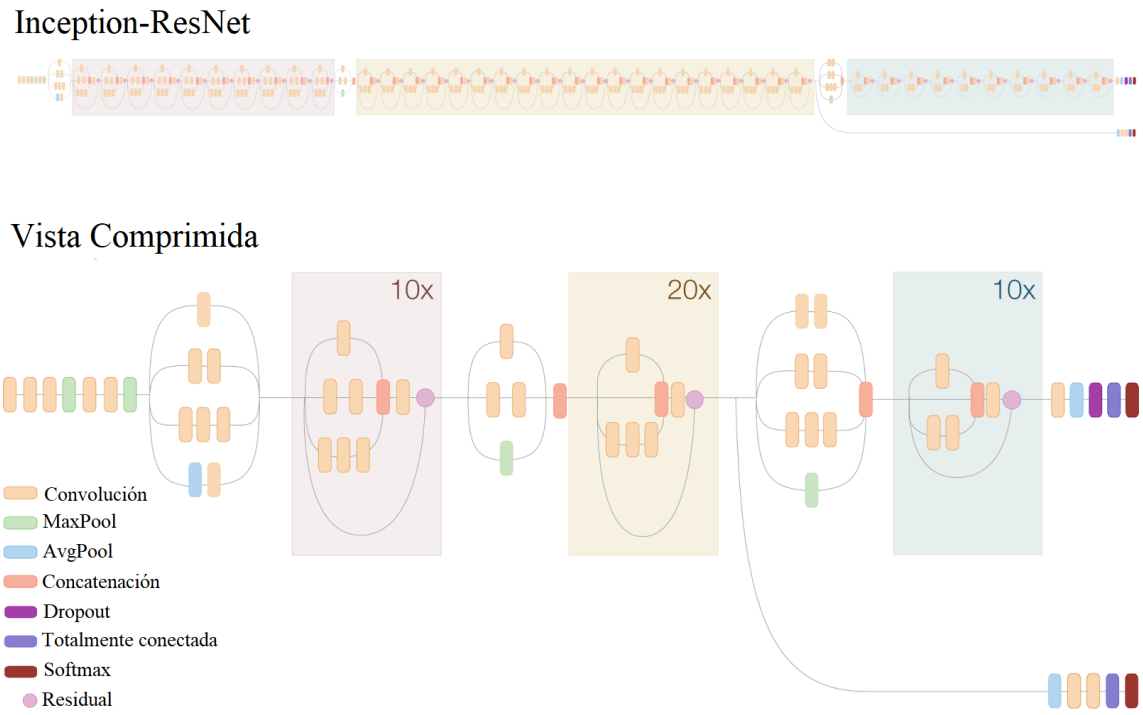


Figura 4.1: Arquitectura Interna de *Inception-ResNet*

Como se observa en la Figura 4.1, se ha procedido a mostrar su versión completa y otra más comprimida de la arquitectura de la red para entender con más facilidad cómo esta hecha. Como se puede observar, la red está creada por una gran cantidad de capas simbolizadas con distintas formas de colores, en las que se incluyen capas de convolución (explicadas con más detalle en la Sección 2.5.1.1), capas de *pooling* (Sección 2.5.1.2), capas de agrupación o concatenación, capas de *dropout* encargadas de apagar una serie de neuronas para evitar el sobre ajuste, capas totalmente conectadas, capas residuales propias de la red *ResNet* y, por último, las funciones de activación *softmax*, similares a las vistas en la Sección 2.4.

Todas ellas forman estructuras que pueden ser susceptibles a repetirse en tandas de 10 y 20, como se puede apreciar en la imagen.

El segundo paso es realizar la propuesta de regiones que puedan ser los rostros. En esta segunda fase, se recibe las características en forma de mapa extraídas con la red anterior y se procesan usando una red neuronal especial capaz de generar regiones, como se puede ver en la Figura 4.2.

Esta red, además, clasifica cada región para saber si lo que ha propuesto pertenece al fondo de la imagen o es una cara en cuestión.

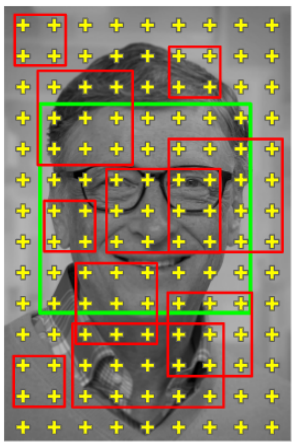


Figura 4.2: Ejemplo más visual de las regiones propuestas

Las propuestas resultantes que no pertenecen al fondo de la imagen, finalmente se propagan a través de una capa de agrupación como las que contiene *Inception-ResNet* para que posteriormente se clasifiquen cada una de ellas y se obtenga una probabilidad que indique cuán semejante es la propuesta a un rostro.

Por último, aquellas propuestas que superen un cierto umbral, el cual se explica en la Sección 4.3, serán consideradas detecciones y se acabarán dibujando los límites que recuadren la cara en la imagen o fotograma del vídeo en cuestión.

Una vez conseguido el correcto funcionamiento del detector, para poder ser usado con vídeos, era necesario tener el modelo exportado de forma que pudiera admitir imágenes y devolver como resultado las coordenadas del rostro dentro de estas.

Para procesar el vídeo, únicamente habría que realizar un bucle donde se leyera fotograma a fotograma y, para cada uno, se utilizase este modelo para inferir el lugar donde se encuentran las caras. Como ya se ha dicho, una vez hecho esto, únicamente sería necesario dibujar los limites de esta en la cara e ir creando un vídeo nuevo con ello.

4.2.2. Modelo para la Detección en Vídeos en Tiempo Real

El objetivo que debe cumplirse con este modelo debe ser reforzar la velocidad de inferencia y pasar a un segundo plano la efectividad, sin descuidar esta en exceso. Es por ello, que se ha usado una combinación del detector *Faster R-CNN* con el extractor *Inception*

Este segundo modelo creado está basado en la CNN *Inception*, desarrollada por Google. Debido a que su estructura y proceso es similar al explicado en la Sección anterior, únicamente se detallará la arquitectura interna que esta CNN sigue para poder extraer las características. Su estructura interna es similar a la que se muestra en la Figura 4.3.

Al igual que se podía observar en el modelo propuesto para la detección en vídeos, *Inception* también dispone de una amplia variedad de capas, las mismas que se podían encontrar en la Sección anterior. La diferencia entre estas dos redes reside en que la primera contiene una serie de capas residuales pertenecientes a *ResNet* y la profundidad de la red es considerablemente mayor, mientras que *Inception* no cuenta con estas características.

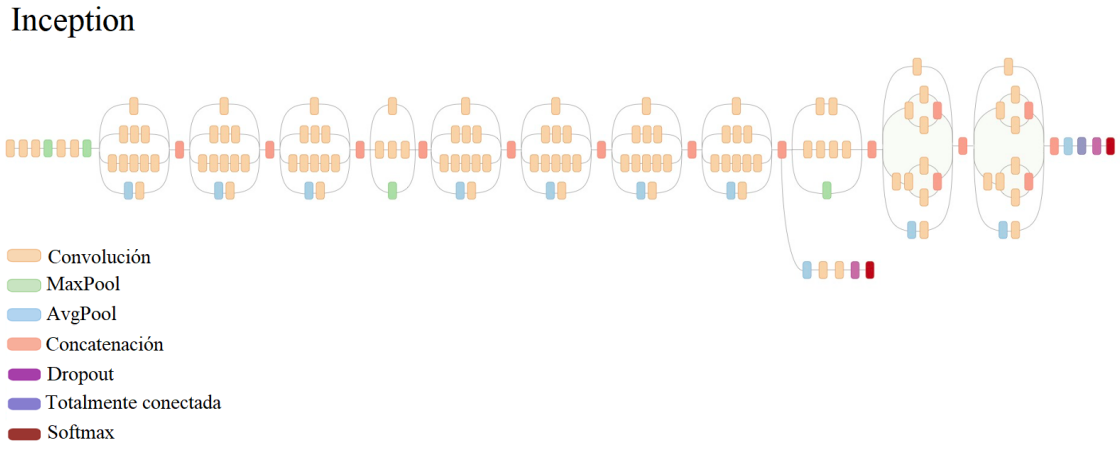


Figura 4.3: Arquitectura Interna de *Inception*

Esta última característica es la que hace que sea la apropiada para usarse en entornos donde la velocidad del procesamiento es clave. Al no haber tantas capas, la cantidad de neuronas por capa disminuye y, en consecuencia, el número de cálculos matemáticos se ve reducido considerablemente.

El proceso de la detección en tiempo real se realiza de la misma forma que en vídeos. Para ello, es necesario utilizar las librerías *Pafy* y *Streamlink* para poder obtener los fotogramas de la fuente en tiempo real. La diferencia respecto a la detección en vídeo local reside en mostrar la salida por pantalla o redirigir la salida a un canal donde se requiera visualizar los resultados a medida que estos se generan, en lugar de crear un vídeo para ser observado con posterioridad.

4.2.3. Configuración de los Modelos

Para poder ejecutar los modelos en cuestión, *TensorFlow* ofrece una serie de ficheros que deben ser configurados de forma que en este se incluya el tipo de detector deseado, el extractor de características, la configuración del entrenamiento y validación, y toda una serie de parámetros asociados a cada cada uno de estos campos para optimizar la detección. Todas estas opciones de configuración hacen referencia en realidad a funciones, clases y parámetros pertenecientes a *TensorFlow* que, en ultima instancia, son los que finalmente serán ejecutados. En general, la estructura que debe seguir este fichero, tiene que seguir unas pautas rigurosas diferenciando de esta forma entre cinco partes que van a permitir realizar la configuración. Estas partes son las siguientes:

- **Model**: Define el tipo de modelo que va a ser utilizado, es decir: el detector, la CNN y los parámetros que definen a la arquitectura en general.
- **Train Config**: En él se indican los parámetros que deben usarse para realizar el entrenamiento, como puede ser un preprocesamiento en la entrada a la red o los valores de inicialización de la CNN.
- **Train Input Reader**: Utilizado para indicar los datos de entrenamiento en formato TFRecord.

- **Eval Config**: Permite ajustar las métricas que van a ser monitorizadas durante el entrenamiento.
- **Eval Input Reader**: Se definen el conjunto de datos de validación a usar. Debe ser distinto al de entrenamiento.

Los aspectos más relevantes para la creación del modelo son: Configuración del detector, Configuración del entrenamiento y Configuración de la evaluación.

Dado que los modelos propuestos en este capítulo se basan ambos en el detector *Faster R-CNN* y tienen parte de la configuración en común, se explicarán los aspectos más relevantes que se han tenido en cuenta durante la configuración de estos, y se detallarán las particularidades que tienen cada uno de ellos.

4.2.3.1. Configuración del Detector

El primer paso para la configuración del modelo es la declaración del detector a usar, en este caso se trata de *Faster R-CNN*. A continuación se define el número de objetos que se van a detectar en el campo *num_classes*, que en este caso el valor debe ser 1 ya que solo se van a detectar rostros.

A continuación, se configuran los tres componentes del detector. (ver Secciones 2.7.1.3 y 4.2.1).

1. ***feature_extractor***: Hace referencia a la CNN que se utilizará. Entre otros campos que aparecen en esta estructura, se encuentra el campo *type* que identifica el tipo de red (*Inception-ResNet*, para el caso de vídeos en local, o *Inception* en el caso de *streaming*). El valor depende del modelo que se quiera crear.
2. ***first_stage***: Es la etapa en la que se realizan propuestas sobre las características que tenía como salida la CNN. Esta etapa queda referenciada por los parámetros que comienzan por *first_stage*. En esta etapa se configuran los 3 parámetros: *first_stage_nms_score_threshold*, *first_stage_nms_iou_threshold* y *first_stage_max_proposals*. Los dos primeros utilizan supresión no máxima (del inglés, *Non-maximum Suppression*) para evitar muchas propuestas sobre un mismo rostro.
    - ***first_stage_nms_score_threshold***: Se utiliza para filtrar las propuestas que se hacen en esta etapa. Al establecerse en 0.0 se indica que todas las propuestas que se hagan van a ser válidas, ya que no van a tener que superar ningún umbral de confianza. De la misma forma se tiene el parámetro *first_stage_nms_iou_threshold*, que al ajustarse en 0.69, se considerará dos propuestas similares cuando estén sobre este umbral de Intersección sobre Unión (del inglés, *Intersection over Union* (IOU)), quedándose con la que tenga una puntuación mayor y eliminando la otra. En la Sección 4.3 se explican más ampliamente los umbrales que están involucrados en estos parámetros.
    - ***first_stage_max_proposals***: Por convención tiene los valores que aparecen en el código: 300 y 100, para *Inception-ResNet* e *Inception* respectivamente. Este es uno de los motivos que hace que *Inception-ResNet* sea más precisa y lenta a la hora de inferir y ser entrenada. Al realizar un mayor número de propuestas sobre una imagen, su rendimiento puede aumentar pero el tiempo para realizar la detección también aumentará.

3. ***second_stage***: Se encarga de recibir las propuestas que genera la primera etapa y clasificarlas para generar un resultado finalmente. Al igual que en la primera etapa, se establece una puntuación (*score_threshold*) y un valor de IOU (*iou_threshold*) por los cuales las propuestas serán consideradas detecciones. También se establece el número máximo de detecciones por clase que se van a realizar (*max_detections_per_class*), el cual coincide con el número total de detecciones (*max_total_detections*) al haber una sola clase. Para finalizar, se modificó *score_converter* poniendo la función de activación *Sigmoide* (Sección 2.4), ya que funciona mejor que *Softmax* para problemas donde solo hay que clasificar un elemento.

El Código 4.1 muestra la configuración realizada.

Configuración 4.1: Configuración tenida en cuenta en Model

```
model {
   faster_rcnn {
       num_classes: 1

       feature_extractor {
         type: "faster_rcnn_inception_resnet_v2" | "faster_rcnn_inception_v2"
       }

       first_stage_nms_score_threshold: 0.0
       first_stage_nms_iou_threshold: 0.69
       first_stage_max_proposals: 300 | 100

       second_stage_post_processing {
         batch_non_max_suppression {
           score_threshold: 0.30
           iou_threshold: 0.60
           max_detections_per_class: 100
           max_total_detections: 100
         }
         score_converter: SIGMOID
       }
   }
}
```

4.2.3.2. Configuración del Entrenamiento

Como se ha explicado previamente, el entrenamiento necesita ser configurado también y, concretamente, consta de dos partes: *train_config* y *train_input_reader*, como se puede ver en el Código 4.2.

Dentro de *train_config*, se encuentran los siguientes parámetros:

1. ***batch_size***: Indica el tamaño de los paquetes de datos que irá utilizando el algoritmo para entrenar al detector. En este caso, se ha establecido en 1, lo cual significa que irá propagando las imágenes sobre las redes de una en una.

2. ***data_augmentation_options***: Hace referencia a la técnica aplicada en DL que permite aumentar sustancialmente el conjunto de entrenamiento mediante la modificación de las imágenes que ya están contenidas en él. En concreto, se ha utilizado *random_horizontal_flip* que realiza un giro horizontal y aleatorio sobre las imágenes, pero se podría haber realizado cualquier otra modificación.

Configuración 4.2: Configuración tenida en cuenta para el entrenamiento

```
train_config {
    batch_size: 1

    data_augmentation_options {
        random_horizontal_flip {
        }
    }

    num_steps: 200000

    from_detection_checkpoint: true
    fine_tune_checkpoint: "ruta_al_modelo_ya_entrenado"
}


train_input_reader {
    label_map_path: "ruta_a_las_etiquetas_del_dataset"

    tf_record_input_reader {
        input_path: "ruta_al_tfrecord_de_entrenamiento"
    }
}
```

3. ***num_steps***: Permite ajustar el número de pasos de ejecución para el entrenamiento. *TensorFlow* asegura que con 200000 se puede encontrar un buen ajuste del modelo y se estableció de esa forma.

4. ***from_detection_checkpoint* y *fine_tune_checkpoint***: Quizás sean de los más importantes que hay en el fichero, ya que son los que ofrecen la opción de poder realizar la técnica de TL. Para abordar esta técnica, los extractores de características que se han utilizado en el presente proyecto han sido entrenados previamente con una serie de imágenes que abarcan casi cualquier tipo de objeto. Esto quiere decir que las redes están preparadas para clasificar correctamente entre los distintos objetos con los que ha sido entrenado y, sin embargo, no son capaces de clasificar un rostro cuando se quiera utilizar la CNN para este cometido. Es por ello que es estrictamente necesario capacitar a la red para poder realizar esta tarea. Mediante estos parámetros, *TensorFlow* ofrece la posibilidad de indicar la ruta donde se encuentra el modelo que ya ha sido entrenado y cuyo aprendizaje se requiere reutilizar para proporcionar un mejor rendimiento al nuevo modelo que se esté creando. Internamente, *TensorFlow* realiza la técnica de *Fine-Tune*, explicada en la Sección 2.2.3.

Por último, dentro del campo *train_input_reader*, se encuentran los siguientes parámetros:

1. ***label_map_path***: Hace referencia a la ruta de un fichero que contiene todas las etiquetas que se utilizarán en la detección. Para poder realizar TL, se ha modificado este archivo añadiendo una etiqueta (*Face*) para los rostros, y eliminando cualquier otra etiqueta que pudiera encontrarse en el modelo preentrenado.

2. ***input_path***: En él se debe indicar la ruta a los *TFRecord* del conjunto de datos que requeridos para entrenar el modelo.

Con estos últimos cuatro parámetros se estaría implementando la técnica de TL. Al establecer: una sola etiqueta posible (*Face*), un conjunto de datos de un solo tipo, el parámetro *num_classes* del modelo a 1, y permitir a la CNN adquirir el aprendizaje de otro entrenamiento previo; lo que ocurre es que el detector vuelve a realizar el entrenamiento con la mayoría de las capas de la red ya ajustadas, pero esta vez se fuerza a *TensorFlow* a que modifique internamente la red de tal forma que, ahora, solo va a ser capaz de detectar rostros.

Por esa razón, aunque se utilice el ajuste de las capas que se consiguió durante el aprendizaje de otros objetos, el nuevo detector solo detecta caras. Esta técnica permite poder entrenar de una forma más rápida y eficiente, puesto que en las capas iniciales de la red siempre se aprenden una serie de características que son comunes a todos los objetos.

4.2.3.3. Configuración de la Evaluación

La configuración más relevante en el proceso de evaluación. En el fragmento de Código 4.3, se observa que en el campo *eval_config* se ha establecido el número de datos de validación a 3226 en el parámetro *num_examples*, que en este caso es el número de imágenes destinadas a la validación del conjunto de imágenes Wider Face. Además, al igual que ocurría en el caso del entrenamiento, hay que indicar la ruta de las etiquetas que se van a usar para la detección (*label_map_path*) y la ruta al conjunto de datos que utilizará para realizar la evaluación (*input_path*)

Configuración 4.3: Configuración tenida en cuenta para la validación

```
48 eval_config {
49    num_examples: 3226
50 }
51
52
53 eval_input_reader {
54  label_map_path: "ruta_a_las_etiquetas_del_dataset"
55
56  tf_record_input_reader {
57    input_path: "ruta_al_tfrecord_de_validacion"
58  }
59 }
```

4.3. Métricas de Evaluación en la Detección Facial

En el campo del ML se vuelve necesario el hecho de poder medir cuál es el rendimiento de los modelos entrenados, para poder compararlos entre sí de una manera objetiva y conocer cuál de ellos es el que mejor se puede ajustar.

Más concretamente, en la rama de la detección facial, el conjunto de métricas que se suelen utilizar se basan en las conocidas competiciones de PASCAL VOC [pas20], COCO [LMB+20] y Open Images [ope20b], que definen una serie de pautas para evaluar el algoritmo propuesto con los respectivos conjuntos de datos de validación.

Estas competiciones tienen algo en común, y es que todas tienen la precisión media como medida principal. Sin embargo, para la evaluación del rendimiento de los modelos creados en este proyecto, se ha escogido el conjunto de métricas definidos por COCO, debido a que es el que mayor número de métricas ofrece, dando incluso la oportunidad de poder obtener el rendimiento basado en la lejanía del rostro en una imagen.

A continuación, se describen dichas métricas y se adicionan otras que no pertenecen a COCO y que se han utilizado debido a su gran importancia en problemas de ML.

4.3.1. Conjunto de Métricas Usadas

Antes de explicar las métricas se utilizan en COCO, es necesario conocer el marco en el que se basan y de qué se encargan cada una.

- **Intersección sobre Unión**: Como se ha comentado previamente, en la detección facial, no solo hay que preocuparse de evaluar la probabilidad de que un rostro en concreto aparezca en la imagen, sino que también se debe ser capaz de localizarla. Esto supone un problema, ya que las métricas comunes utilizadas sobre modelos de clasificación deben ser extendidos. Aquí es donde entra en acción la IOU. La IOU es una manera de cuantificar la similitud que existe entre el cuadro delimitador predicho por el modelo y el original que va asociado con los datos de entrada. Su umbral puede variar de 0 a 1, siendo más alto cuanto más superpuestos estén entre los dos. Se puede ver un ejemplo en la Figura 4.4.

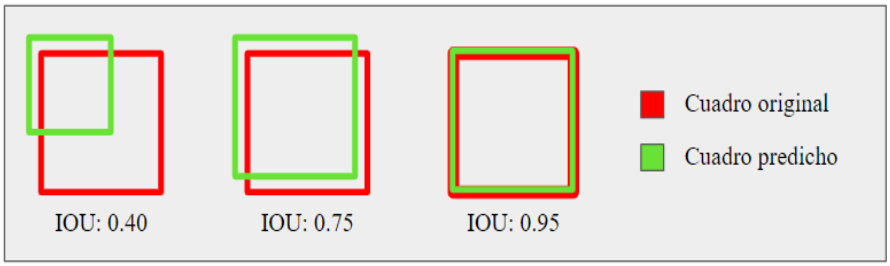


Figura 4.4: Ejemplo de la Intersección sobre Unión

- **Puntuación de Confianza**: Se refiere a la probabilidad con la que se clasifica una cierta imagen y representa el porcentaje de seguridad por el que una clase determinada ha sido detectada.
- **Umbral de Confianza**: Es el porcentaje que indica a partir de qué valor de Puntuación de Confianza se va a considerar una predicción como una detección. Si se establece, por ejemplo, un umbral del 30 %, todas las predicciones que no superen esa puntuación, no serán consideradas una detección.
- **Tipos de Predicciones**: Las predicciones se pueden clasificar en los siguientes puntos:

- **Verdadero Positivo**: Se produce cuando hay una cara y el modelo la detecta perfectamente. Sin embargo, para que se considere como tal deben cumplirse tres condiciones: La puntuación de confianza tiene que ser mayor que el umbral de confianza, la clase predicha tiene que ser la misma que la del objeto detectado y el cuadro limitador predicho debe ser mayor que el umbral de IOU.
- **Falso Positivo**: Se produce cuando en la realidad no hay ninguna cara pero el modelo detecta que la hay. Cualquier violación de las condiciones anteriores supondría un Falso Positivo.
- **Falso Negativo**: Se produce cuando hay una cara en la imagen y el modelo no es capaz de detectarla. Supone que la puntuación de confianza es más baja que el umbral y, por tanto, no se ha considerado la detección.
- **Verdadero Negativo**: Se produce cuando no hay una cara en la imagen y el modelo no la detecta. En nuestro caso los Verdaderos Negativos no importan ya que solo se tiene una clase a detectar, por tanto, hay 3 opciones solo: haber una cara y detectarla (VP), haber una cara y fallar al detectarla (FP) y no detectar una cara y en realidad haber una (FN). No se puede no detectar una cara y que no la haya, porque siempre la va a haber (sólo hay una clase).

Los distintos tipos de predicciones quedan entonces resumidos en la Figura 4.5

Figura 4.5: Tipos de Predicciones

Para contabilizar los distintos tipos de predicciones que realiza el modelo, se han generado las correspondientes Matrices de Confusión [Val20], que permiten contabilizar de forma esquemática el rendimiento del algoritmo. En concreto, las Matrices generadas siguen el formato de la Tabla 4.1.

Tabla 4.1: Esquema de las Matrices de Confusión generadas

	Detección: Sí	**Detección: No**
Realidad: Sí	VP	FN
Realidad: No	FP	VN

- **Precisión**: Es la probabilidad de que un cuadro limitador original coincida con uno que ha sido predicho. Su valor varía de 0 a 1, por lo que si se dispone de una precisión de 0.8 indicaría que el 80 % de las veces los objetos detectados coinciden con los originales. La precisión se consigue mediante la fórmula:

$$\text{Precisión} = \frac{VP}{VP + FP} = \frac{\text{Detectados y Acertados}}{\text{Todas los cuadros detectados}} \tag{4.1}$$

- **Exhaustividad**: Se encarga de medir la probabilidad con la que los objetos originales sean detectados correctamente, es decir, podría considerarse una tasa para los VP.

$$Exhaustividad = \frac{VP}{VP + FN} = \frac{\text{Detectados y Acertados}}{\text{Todas los cuadros originales}} \tag{4.2}$$

- **Exactitud**: Es una de las métricas más conocidas en ML y define el porcentaje de acierto de todas las predicciones realizadas.

$$Exactitud = \frac{VP + VN}{VP + FP + VN + FN} = \frac{\text{Acertados}}{\text{Todas las detecciones}} \tag{4.3}$$

- **Tiempo de Entrenamiento**: Es el tiempo que se ha tardado en entrenar el modelo. Se diferenciarán los tiempos dependiendo de donde se hayan ejecutado. Concretamente, se obtendrán el tiempo de entrenamiento en local y en cloud.

- **Tiempo de Inferencia o Detección**: Es el tiempo que tarda el modelo en realizar una detección. Se medirá en segundos y esto permitirá conocer si podría usarse para detecciones de rostros en tiempo real o en *streaming*.

- **Fotogramas Por Segundo**: Del inglés, *Frames Per Second* (FPS). Es el número de imágenes sobre las que el modelo puede detectar caras en un segundo. Cuanto mayor sea esta medida, más útil será para aplicaciones en tiempo real, puesto que la detección funcionará de forma más fluida.

4.3.1.1. Métricas de COCO

COCO cuenta con 12 métricas distintas que ayudan a medir de manera exhaustiva el rendimiento de un detector. Las métricas son las siguientes:

- **Precisión Media**: Denotado indistintamente por COCO como mAP o AP. Hace referencia a la media de las precisiones sobre todas las clases que tiene que detectar un modelo. El modelo que se ha creado solo detecta una, por lo que se considera como la precisión previamente explicada. A diferencia de otras competiciones donde la precisión se computa solo con un IOU de 0.5, en COCO, se realiza la media sobre distintos umbrales de IOU, concretamente de 0.5 hasta 0.95 con incrementos de 0.05. Esto hace que se penalice al modelo por realizar una localización deficiente y se optimiza para que lo haga mejor. Por tanto, una mAP indicaría también que los rostros están perfectamente localizados.

- **Precisión Media Sobre un Umbral de 0.5 y 0.75 IOU**: Denotado comúnmente como mAP@.50 IOU y mAP@.75 IOU respectivamente. Hace referencia a lo comentado en el punto anterior pero para los dos umbrales más usados en otras competiciones.

- **Precisión Media en Objetos Pequeños, Medianos y Grandes**: Se hará referencia a esta métrica como mAP *Small*, mAP *Medium* y mAP *Large* respectivamente. Va a permitir obtener la precisión dependiendo de la distancia a la que se encuentren las caras. Concretamente, caras que tengan un área menor de 32^2 píxeles se considerarán pequeñas, un área entre 32^2 y 96^2 píxeles serán medianas y un área mayor de 96^2 píxeles serán grandes.
- **Exhaustividad Media Sobre 1, 10 y 100 Detecciones**: La exhaustividad media funciona igual que la precisión y realiza la media sobre distintos valores de IOU. Además, la variante que usa COCO se basa en diferenciar esta, teniendo en cuenta el número de detecciones también. Se hará referencia a esta métrica como AR@1, AR@10 y AR@100 respectivamente.
- **Exhaustividad Media en Objetos Pequeños, Medianos y Grandes**: Esta métrica se realiza sobre 100 detecciones y se denotará como AR Small, AR Medium y AR Large respectivamente. Va a permitir obtener la exhaustividad dependiendo de la distancia a la que se encuentren las caras. Concretamente, caras que tengan un área menor de 32^2 píxeles se considerarán pequeñas, un área entre 32^2 y 96^2 píxeles serán medianas y un área mayor de 96^2 píxeles serán grandes.

4.4. Tecnologías

Durante el desarrollo del proyecto se han utilizado diversas tecnologías entre las que se incluyen librerías y algunas herramientas y plataformas que han permitido acelerar y entender mejor el proceso de creación de los modelos propuestos. A continuación se explicarán cada una de ellas.

4.4.1. Librerías

El lenguaje de programación utilizado para realizar la detección ha sido *Python*. Las librerías utilizadas para conseguir el objetivo abarcan desde la computación numérica hasta la lectura y/o extracción de contenido visual, pasando por algunas más especializadas en el campo de la visión por computador. A continuación, se explicarán aquellas que se han utilizado para desarrollar el proyecto.

- **NumPy [num20]:** Es una librería de código abierto cuyo objetivo es habilitar la computación numérica con Python. Agrega mayor soporte para vectores y matrices, proporcionando un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en matrices, que incluyen matemática, lógica, manipulación de formas, clasificación, selección, E/S , transformadas discretas de *Fourier*, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

  En lo que se refiere a este trabajo, se han usado para añadir y quitar dimensiones al *array* de datos de la imagen, donde se almacenan y extraen los parámetros de las detecciones.

- ***OpenCV*** **[ope20a]:** Es una librería de software libre desarrollada por *Intel*, dedicada a la CV. Es una de las librerías más populares en el ámbito de la visión artificial debido a su amplia funcionalidad, permite procesar imágenes, leer y escribir imágenes y vídeos, ofrece una interfaz gráfica de alto nivel, ofrece análisis de vídeo, calibración y reconstrucción 3D de objetos, reconocimiento de objetos, un marco de características 2D, agrupación y búsqueda en espacios multidimensionales, fotografía computacional, fusión de imágenes (*stitching*) y una API para gráficos. También posee módulos dedicados al ML y DL.

  En este trabajo se ha hecho uso de su potencial a la hora de procesar imágenes y vídeos para obtener uno a uno los fotogramas de un vídeo para analizarlos y posteriormente ir generando un vídeo como resultado con las caras detectadas. Además, permite la previsualización del vídeo en cuestión mientras se procesa.

- ***Pafy*** **[paf20]:** Es una librería de *Python* para descargar contenido de *YouTube*, vídeos o listas de reproducción y sus metadatos, como las visitas, duración, autor, puntuación, etc. Permite seleccionar la resolución, el *bitrate* (frecuencia de bits) y el formato del vídeo o incluso de emisiones en directo. Esta librería se ha usado para obtener dichos vídeos o emisiones y usarlos como entrada para *OpenCV* para poder procesar vídeos subidos a esta plataforma usando su *Uniform Resource Locator* (URL).

- ***Streamlink*** **[str20]:** Es una librería con interfaz por línea de comandos compatible con *Python* que permite extraer emisiones en directo de varios servicios a un reproductor de vídeo. Su propósito principal es evitar el uso de sitios web con muchos recursos y sin optimizar, permitiendo al usuario disfrutar del contenido sin distracciones. Es compatible con la mayoría de servicios de transmisión, los más famosos son *Twitch*, *YouTube*, *Livestream* y *Dailymotion*. Se he hecho uso de esta librería para ampliar la funcionalidad de *Pafy* con otros servicios de transmisión, ya que este se limitaba solo a *YouTube*.

4.4.2. Herramientas y Plataformas

Además de librerías, se han optado por algunas herramientas y plataformas que ayudaban a la hora de realizar la detección y entrenar los modelos con una mayor velocidad. Dichas herramientas son: *TensorFlow Object Detection API*, *TensorBoard* y *Google Cloud Platform*. A continuación se explicarán cada una de ellas.

- ***TensorFlow Object Detection API*:** A día de hoy, crear modelos precisos de detección de objetos sigue siendo una tarea desafiante en el campo de la visión por computador. Es por ello que *TensorFlow* ha creado la API *Object Detection*, el cual es un marco de código abierto que facilita construir e implementar modelos de detección de objetos [ten20]. Con esta API, se facilita en gran cantidad el proceso de creación del detector al tener: un fichero con las etiquetas de los objetos a detectar, un conjunto de datos en formato *TFRecord* (propio de *TensorFlow*) y un fichero de configuración que modele la arquitectura a desarrollar. Además de ello, permite realizar la famosa técnica de TL de una manera muy intuitiva. Pese a haber probado otras tecnologías como *Keras* durante el desarrollo de este trabajo, se ha llegado a la conclusión de que esta API es la que mejor se adaptaba a las necesidades del equipo de trabajo ya que, aunque *Keras* ofrece la posibilidad de crear redes neuronales

que puedan servir de extractor de características, dificulta mucho la creación de un detector como tal, al ser una librería en la que es necesario crear las redes neuronales capa a capa.

- ***TensorBoard*:** Es una herramienta ofrecida por *TensorFlow* que proporciona herramientas de visualización y gestión de los modelos. Dicha herramienta es usada para examinar desde una perspectiva más interna el comportamiento de estos. Con *TensorBoard* se pueden observar las curvas de aprendizaje, así como distintos histogramas que representan la evolución del modelo a lo largo del proceso de entrenamiento de este. También se pueden observar distintas métricas, como algunos ejemplos de salida del modelo usando el conjunto de imágenes de validación. En lo que se refiere a este trabajo, *TensorBoard* ofrecía algunas detecciones durante el mismo proceso de entrenamiento. Las métricas obtenidas pueden ser analizadas con todo detalle, como en el caso de las gráficas de las curvas de aprendizajes, que pueden ser suavizadas y cambiadas para observar mejor los datos. En este trabajo, se ha hecho uso de *TensorBoard* para extraer todas las métricas posibles tras el entrenamiento de cada modelo, y poder así contrastar la información de cada uno con el objetivo de elegir el mejor de todos. En la Figura 4.6, se puede ver la interfaz de esta herramienta.

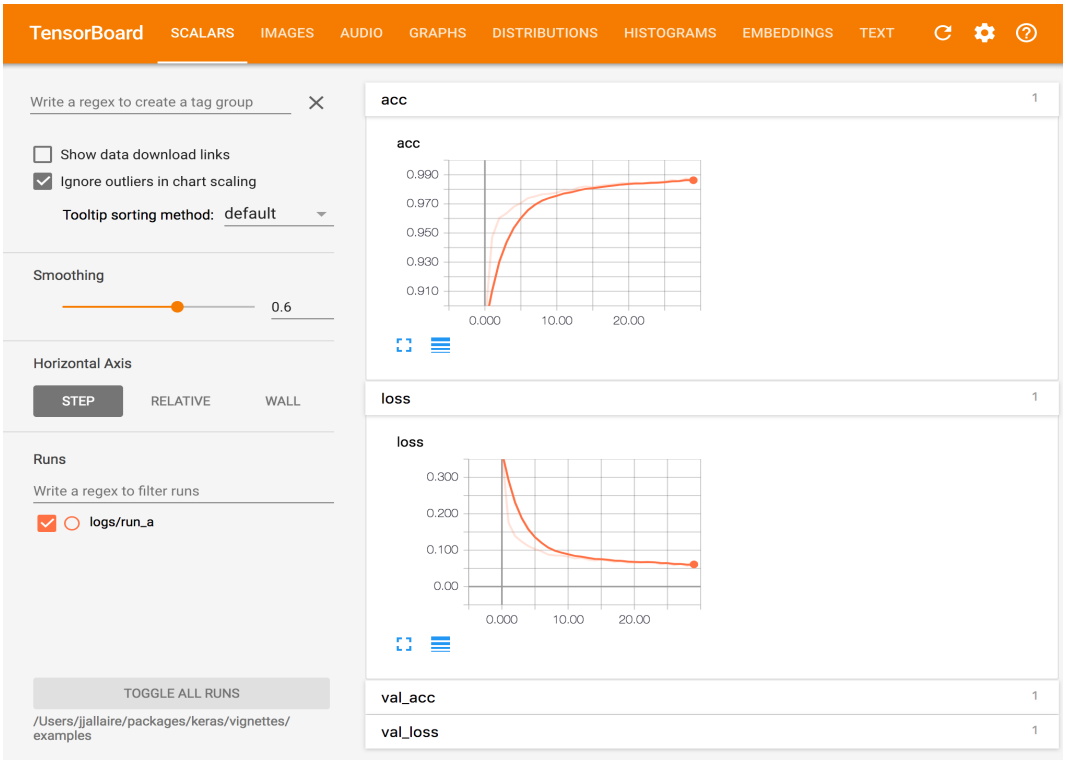


Figura 4.6: Uno de los paneles de *TensorBoard*, junto con algunas gráficas

- ***Google Cloud Platform*:** Es un servicio de *Google* que ofrece una infraestructura en la nube, la cual permite la creación de máquinas virtuales y el manejo de entornos ofertados por ellos mismos. Con esta plataforma se pueden ejecutar tareas que requieren de mucha capacidad y velocidad de procesamiento, puesto que las máquinas que oferta esta herramienta cuentan con especificaciones técnicas muy punteras y eficientes. Gracias a esta plataforma, tareas que podrían llevarse a cabo en día o

semanas en ser ejecutada, pueden llevarse a cabo en cuestión de horas, lo cual supone un gran ahorro de tiempo, el cual se puede invertir en más experimentación. Para poder acceder a la instancia propiamente dicha, y poder así hacer uso de las utilidades de la máquina virtual, es necesaria una comunicación vía *Secure Shell* (SSH) con la plataforma. Cabe destacar que es necesario un mantenimiento económico para poder acceder a la plataforma y que, además, como el entorno se encuentra en la nube y es, por tanto, independiente de los entornos locales, muchos son los problemas que pueden ocasionar el impedimento al acceso, como una caída de los servidores o la falta de acceso a internet. En lo que al desarrollo de este trabajo se refiere, se ha utilizado dicha plataforma para ejecutar la tarea de entrenamiento de los modelos, la cual requiere demasiada capacidad de computación para un ordenador de sobremesa convencional, obteniendo en algunos casos el beneficio de poder realizar un entrenamiento cinco veces más rápido de lo que se estaba realizando con nuestro equipo en local.

Capítulo 5

Experimentos y Resultados

En este capítulo se describen los experimentos que se han realizado y se evaluará el rendimiento y eficacia de cada uno de ellos, obteniendo finalmente unos resultados de los que se podrán obtener posteriormente conclusiones. Para ello, se explicará primero el contexto sobre cual se ha realizado la experimentación y, posteriormente, se hará una pequeña introducción a los experimentos realizados para pasar luego a explicarlos detalladamente cada uno de ellos. Se terminará con una sección donde se comparen los mejores experimentos realizados y se determine cuáles son los más apropiados para el trabajo.

5.1. Contexto de la Experimentación

Para llevar a cabo la experimentación, se han escogido los dos mejores detectores que existen actualmente y que más garantías han dado en trabajos previos: SSD y *Faster R-CNN*. Para cada uno de ellos, se han utilizado diferentes extractores de características preentrenados con el conjunto de datos de COCO para comprobar cuáles de ellos conseguía extraer mejor la información de nuestro conjunto de datos escogido. A su vez, se ha aplicado la técnica de TL de manera que el modelo sea capaz de detectar rostros también. El conjunto de datos elegido para esta experimentación ha sido Wider Face, puesto que tenía fotos de mayor variedad, en distintas situaciones y en mayor cantidad.

Para cada modelo que se ha creado, se ha ejecutado las veces necesarias hasta que se ha obtenido la información apropiada y se han podido generar todos los checkpoints necesarios para crear el modelo posteriormente en un punto que se encontrara en un ajuste perfecto, es decir, que no hubiera *Overfitting* ni *Underfitting*. Además, para poder evaluarlos en igualdad de condiciones, se ha dejado por defecto la configuración más importante de las redes, teniendo por delante cada uno de ellos 200000 pasos de ejecución: la cantidad recomendada por *TensorFlow* para los modelos seleccionados.

En cuanto a la fase de evaluación hay que destacar que, por defecto, estaba establecido que se realizarán periódicamente evaluaciones sobre los últimos 10 checkpoints generados, de manera que durante el entrenamiento se han ido obteniendo unos resultados basados no solo en el punto actual, sino en los anteriores también. La ventaja de esto reside en que se puede obtener una visión más globalizada del rendimiento del modelo. No obstante, se ha realizado una evaluación completa final sobre el punto de inflexión del modelo, es decir, donde este actúa con mayor rendimiento.

Esta última evaluación se ha realizado de la misma manera en la que está especificado en las métricas de COCO. Se ha establecido un umbral de confianza de 0.3 y se ha hecho la media aritmética sobre 10 umbrales de IOU, penalizando así al modelo cuando realice una mala localización de la cara en cuestión.

Se han analizado los resultados no solo en vídeos, sino también con *webcam* y en plataformas de *streaming*. Las plataformas escogidas han sido las que mayor uso tienen actualmente, es decir, *YouTube* y *Twitch*. Para cada una de estas opciones se mostrará el tiempo de inferencia y los fotogramas por segundo (del ingles FPS) resultantes. En el caso de la *webcam* y las plataformas de *streaming*, el tiempo de lectura del fotograma puede variar según diferentes factores, como podría ser la conexión a internet y/o el procesamiento que realicen las librerías utilizadas para la obtención de este. Debido a esto, los FPS en estos casos puede verse afectado.

Por último, todas las pruebas se han hecho en dos equipos distintos. Los primeros entrenamientos se realizaron en un equipo propio, el cual realizaba estos de una manera muy lenta y no iba a permitir poder hacer una gran cantidad de experimentos con él. Debido a ello, se buscaron otras alternativas. Finalmente se decidió usar *Google Cloud* para poder crear una instancia mejor y realizar todos los entrenamientos en ella.

El primer entorno de experimentación utilizado se podría resumir con las características descritas en la Tabla 5.1.

Tabla 5.1: Primer entorno de experimentación

CPU	Memoria RAM	Tarjeta Gráfica
Intel Core i5-7600K CPU 3.8GHz	8 GB	MSI GeForce GTX 1060 x 6 GB

El segundo equipo utilizado basado en computación en la nube posee las características que se muestran en la Tabla 5.2.

Tabla 5.2: Segundo entorno de experimentación

CPU	Memoria RAM	Tarjeta Gráfica
Procesador de Intel de 16 núcleos	60 GB	NVIDIA Tesla P100 x 16 GB

5.2. Experimentos

Los extractores de características seleccionados son los siguientes: *MobileNetV1*, *InceptionV2*, *ResNet50* y *Inception-ResNetV2*. Cada uno de ellos cuenta con unas características y complejidad diferente. Se ha decidido descartar otras CNN, puesto que su complejidad era demasiado grande y podría afectar tanto al tiempo de entrenamiento, como a su rendimiento, ya que se podrían *Overfittear* muy fácilmente.

MobileNet se caracteriza por ser la CNN más simple con 4 millones de parámetros configurables y una profundidad de 88, lo cual simboliza el número de filtros o características que se van a extraer de cada imagen.

Inception cuenta con 23 millones de parámetros y una profundidad mucho mayor que *MobileNet*, siendo esta de 159 filtros. De la misma forma, *ResNet* tiene una complejidad similar teniendo la cantidad de 25 millones de parámetros configurables.

Por último, *Inception-ResNet* es la red más elaborada. Tiene 55 millones de parámetros y la cantidad de 572 filtros para extraer diferentes características de las imágenes.

La experimentación comienza con dos fases diferenciadas donde se prueba con los dos detectores mencionados previamente. Para cada uno de ellos, se han aplicado de manera común las CNN *Inception* y *ResNet*.

Además, en particular, se ha aplicado *MobileNet* al detector SSD, ya que al ser la red más simple podría potenciar aún más la velocidad de inferencia que ostentan los detectores de una etapa y, por otro lado, se ha aplicado *Inception-ResNet* a *Faster R-CNN* para potenciar el rendimiento de los detectores de dos etapas con un extractor más complejo.

5.2.1. Fase 1: Experimentos Realizados con el Detector SSD

Se realizó una primera fase donde se probó con el Detector SSD. En él se utilizaron principalmente 3 extractores de características diferentes, entre los que se encuentran: *MobileNet*, *Inception* y *ResNet*.

5.2.1.1. MobileNet

El primer modelo que se ha probado, se basó en *MobileNet*. Con esta se hicieron todas las pruebas necesarias para entender como funcionaba la detección con *TensorFlow* y se entrenó el modelo tantas veces hasta que se consiguió entender como conseguir toda la información necesaria para un análisis posterior.

Este modelo fue el detonante para pensar otras alternativas más potentes para poder realizar las pruebas, debido a que se entrenó en el ordenador en local y tardó 5 días en efectuar 200000 pasos de ejecución. Esa prueba que se hizo, no generó todos los checkpoints necesarios para poder crear luego un modelo del cual se pueda inferir sobre nuevas imágenes, por lo que quedó desechada.

Mientras se buscaba la solución al problema de los checkpoints y se estaba comenzando a hacer la instalación necesaria en una máquina disponible en la nube, se siguió haciendo pruebas con este modelo intentando parar el entrenamiento cuando se comenzara a ver *Overfitting* en él.

Finalmente, se pudo completar el entrenamiento en 40 horas en la máquina local y se generó el modelo que se ve en la Figura 5.1.

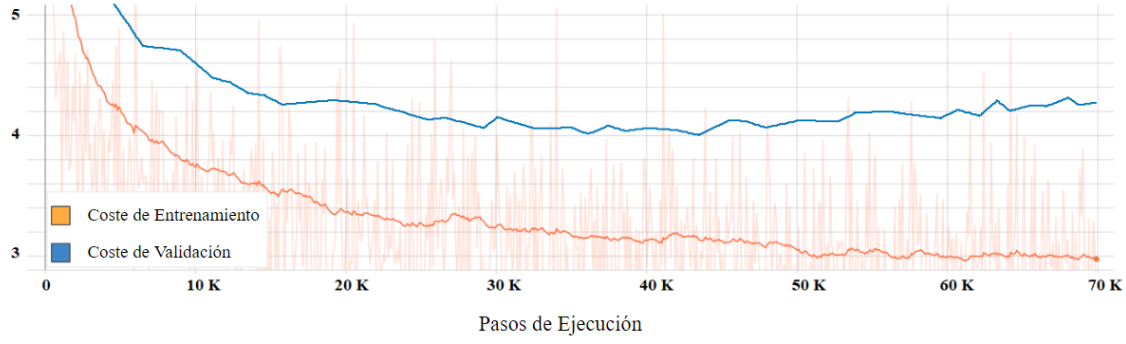


Figura 5.1: Curvas de aprendizaje de SSD con *MobileNet*

Como se puede ver en la gráfica, se efectuaron 70000 pasos de ejecución y se aprecia que no se consigue un buen ajuste entre las dos curvas generadas.

Si se sigue la trayectoria de estas dos curvas, se ve que ambas tienen un coste bastante alto pero continúan disminuyendo hasta los 43200 pasos de ejecución aproximadamente. En ese punto, el coste de validación comienza a ascender, mientras que el de entrenamiento sigue bajando, por lo que se podría determinar que a partir de ese punto comienza a haber un sobreajuste.

En ese instante del entrenamiento, se ha generado el modelo y se han obtenido los siguientes resultados:

En la Tabla 5.3, se observa la precisión media obtenida durante el entrenamiento, junto con la precisión media sobre los umbrales IOU de 0.5 y 0.75. Como se puede ver, no es una media muy buena y, como es obvio, empeora bastante cuanto mayor es el umbral de la localización.

Tabla 5.3: Precisión media usando SSD con *MobileNet*

mAP (%)	**mAP@.50 IOU (%)**	**mAP@.75 IOU (%)**
30.35	63.92	23.22

La Tabla 5.4, representa los resultados que se obtienen en cuanto a precisión se refiere dependiendo de la distancia a la que se encuentre la cara en la imagen. Para caras que se encuentra muy alejadas, la precisión es bastante mala, con solo un 5.94 %. Sin embargo, según van haciéndose más grande, la precisión aumenta hasta un 53.11 % en su mayor tamaño.

Tabla 5.4: Precisión media en distintas distancias usando SSD con *MobileNet*

mAP *Small* (%)	**mAP *Medium* (%)**	**mAP *Large* (%)**
5.94	26.32	53.11

En cuanto a la exhaustividad, se aprecia que realizando una sola detección, se obtiene un resultado bastante malo también con un 13.36 %. Este porcentaje aumenta según se incrementa el número de detecciones realizadas. Esto se puede ver en la Tabla 5.5.

Tabla 5.5: Exhaustividad media usando SSD con *MobileNet*

AR@1 (%)	**AR@10 (%)**	**AR@100 (%)**
13.36	32.82	35.76

Estos resultados mejoran cuando las caras que se están detectando están bastante cerca en la imagen, tratándose así un 58.47 % en este caso. Al igual que con la precisión, en rostros lejanos, es bastante malo el resultado y, con caras a distancia media, se mantiene de forma parecida que en AR@100. Se muestra esto en la Tabla 5.6.

Tabla 5.6: Exhaustividad media en distintas distancias usando SSD con *MobileNet*

AR *Small* (%)	**AR *Medium* (%)**	**AR *Large* (%)**
8.67	32.54	58.47

Es de debida importancia destacar y recalcar que los resultados que se acaban de mencionar con las distintas tablas, están basados no solo en el punto de los 43.200 pasos de ejecución, sino también en los 10 *checkpoints* anteriores a él. En este caso, estarían incluidos todos los checkpoints desde el paso de ejecución 0. Al haber tanta diferencia de rendimiento, los resultados se ven empeorados. Debido a ello, se va a calcular tanto la precisión, como la exhaustividad y a realizar un recuento de los tipos de predicciones que se han realizado únicamente en el punto de mayor rendimiento.

Como se muestra en la Tabla 5.7, se han puesto los resultados según el umbral de IOU, como estipulan las métricas de COCO.

Tabla 5.7: Resultados más específicos usando SSD con *MobileNet*

Medida	mAP (%)	AR (%)	VP	FN	FP	VN	Exactitud (%)
@.50 IOU	94.52	64.44	7573	4178	439	0	62.12
@.55 IOU	94.48	64.40	7568	4183	442	0	62.06
@.60 IOU	94.41	64.31	7558	4193	447	0	61.96
@.65 IOU	94.41	64.28	7554	4197	447	0	61.92
@.70 IOU	94.26	64.16	7540	4211	459	0	61.75
@.75 IOU	94.24	64.13	7536	4215	460	0	61.71
@.80 IOU	94.04	63.95	7515	4236	476	0	61.46
@.85 IOU	93.72	63.65	7480	4271	501	0	61.05
@.90 IOU	93.33	63.18	7425	4326	530	0	60.45
@.95 IOU	89.55	58.64	6891	4860	804	0	54.88
Media	93.69	63.51	7464	4287	500	0	60.92

Se puede ver que la precisión mejora bastante en comparación con el rendimiento anterior, siendo este de un 93.69 %. Esto significa que casi el 94 % de las veces, los objetos detectados coinciden con los originales. Sin embargo, la exhaustividad es bastante peor siendo de un 63.51 %. Esto refleja que el modelo no es capaz de detectar muchas caras y, en consecuencia, el número de falsos negativos es bastante grande.

Todo esto da una exactitud de casi un 61 %, significando que el 61 % de las veces, una cara va a ser correctamente detectada.

Se van a considerar ahora a explicar los resultados que se han obtenido utilizando este modelo para inferir no solo en vídeos, sino también en *streaming* en *Twitch* y *YouTube* y con una *Webcam*.

Como se puede ver en la Tabla 5.8, resulta ser un modelo bastante rápido en cuanto a tiempo de inferencia se refiere, siendo el tiempo de detección casi instantáneo: 0.013 segundos. Para el número de 3398 fotogramas, es capaz de procesarlos con un ratio de 33 FPS en vídeos y 18 en *webcam*. Es una cantidad bastante alta para ambos y le hace un buen candidato para usar en formato *streaming*.

En definitiva, usando *MobileNet* como extractor de características proporcionaría un modelo rápido a la hora de detectar sobre los fotogramas de un vídeo, no obstante, pecaría de no tener una gran exactitud. Principalmente esto se debe a que contiene muchos falsos negativos, que hace que muchos rostros no sean detectados. En la Figura 5.2 se pueden mostrar algunos ejemplos de detecciones realizados con este modelo.

Tabla 5.8: Resultados usando el modelo SSD con *MobileNet* en vídeos y *streaming*

Plataforma	Fotogramas	T. Inferencia	FPS
Vídeo	3398	0.013 segundos	33
Webcam	3398	0.013 segundos	18
YouTube	3398	0.013 segundos	5
Twitch	3398	0.013 segundos	8

Figura 5.2: Ejemplos de detección usando SSD con *MobileNet*

En las imágenes se pueden apreciar como actúa con un grupo de personas a corta, media y larga distancia, así como con personas de origen asiático.

5.2.1.2. Inception

El segundo modelo que se ha creado, se basó en la CNN *Inception*. Esta finalmente se ejecutó en la instancia que creada en la nube y se pudo percibir los primeros atisbos de la gran decisión que se había tomado al buscar otra alternativa para realizar los entrenamientos. El tiempo de ejecución fue únicamente de 22 horas, en comparación con las 111 horas que hubiese tardado aproximadamente en ejecutarse en nuestro ordenador en local.

En este punto también se había entendido por completo como generar todos los archivos y *checkpoints* necesarios para poder crear un modelo más tarde para ser analizado. Debido a esto, se pasó a entrenar el modelo y se consiguió la curva de aprendizaje que se aprecia en la Figura 5.3.

Como se puede ver en ella, esta vez se dejó más tiempo entrenando, llegando a los 200000 pasos de ejecución. Para poder analizar de manera adecuada esta curva, se hará por partes.

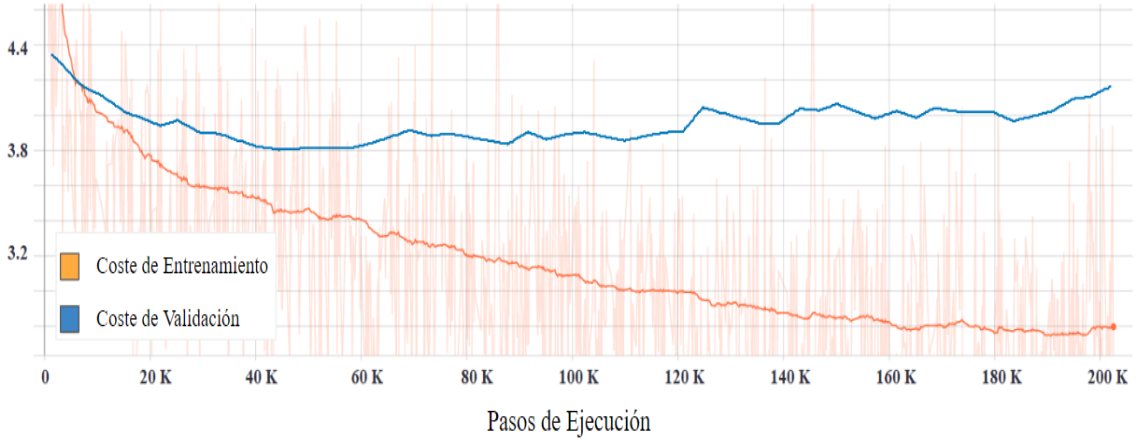


Figura 5.3: Curvas de aprendizaje de SSD con *Inception*

En primer lugar, se ve que tanto el coste de entrenamiento como de validación comienzan desde un punto más bajo que *MobileNet*, lo que podría ser una señal de que es un mejor modelo. Al comienzo y hasta los 10000 pasos de ejecución aproximadamente, el coste de validación es menor que el de entrenamiento, lo cual significa que en ese instante el conjunto de validación era mucho más fácil de predecir que el de entrenamiento.

Sin embargo, cuando se pasa esta barrera, el coste de entrenamiento comienza a descender más rápidamente que el de validación, llegando hasta los 42000 pasos de ejecución, donde los dos están en el punto más bajo en común.

A partir de aquí, el coste de validación comienza de nuevo a ascender y el de entrenamiento continua bajando, por lo que se puede considerar el punto de sobreajuste del modelo. Este es el paso de ejecución elegido para generar el modelo.

Al igual que pasó con *MobileNet*, los resultados que se van a mostrar a continuación están basados no solo en el instante escogido, sino también en los 10 checkpoints anteriores. En este caso, se estarían teniendo en cuenta los resultados obtenidos desde el principio también, por lo que el rendimiento en el punto 42000 debe ser mejor.

Como se ve en la Tabla 5.9, se obtiene una precisión media de 35.31 % que es similar a la obtenida con un umbral de 0.75 IOU y casi la mitad teniendo un 0.5 de IOU. Pese a no ser precisiones muy buenas tampoco, se observa que son mejores que las obtenidas con el modelo anterior.

Tabla 5.9: Precisión media usando SSD con *Inception*

mAP (%)	**mAP@.50 IOU (%)**	**mAP@.75 IOU (%)**
35.31	67.12	33.03

Si se presta atención con más detalle en la precisión obtenida según la distancia a la que esté el rostro 5.10, se verá que la detección sigue siendo bastante mala para las caras que se encuentran alejadas. Lo mismo ocurre para rostros a media y larga distancia, siendo este último donde mejor precisión se obtiene. No obstante, los tres resultados vuelven a ser mejor que usando *MobileNet*.

Tabla 5.10: Precisión media a distintas distancias usando SSD con *Inception*

mAP *Small* (%)	**mAP *Medium* (%)**	**mAP *Large* (%)**
7.73	31.78	58.27

Se pasa ahora a analizar la exhaustividad. Como se puede ver en la Tabla 5.11, esta vuelve a ser mejor en todos los aspectos, respecto al modelo anterior teniendo como máximo resultado un 41 % en 100 detecciones. De igual manera pasa con la exhaustividad si se mide con respecto a la distancia (Tabla 5.12): pese a ser bastante mala, mejora en comparación con *MobileNet*, teniendo como máximo resultado un 64 % cuando la cara se encuentra en una posición cercana.

Tabla 5.11: Exhaustividad media usando SSD con *Inception*

AR@1 (%)	**AR@10 (%)**	**AR@100 (%)**
14.51	37.09	40.95

Tabla 5.12: Exhaustividad media en distintas distancias usando SSD con *Inception*

AR *Small* (%)	**AR *Medium* (%)**	**AR *Large* (%)**
11.44	37.89	64.21

Una vez se han mostrado los resultados obtenidos mientras el entrenamiento se estaba realizando, se procede a ver cuáles son realmente los que se obtienen justo en el punto en el que se ha creado el modelo, utilizando para ello diversos umbrales de IOU.

Como se ve en la Tabla 5.13, la precisión mejora bastante también con respecto a los resultados mostrados en las tablas anteriores. Lo mismo ocurre con la exhaustividad, mejora aunque en menor medida que la precisión.

Tabla 5.13: Resultados más específicos usando SSD con *Inception*

Medida	**mAP (%)**	**AR (%)**	**VP**	**FN**	**FP**	**VN**	**Exactitud (%)**
@.50 IOU	92.44	75.04	8818	2933	721	0	70.70
@.55 IOU	92.34	74.93	8806	2945	730	0	70.55
@.60 IOU	92.31	74.90	8802	2949	733	0	70.50
@.65 IOU	92.24	74.83	8794	2957	739	0	70.40
@.70 IOU	92.14	74.75	8784	2967	749	0	70.27
@.75 IOU	92.06	74.66	8774	2977	756	0	70.15
@.80 IOU	91.87	74.48	8753	2998	774	0	69.88
@.85 IOU	91.47	74.11	8709	3042	812	0	69.32
@.90 IOU	90.91	73.49	8636	3115	863	0	68.46
@.95 IOU	87.88	69.82	8205	3546	1131	0	63.69
Media	91.56	74.10	8708	3043	801	0	69.37

El 91.56 % de las veces los objetos detectados coinciden con los originales, sin embargo, hay todavía una cantidad considerable de falsos negativos que hace bajar su exhaustividad.

No obstante, si se compara con los resultados obtenidos en *MobileNet*, se está ante un modelo más equilibrado en cuanto a precisión y exhaustividad se refiere y, por tanto, la exactitud resultante es considerablemente mayor: un 69 %, haciendo de este el mejor modelo probado hasta el momento.

Una vez se ha hallado la exactitud, se procede a ver qué rendimiento ofrece el modelo al inferir en *streaming* y en vídeos, con la Tabla 5.14.

Tabla 5.14: Resultados usando el modelo SSD con *Inception* en vídeos y *streaming*

Plataforma	Fotogramas	T. Inferencia	FPS
Vídeo	3398	0.019 segundos	28
Webcam	3398	0.021 segundos	15
YouTube	3398	0.021 segundos	4
Twitch	3398	0.020 segundos	7

En ella se observa que el tiempo de inferencia, y en consecuencia los FPS, es bastante similar al obtenido con el modelo que usa *MobileNet*. Tanto para vídeo como para cámara web, sigue siendo una cantidad bastante alta para poder considerarlo un gran modelo que permita su uso en *streaming* y dado que el rendimiento en términos generales es considerablemente mayor, se podría pensar que este modelo es bastante mejor que el anterior probado.

Finalmente, en la Figura 5.4, se puede ver algunos ejemplos de su rendimiento en el momento de detección.

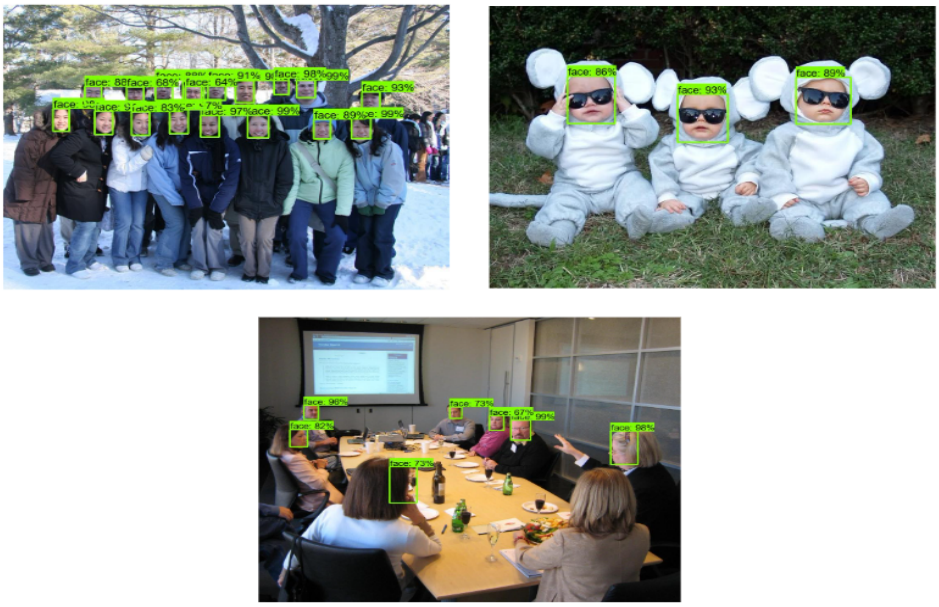


Figura 5.4: Ejemplos de detección usando SSD con *Inception*

En los ejemplos se ve un grupo de personas asiáticas, una reunión que contiene algunos rostros girados y tres bebés cubriéndose la cara con unas gafas de sol.

5.2.1.3. ResNet

Para terminar con los experimentos que se han realizado con el detector SSD, se pasará a explicar los resultados obtenidos habiendo probado la CNN *ResNet* como extractor de características.

Durante la creación de este modelo, surgieron varios problemas debido a que contenía un fichero de configuración que no estaba preparado completamente para ser ejecutado por defecto. Se hicieron varias pruebas intentando completar algunos ajustes que permitieran terminar de crear el modelo pero todas ellas producían unas curvas de aprendizaje en las que se veía que se estaba produciendo *underfitting*. Tras varios intentos e indagación en internet, se pudo completar y entrenar el modelo en un total de 9 horas en *Google Cloud*, lo que hubiera supuesto un total de 17 horas aproximadamente en local.

Se puede observar en la Figura 5.5 la gráfica definitiva que define este último entrenamiento.

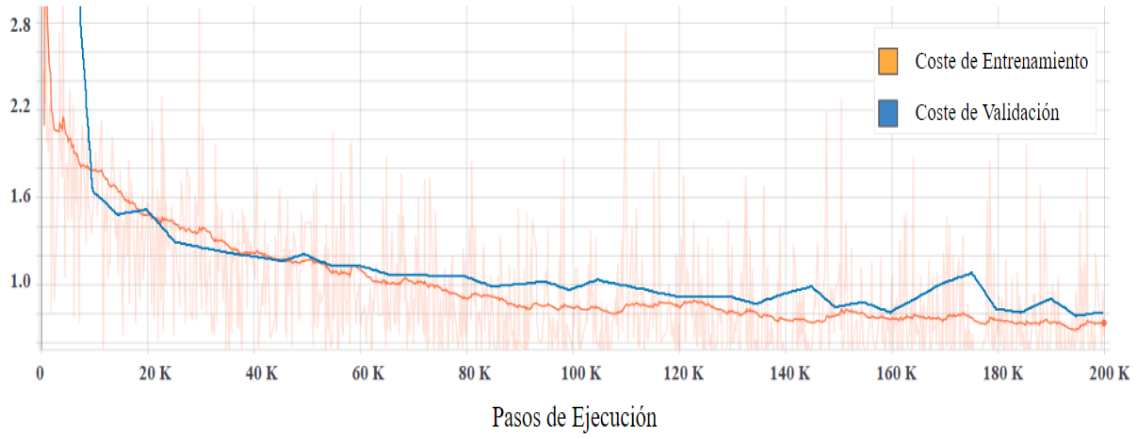


Figura 5.5: Curvas de Aprendizaje de SSD con ResNet

Como se puede ver, a grandes rasgos, se aprecia que tanto el coste de entrenamiento como el de validación están en su punto más bajo de todas los modelos probados hasta el momento. Hay ciertas zonas donde la curva de validación indica que las imágenes que se están evaluando son más sencillas de lo que lo son realmente las de entrenamiento, sin embargo, esta situación se corrige rápidamente.

Además, se observa que, durante toda la curva, estos dos costes se van ajustando bastante bien entre sí, dando como resultado el paso de ejecución 195.000, donde las dos gráficas están en su punto más bajo aproximadamente.

También se puede observar que la variabilidad de los costes no es tan grande a lo largo de los *checkpoints* en comparación con las gráficas previamente comentadas. Esto habrá que tenerlo en cuenta durante el análisis de los resultados, puesto que estos serán más fieles a la realidad al no tener que hacer la media entre unos costes muy malos y otros muy buenos, como pasaba con los modelos ya analizados.

Una vez explicada la gráfica, se va a pasar a ver qué resultados se obtuvieron durante el entrenamiento.

En la Tabla 5.15, se observa que los resultados que se obtienen en cuanto a precisión media no son muy distintos de los obtenidos usando *Inception*, aunque sí difieren un poco más de los que se obtuvieron con *MobileNet*, sobre todo en el caso de establecer un 0.75 como umbral de IOU, suponiendo una diferencia de casi un 21 %.

Tabla 5.15: Precisión media usando SSD con *ResNet*

mAP (%)	**mAP@.50 IOU (%)**	**mAP@.75 IOU (%)**
41.26	70.81	44.43

Prestando atención a la precisión obtenida dependiendo de la distancia en la Tabla 5.16, se observa que se vuelven a asemejar y diferenciar a *Inception* y *MobileNet* respectivamente. La mayor diferencia entre *Inception* y *ResNet* reside en el porcentaje que se obtiene cuando la cara está a grandes distancias, siendo de un 10 % en beneficio de este modelo. Sin embargo, con respecto a *MobileNet* se consigue mejorar sustancialmente tanto a largas como medianas distancias.

Tabla 5.16: Precisión media a distintas distancias usando SSD con *ResNet*

mAP *Small* (%)	**mAP *Medium* (%)**	**mAP *Large* (%)**
17.98	39.77	56.35

Se procede ahora a analizar qué ocurre en el caso de la Exhaustividad. Para ello se mostrarán los resultados en las Tablas 5.17 y 5.18.

Tabla 5.17: Exhaustividad media usando SSD con *ResNet*

AR@1 (%)	**AR@10 (%)**	**AR@100 (%)**
14.75	41.67	52.76

Tabla 5.18: Exhaustividad media en distintas distancias usando SSD con *ResNet*

AR *Small* (%)	**AR *Medium* (%)**	**AR *Large* (%)**
31.31	51.69	65.34

Si se comparan estas tablas con los modelos anteriores, se aprecia que los resultados vuelven a aumentar un poco. La exhaustividad media aumenta un 17 % y 12 %, correspondiendo a *MobileNet* e *Inception* y tomando como referencia las 100 detecciones. Sin embargo, atendiendo a cómo de lejos se encuentre el rostro, se observa que se consigue aumentar los resultados a media y larga distancia. Se consigue un aumento del 23 % y 19 % respectivamente en *MobileNet*, y un incremento del 20 % y 14 % en *Inception*.

Pese a haber obtenido estos resultados que aparentemente son mejores, no hay que fiarse de ellos puesto que los anteriores modelos se veían más influenciados por los *checkpoints*. Por tanto, podría darse el caso de que las CNN ya mencionadas, sean mejores que *ResNet* en ultima instancia.

Por ello, para salir de dudas se ha creado la Tabla 5.19 donde se podrá ver con más exactitud lo que ocurre en el *checkpoint* generado.

Tabla 5.19: Resultados más específicos usando SSD con *ResNet*

Medida	mAP (%)	AR (%)	VP	FN	FP	VN	Exactitud (%)
@.50 IOU	95.35	57.97	6813	4938	332	0	56.38
@.55 IOU	95.31	57.94	6809	4942	335	0	56.33
@.60 IOU	95.28	57.89	6803	4948	337	0	56.27
@.65 IOU	95.23	57.86	6800	4951	340	0	56.24
@.70 IOU	95.18	57.83	6796	4955	344	0	56.18
@.75 IOU	95.13	57.78	6790	4961	347	0	56.12
@.80 IOU	95.06	57.71	6782	4969	352	0	56.03
@.85 IOU	94.98	57.64	6774	4977	358	0	55.94
@.90 IOU	94.61	57.22	6724	5027	383	0	55.41
@.95 IOU	92.77	54.84	6445	5306	502	0	52.59
Media	94.89	57.46	6753	4997	363	0	55.75

Como se sospechaba, los resultados son bastantes malos usando *ResNet* como extractor de características. La relación entre precisión y exhaustividad es bastante desequilibrada y eso produce que la exactitud deje mucho que desear también. En las detecciones se producen una gran cantidad de falsos negativos, casi tantos como verdaderos positivos, por lo que habrá muchas caras que no sean detectadas correctamente.

No solo no tiene una gran exactitud, sino que además tampoco responde demasiado bien cuando se utiliza para inferir sobre vídeos o *streaming*. En la Tabla 5.20 se observa que detectando en vídeos se consigue una cantidad de 11 FPS procesados, durando esta inferencia casi 0.1 segundos. Además, usando la *webcam* se obtiene casi el mismo rendimiento. Por lo que se podría concluir con que, de todas las que se han analizado hasta el momento, no es un gran modelo para *streaming*.

Tabla 5.20: Resultados usando el modelo SSD con *ResNet* en vídeos y *streaming*

Plataforma	Fotogramas	T. Inferencia	FPS
Vídeo	3398	0.075 segundos	11
Webcam	3398	0.076 segundos	10
YouTube	3398	0.077 segundos	3
Twitch	3398	0.075 segundos	5

En la Figura 5.6 se pueden ver algunos ejemplos del rendimiento de este modelo. Se puede observar una clase *fitness* donde se aprecian caras giradas, un grupo de amigos de distintas etnias a media distancia y un grupo de deportistas con diversos accesorios en la cabeza.

Figura 5.6: Ejemplos de detección usando SSD con *ResNet*

5.2.2. Resultados de la Fase 1

Tras finalizar esta fase, se ha podido comprobar que los modelos que usan el detector SSD son bastante rápidos en cuanto a tiempo de inferencia se refiere, siendo aquel que usa la CNN *MobileNet* el más veloz en este sentido. Además, se ha podido ver que, en general, no se pueden obtener unos resultados muy destacables si el algoritmo se centra en detectar de forma rápida. No obstante, el modelo que usa *Inception* funciona bastante bien, ya que proporciona casi un 70 % de exactitud, con una velocidad suficientemente aceptable como para usarlo en vídeos y *streamings* en tiempo real.

Por tanto, como conclusión de esta primera fase de experimentación, se considera que el modelo SSD con *Inception* es el mejor en términos generales.

Se descarta, por consiguiente, el modelo que usa *MobileNet* puesto que a pesar de ser la más rápida, tiene mucho más falsos positivos y es incapaz de realizar inferencias de una forma más efectiva que *Inception*. A su vez, *ResNet* queda descartada también por su escasa exactitud y lenta detección a la hora de actuar sobre vídeos y *streaming*.

5.2.3. Fase 2: Experimentos Realizados con el Detector Faster R-CNN

En esta segunda fase se utilizó el Detector *Faster R-CNN* como base. En él se fueron aplicando diversos extractores de características, entre los que se encuentran: *Inception*, *ResNet* e *Inception-ResNet*.

5.2.3.1. Inception

Inception fue el primer extractor de características que se utilizó con el detector *Faster R-CNN*. Esta red, tiene como autor varios desarrolladores de *Google* [SVI+15] y es uno de los extractores que mejores resultados ha dado en otros proyectos, por lo que se podrían esperar buenos resultados.

Las pruebas se hicieron en la instancia *Cloud*, obteniendo un tiempo de entrenamiento de 2 horas aproximadamente, uno de los más rápidos hasta el momento. No obstante, si no se hubiera hecho mediante este método podría haber tardado unas 11 horas.

En la Figura 5.7 se puede ver la curva de aprendizaje obtenida durante su ejecución.

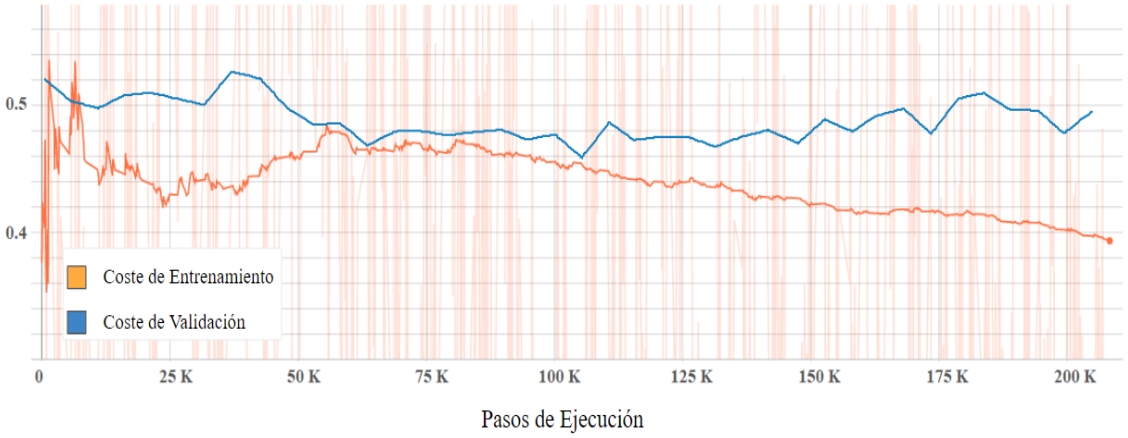


Figura 5.7: Curvas de Aprendizaje de Faster R-CNN con *Inception*

Como se ve en la gráfica, el entrenamiento se realizó durante 200000 pasos de ejecución al igual que el resto de pruebas realizadas. El coste, además de ser el más bajo que se ha obtenido hasta el momento, es de los más estables, no habiendo de esta manera una gran diferencia entre el punto más alto y el más bajo. Se observa además que el instante de mayor rendimiento ocurre en las 105000 iteraciones aproximadamente, donde ambas curvas están en su punto más bajo. A partir de ahí, el coste de entrenamiento continua bajando y el de validación comienza a subir de nuevo.

Se procede a analizar los resultados comparándolos con los de el modelo basado en *Inception* y SSD, puesto que ha sido escogido como el mejor de dicho detector.

Comenzando a analizar estos (Tabla 5.21), se observa que son los más destacables que se han producido hasta este punto. Además sucede un aumento de un 7 % aproximadamente en la peor de estas tres métricas.

La precisión dependiendo de la distancia vuelve a aumentar también, con respecto a la versión homónima y con diferente detector. El mayor porcentaje de mejora también corresponde a un 7 %, producido a media distancia.

Tabla 5.21: Precisión media usando *Faster R-CNN* con *Inception*

mAP (%)	**mAP@.50 IOU (%)**	**mAP@.75 IOU (%)**
42.88	78.93	41.83

Tabla 5.22: Precisión media en distintas distancias usando R-CNN con *Inception*

mAP *Small* (%)	**mAP *Medium* (%)**	**mAP *Large* (%)**
11.68	38.80	63.38

Se pasa a analizar cuál ha sido esta vez la exhaustividad tanto dependiendo de la cantidad de detecciones como de la distancia del rostro: Tablas 5.23 y 5.24, respectivamente.

Tabla 5.23: Exhaustividad media usando *Faster R-CNN* con *Inception*

AR@1 (%)	**AR@10 (%)**	**AR@100 (%)**
15.76	43.02	48.12

Tabla 5.24: Exhaustividad media en distintas distancias usando Faster R-CNN con *Inception*

AR *Small* (%)	**AR *Medium* (%)**	**AR *Large* (%)**
24.86	45.27	67.94

En este caso todos los resultados son prometedores, viéndose un aumento bastante considerable en la detección a grandes distancias con un 24 % en comparación con el 11 % obtenido en SSD. Además, también se incrementa el porcentaje para AR@100, por lo que es posible que el número de falsos negativos decrezca en comparación a todo lo que se ha probado hasta el momento.

Se procede a generar, por tanto, los resultados sin la influencia de los *checkpoints* anteriores al que se ha seleccionado como mejor.

En la Tabla 5.25 se puede percibir fácilmente que el modelo ofrece unos resultados bastante favorables. Pese a tener menor precisión que SSD con *Inception*, tiene mayor exhaustividad. Esto se traduce en que tiene mayor número de falsos positivos y, a su vez, menores falsos negativos.

Tabla 5.25: Resultados más específicos usando Faster R-CNN con *Inception*

Medida	**mAP (%)**	**AR (%)**	**VP**	**FN**	**FP**	**VN**	**Exactitud (%)**
@.50 IOU	85.53	86.85	10206	1545	1726	0	75.72
@.55 IOU	85.51	86.78	10198	1553	1728	0	75.65
@.60 IOU	85.45	86.72	10191	1560	1734	0	75.57
@.65 IOU	85.41	86.68	10186	1565	1739	0	75.50
@.70 IOU	85.28	86.54	10170	1581	1755	0	75.29
@.75 IOU	85.16	86.40	10153	1598	1769	0	75.09
@.80 IOU	85.03	86.23	10134	1617	1783	0	74.87
@.85 IOU	84.76	85.90	10095	1656	1814	0	74.41
@.90 IOU	84.39	85.39	10035	1716	1855	0	73.75
@.95 IOU	82.62	82.16	9655	2096	2031	0	70.05
Media	84.91	85.96	10102	1648	1793	0	74.59

Estos resultados hacen que, al estar más equilibrados que el resto, se consiga una exactitud mayor, siendo en este caso de un 74.59 %.

Atendiendo a su rendimiento sobre vídeos y *streaming* (Tabla 5.26), se observa que obtiene un rendimiento bastante bajo en vídeos y, sin embargo, mejora en *streaming* donde se llega a obtener 13 FPS en el caso de la *webcam*, funcionando 2 FPS por debajo del mejor modelo que usa SSD.

Tabla 5.26: Resultados usando el modelo *Faster R-CNN* con *Inception* en vídeos y *streaming*

Plataforma	Fotogramas	T. Inferencia	FPS
Vídeo	3398	0.058 segundos	13
Webcam	3398	0.055 segundos	13
YouTube	3398	0.058 segundos	3
Twitch	3398	0.061 segundos	4

Además de eso, proporciona una mejor exactitud por lo que podría considerarse otro buen candidato para usar en tiempo real.

En la Figura 5.8 se muestran algunos ejemplos de detecciones realizados con este modelo. En ella se aprecia como se detecta a la perfección rostros ya sea con la cara pintada o portando un casco. Además, a diferencia de los modelos que usan SSD como detector, se ve que los resultados a largas distancias son más favorables.

Figura 5.8: Ejemplos de detección usando *Faster R-CNN* con *Inception*

5.2.3.2. ResNet

El modelo *ResNet* es el segundo que se ha probado con este detector. Las pruebas se hicieron de nuevo en *Cloud* y el entrenamiento duró unas 18 horas. De no haber sido por esta plataforma, hubiera demorado 51 horas, es decir, un poco más de dos días completos en poder obtener unos resultados.

En la Figura 5.9, se puede ver las curvas de aprendizaje generadas al realizar el entrenamiento.

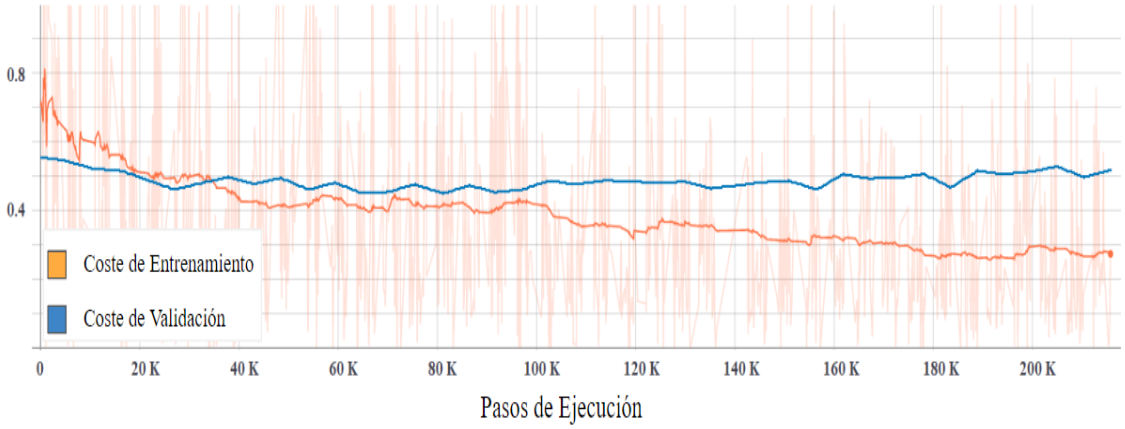


Figura 5.9: Curvas de Aprendizaje de *Faster R-CNN* con *ResNet*

En ella se aprecia que el coste tanto de entrenamiento como de validación es ligeramente superior al que usa *Inception*, al igual que la diferencia existente entre el punto más alto y el más bajo. Además, el coste también es bastante inferior al que se puede hallar en *Inception* con SSD.

Pese a ser más bajo el coste de validación al comienzo de la curva, rápidamente se adapta y consigue un ajuste bastante bueno hasta el paso de ejecución 95000 aproximadamente: el punto donde se ha generado el modelo. A partir de ahí comienza a suceder un ligero *overfitting* el cual debe evitarse.

En la Tabla 5.27 se pueden ver los resultados que se obtienen en cuanto a precisión media.

Tabla 5.27: Precisión media usando *Faster R-CNN* con *ResNet*

mAP (%)	**mAP@.50 IOU (%)**	**mAP@.75 IOU (%)**
47.01	82.75	48.63

Los tres valores superan los producidos usando *Inception* como CNN. El valor en el que más aumento se ha producido ha sido especialmente cuando se establece la restricción de 0.75 en el IOU, por lo que se produciría una localización más efectiva.

Si se comparan teniendo en cuenta la lejanía del rostro en la Tabla 5.28, se vuelve a ver que los resultados son de nuevo favorables tanto para largas como para medias distancias. Sin embargo, en cortas distancias se mantiene el mismo resultado.

Tabla 5.28: Precisión media en distintas distancias usando *Faster R-CNN* con *ResNet*

mAP *Small* (%)	**mAP *Medium* (%)**	**mAP *Large* (%)**
13.97	44.53	63.84

Se procede por último a analizar qué ha ocurrido con la exhaustividad. Si se presta atención en la exhaustividad media (Tabla 5.29), no ha aumentado demasiado con respecto a *Inception*. El valor más destacable es 52.46 % cuando se realizan 100 detecciones, superando en un 4 % al obtenido con el modelo anterior. Lo mismo ocurre con 5.30, produciéndose un ligero aumento en las tres métricas.

Tabla 5.29: Exhaustividad media usando *Faster R-CNN* con *ResNet*

AR@1 (%)	AR@10 (%)	AR@100 (%)
16.37	45.91	52.46

Tabla 5.30: Exhaustividad media en distintas distancias usando *Faster R-CNN* con *ResNet*

AR *Small* (%)	AR *Medium* (%)	AR *Large* (%)
32.31	50.42	68.14

En este caso, vuelve a pasar lo mismo que ocurrió usando SSD con *ResNet*. La gráfica presenta una tendencia bastante recta desde el paso de ejecución 40000 hasta el 100000 aproximadamente. En este rango se ve involucrado los 10 *checkpoints* anteriores al que se ha elegido para generar el modelo. Por lo tanto, los resultados durante el entrenamiento en esta parte, estuvieron menos influenciados por la alta variabilidad de la curva y se acercan más a la realidad que en el caso de *Faster R-CNN* con *Inception*, el cual deberían ser unos resultados mayores si no se hubiese visto afectado por este motivo.

Los resultados que se han generado al centrarse únicamente en los 95000 pasos de ejecución son los que se pueden ver en la Tabla 5.31. En primera instancia, se ve que la relación precisión-exhaustividad es más desequilibrada que la obtenida en *Faster R-CNN* con *Inception*. Se caracteriza por tener una exhaustividad mayor que la precisión, por lo que el modelo será capaz de detectar más rostros, pero también sucederá un mayor número de falsos positivos.

Tabla 5.31: Resultados más específicos usando *Faster R-CNN* con *ResNet*

Medida	mAP (%)	AR (%)	VP	FN	FP	VN	Exactitud (%)
@.50 IOU	79.52	89.60	10529	1222	2711	0	72.80
@.55 IOU	79.50	89.56	10525	1226	2713	0	72.76
@.60 IOU	79.49	89.53	10521	1230	2714	0	72.73
@.65 IOU	79.44	89.48	10515	1236	2720	0	72.66
@.70 IOU	79.38	89.41	10507	1244	2728	0	72.56
@.75 IOU	79.16	89.27	10491	1260	2743	0	72.38
@.80 IOU	79.16	89.09	10470	1281	2756	0	72.17
@.85 IOU	78.93	88.79	10434	1317	2785	0	71.78
@.90 IOU	78.61	88.17	10362	1389	2819	0	71.11
@.95 IOU	77.01	85.09	9999	1752	2984	0	67.85
Media	79.02	88.79	10435	1315	2767	0	71.88

Esto se ve claramente reflejado en su exactitud, siendo un poco menor que en *Inception*, pero ligeramente mayor que en el mejor modelo de SSD, donde los resultados fueron justo al revés teniendo una precisión mayor que la exhaustividad.

El resultado que se observa en vídeos y *streaming* (Tabla 5.32) es parecido a su homónimo usando el detector SSD. Es de los peores modelos analizados hasta el momento en este sentido, teniendo la capacidad de procesar 8 FPS en el caso de los vídeos y la *webcam*. Por lo que, hasta el momento, no le haría merecedor de ser mejor que el modelo compuesto por *Faster R-CNN* e *Inception*, debido a su exactitud y el rendimiento que se acaba de ver en vídeos y *streaming*.

Tabla 5.32: Resultados usando el modelo *Faster-RCNN* con *ResNet* en vídeos y *streaming*

Plataforma	Fotogramas	T. Inferencia	FPS
Vídeo	3398	0.112 segundos	8
Webcam	3398	0.103 segundos	8
YouTube	3398	0.110 segundos	1
Twitch	3398	0.109 segundos	3

En la Figura 5.10 se pueden ver algunos de ejemplos de cómo se comporta este modelo en diversas imágenes. En ellas se aprecia como, a pesar de haber una serie de personas con la cara tapada con mascarillas o indumentaria militar, es capaz de detectarles perfectamente.

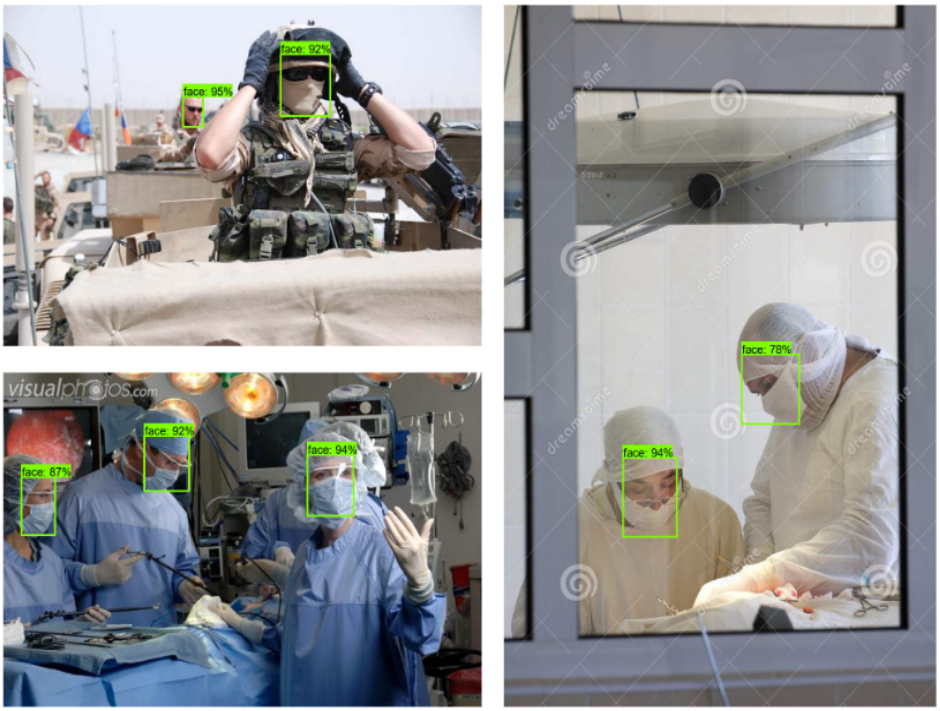


Figura 5.10: Ejemplos de detección usando *Faster R-CNN* con *ResNet*

5.2.3.3. Inception-ResNet

Inception-ResNet fue el último extractor que se ha usado para concluir con esta fase de experimentación. La arquitectura de esta CNN se basa en hacer una red híbrida entre *Inception* y *ResNet*. Durante su entrenamiento, llamó la atención que fue una de las que más tardó en ejecutarse, llegando a la cantidad de 44 horas en la instancia albergada en la nube, lo que se hubiera traducido en 109 horas aproximadamente en nuestro ordenador en local.

Finalmente se consiguió ejecutar el modelo y generar las gráficas correspondientes, las cuales se pueden ver en la Figura 5.11.

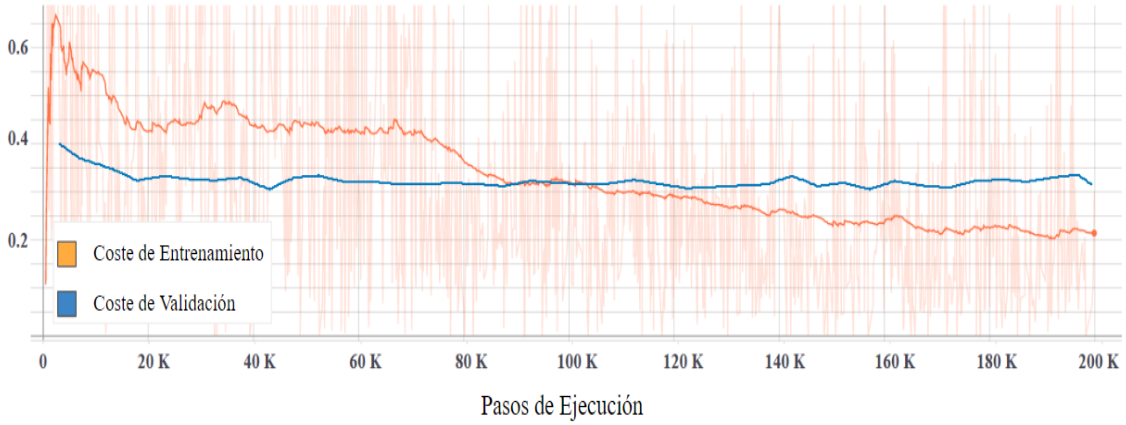


Figura 5.11: Curvas de Aprendizaje de *Faster R-CNN* con *Inception-ResNet*

Como se puede ver en ella, se volvieron a ejecutar 200000 pasos de ejecución en los que el coste volvió a ser similar a los modelos probados con *Faster R-CNN*. Pese a que el coste de validación se mantuvo bastante lineal durante todo el entrenamiento, el de entrenamiento fue cada vez siendo menor, llegando a cruzarse alrededor de los 100000 pasos de ejecución y teniendo como punto común más bajo el instante de los 123000 pasos de ejecución.

Los resultados obtenidos en cuanto a precisión media en la Tabla 5.33 son prometedores. Si se comparan con todos los que se han obtenido hasta el momento se podrá ver que son los mejores. En comparación con SSD e *Inception*, se puede ver que la precisión media aumenta un 19 % y casi un 28 % para el caso en el que se impone la restricción de IOU en 0.75.

Tabla 5.33: Precisión media usando *Faster R-CNN* con *Inception-ResNet*

mAP (%)	**mAP@.50 IOU (%)**	**mAP@.75 IOU (%)**
54.37	87.79	61.36

Con respecto a *Faster R-CNN* e *Inception*, se obtienen muy buenos resultados también, produciéndose un incremento del 20 % teniendo un 0.75 de IOU y casi un 12 % en la precisión media.

Atendiendo a la precisión con respecto a la distancia (Tabla 5.34, se ve que donde se produce un mayor rendimiento en comparación con SSD e *Inception* y *Faster R-CNN* e *Inception* es a media distancia, donde aumenta un 22 % y un 15 % respectivamente.

Tabla 5.34: Precisión media en distintas distancias usando *Faster R-CNN* con *Inception-ResNet*

mAP *Small* (%)	**mAP *Medium* (%)**	**mAP *Large* (%)**
20.40	53.12	67.07

Se procede ahora a mostrar los resultados que se han obtenido de exhaustividad. Como se puede apreciar en las Tablas 5.35 y 5.36, los resultados vuelven a aumentar considerablemente. Se refleja un claro incremento sobre todo en la métrica AR@100 y AR Small, con respecto a las Tablas 5.23 y 5.24.

Tabla 5.35: Exhaustividad media usando *Faster R-CNN* con *Inception-ResNet*

AR@1 (%)	**AR@10 (%)**	**AR@100 (%)**
17.61	51.98	60.62

Tabla 5.36: Exhaustividad media en distintas distancias usando *Faster R-CNN* con *Inception-ResNet*

AR *Small* (%)	**AR *Medium* (%)**	**AR *Large* (%)**
43.26	59.34	72.30

No solo ahí consiguen mejorarse los resultados, sino que en comparación con las Tablas 5.11 y 5.12, pertenecientes al detector SSD, el incremento es aún más acusado.

Para asegurarse de que estos resultados pueden ser fiables, se va a ver qué rendimiento tiene el modelo sin tener en cuenta los últimos *checkpoints*:

En la Tabla 5.37 se puede ver que la precisión obtenida no es de las mejores, por lo que se pueden producir una cantidad considerable de falsos positivos. No obstante, tiene una exhaustividad muy alta haciendo que este modelo no deje sin detectar una gran cantidad de rostros. Estas medidas quedan reflejadas claramente en la columna de los falsos positivos y falsos negativos, donde se ve que, en cuanto a falsos positivos, se producen casi 1100 más que en el caso de *Faster R-CNN* con *Inception*. A su vez, se produce la mitad de falsos negativos que en dicho modelo. Estos resultados quedan reflejados en su exactitud, haciendo que sea el mejor con casi un 75 %.

Tabla 5.37: Resultados más específicos usando *Faster R-CNN* con *Inception-ResNet*

Medida	**mAP (%)**	**AR (%)**	**VP**	**FN**	**FP**	**VN**	**Exactitud (%)**
@.50 IOU	79.65	93.72	11014	737	2814	0	75.61
@.55 IOU	79.62	93.69	11010	741	2817	0	75.57
@.60 IOU	79.62	93.66	11006	745	2817	0	75.54
@.65 IOU	79.61	93.63	11003	748	2817	0	75.52
@.70 IOU	79.54	93.54	10992	759	2827	0	75.40
@.75 IOU	79.47	93.45	10982	769	2837	0	75.28
@.80 IOU	79.39	93.36	10971	780	2848	0	75.14
@.85 IOU	79.22	93.11	10942	809	2869	0	74.84
@.90 IOU	78.95	92.56	10877	874	2899	0	74.24
@.95 IOU	77.41	89.98	10574	1177	3085	0	71.27
Media	79.24	93.07	10937	814	2863	0	74.83

Se observa ahora cómo se comporta con vídeos y en plataformas de *streaming*.

En la Tabla 5.38 se observa que su rendimiento en vídeos es bastante malo, procesando un fotograma por segundo y, por tanto, obteniendo 1 FPS como resultado. En cuanto a *streaming*, se puede ver que se obtienen unos resultados bastantes malos también, por lo que se podría concluir que este modelo no puede ser usado en *streaming*. No obstante, pese a los resultados obtenidos en vídeos, es un modelo que tiene una exactitud bastante buena y debe tenerse en cuenta.

Tabla 5.38: Resultados usando el modelo *Faster R-CNN* con *Inception-ResNet* en vídeos y *streaming*

Plataforma	Fotogramas	T. Inferencia	FPS
Vídeo	3398	0.938 segundos	1
Webcam	3398	0.892 segundos	1
YouTube	3398	0.948 segundos	0.3
Twitch	3398	0.943 segundos	0.5

En la Figura 5.12 se reflejan algunos ejemplos de como actúa este detector. En las imágenes se aprecia cómo es capaz de detectar a personas bajo unas condiciones de luz escasas y algunas letras tapándoles la cara. Además, es capaz de encontrar al conductor del coche y la cara de otra persona pese a tenerla tapada y pintada de colores.

Figura 5.12: Ejemplos de detección usando *Faster R-CNN* con *Inception-ResNet*

5.2.4. Resultados de la Fase 2

Como se ha podido ver en el análisis de esta segunda fase, los modelos basadas en *Faster R-CNN* actúan de una forma más lenta en cuanto a tiempo de inferencia se refiere, llegando en algunos casos a procesar un fotograma por segundo.

Sin embargo, queda demostrado que son las más efectivas a la hora de realizar la detección y su rendimiento en este sentido es mucho mejor, teniendo como máxima exactitud un 75 % aproximadamente en el caso de *Inception-ResNet*.

En primer lugar, el modelo basado en *ResNet* volvería a quedar descartado puesto que el rendimiento que ofrece tanto en tiempo como en exactitud no es lo suficientemente bueno como para utilizarlo en este caso.

Por otro lado, se tendría a los modelos basados en *Inception* e *Inception-ResNet*. *Inception* ha demostrado ser bastante bueno en exactitud con un 74.59 %, una cantidad de falsos positivos y falsos negativos equilibrada y un tiempo de inferencia en *streaming* semejante al mejor modelo obtenido en la primera fase. En cambio, se ha visto que *Inception-ResNet* es el mejor modelo que se ha podido crear con un 74.83 % de exactitud y un escaso número de falsos negativos que permitirían realizar un mayor número de detecciones. Además, pese a ser bastante lento en la inferencia, se podría aprovechar esta virtud para la detección en vídeos, puesto que en esa situación no es crucial el tiempo que se tarda en procesar un fotograma.

Por tanto, se podría concluir que estos dos últimos modelos comentados son los mejores de esta segunda fase.

5.2.5. Elección del Mejor Modelo

Dado que la experimentación se ha centrado en diferenciar los modelos no solo por su exactitud, sino también por su rendimiento en vídeos y *streaming*, lo correcto sería presentar un modelo por cada uno de estos dos usos. Se elegirá un modelo que esté mejor adaptado para vídeos y otro cuyo rendimiento en *streaming* sobresalga respecto al resto. Para ello, se compararán aquellos que se han propuesto como vencedores de cada una de las dos fases, mostrando su exactitud y los FPS de aquella plataforma en la que mejor hayan rendido. En este caso, los modelos se han comportado mejor con la *webcam*

En la Figura 5.13 se puede ver de manera muy resumida el comportamiento de los tres modelos elegidos. En ella se observa que la diferencia de FPS entre los modelos que usan sólo *Inception* no es muy grande, permitiendo que puedan usarse ambos en tiempo real. Esto no sucede así con el caso del último modelo, donde los resultados en este sentido dejan bastante que desear.

Sin embargo, en el caso de la exactitud, se observa que *Faster R-CNN* con *Inception* ha obtenido mejores resultados que su versión con SSD. Además, se observa también que son un poco inferiores a los que *Inception-ResNet* podría proporcionar.

Por tanto, por todo lo analizado en este capítulo, podría concluirse que el modelo más apropiado para usar en vídeos es *Faster R-CNN con Inception-ResNet*, mientras que se propone usar *Faster R-CNN* con *Inception* en el caso de *streaming*.

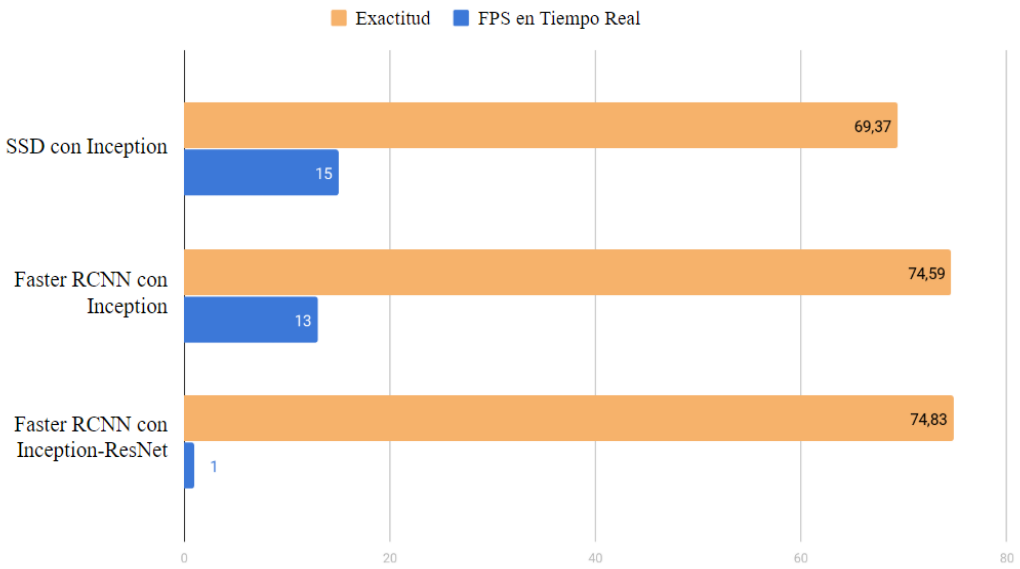


Figura 5.13: Comparativa final de los mejores modelos creados

Capítulo 6

Aportaciones Individuales

6.1. Arturo Barbero Pérez

Pese a haber cursado la asignatura de Aprendizaje Automático en la universidad, este proyecto supuso para mí un reto puesto que dicha asignatura no fue suficiente para entender la problemática que teníamos delante y a la que nos estábamos enfrentando. Las tareas en las que me he visto involucrado y aportaciones que he realizado al proyecto son las siguientes:

Lo primero que hice cuando conocí a mis compañeros fue proporcionarles un repositorio con los apuntes y prácticas que había realizado durante la asignatura comentada previamente. Dado que era la única persona del grupo con nociones en ML, decidí llevar a cabo esto con el objetivo de que pudieran entender este campo con la mayor brevedad posible y tuviéramos todos el mismo nivel a la hora de trabajar juntos.

Poco después me di cuenta de que, para aligerar el avance del proyecto, necesitábamos un método para que todos supiéramos qué cosas se han intentado y qué conocimiento ha adquirido el grupo en general. Por ello, abrí una unidad compartida en *Google Drive* donde añadí unos documentos para que todos apuntáramos los descubrimientos que íbamos haciendo y conceptos difíciles que hubiéramos entendido hasta el momento. Además, hice una pequeña investigación sobre la bibliografía recomendada y la añadí al *Drive*. Con el tiempo, el equipo fuimos subiendo artículos científicos y resúmenes que nos servían a todos.

Para entender más la problemática que teníamos ante nosotros, realicé una serie de tutoriales que estaban en la línea de este proyecto. Entre ellos se encontraban un software que permitía clasificar imágenes de perros y gatos, y otro que permitía realizar una detección mediante webcam diferenciando cuando un rostro que salía en escena era real o no. Los tutoriales me sirvieron para afianzar los conocimientos de ML que ya tenía y empezar a introducirme en el campo de CV.

En paralelo a estos tutoriales, fui haciendo una lectura y subrayado de los conceptos más importantes que encontraba en unos trabajos científicos que nos proporcionaron nuestros tutores, intentando relacionar estos conceptos con los tutoriales que realizaba, para poder explicárselos a mis compañeros y tutores en reuniones posteriores.

Dado que había realizado un tutorial muy relacionado con nuestra problemática, nuestro tutor propuso dividir el grupo para que yo fuera investigando más en la parte

de la implementación. Esta fase duró gran parte del primer cuatrimestre y, en ella, intenté desarrollar algunas Redes Convolucionales y Recurrentes desde cero con la librería de *Keras*. Esto fue una tarea bastante dura, dado que se necesita mucha más experiencia de la que teníamos para poder crear algo verdaderamente funcional.

Con el tiempo, desarrollé una Red Convolucional que funcionaba, aunque de forma muy imprecisa. No conseguí realizar el resto de redes necesarias para la creación de un detector. Como no había muchos avances, nuestro tutor indicó que tratase de investigar en la creación de una LSTM que permitiera realizar la detección. Esto tampoco pudo ser posible debido a la escasa información que hay actualmente sobre estas redes en el ámbito de la detección.

Para que pudiese realizar todas estas pruebas, tuvimos que buscar un conjunto de datos. Junto con mis compañeros, decidimos que debía utilizar un conjunto que fuera conocido en el estado del arte, por lo que finalmente elegimos YF. Para efectuar las pruebas, desarrollé un *script* que permitía procesar el conjunto de datos e introducirlo en las redes. Finalmente, este quedó descartado debido a que ocupaba alrededor de 20 GB y no teníamos medios en ese momento para procesar semejante cantidad de información.

Llegó un punto en el que el equipo se estancó, debido a que no conseguíamos prosperar con ninguna de las vías que estábamos tomando. Al ver esto, tomé la decisión de realizar unos resúmenes que permitieran comparar, de una manera más visual, lo que se había hecho hasta la fecha en otros proyectos y lo que estábamos haciendo nosotros, proporcionando algunas ideas y tecnologías nuevas que podíamos utilizar y exponiendo los pros y contras de todo ello. Estos resúmenes sirvieron durante un gran número de reuniones que tenía el equipo y fueron los cimientos para probar cosas nuevas que permitieron seguir avanzando.

En esta nueva fase, encontramos el término de TL y la API de *TensorFlow*. Investigué sobre esta API y me di cuenta de que podíamos crear un detector utilizando para ello cualquier tipo de red neuronal de *TensorFlow* que permitiera extraer las características de una imagen. A su vez, descubrí que también nos daba la opción de integrar un extractor de características que hubiéramos construido nosotros. Por ello, comencé a investigar cómo podríamos realizar TL sobre un extractor que desarrolláramos nosotros, e integrarlo con la API. La información que encontré, se la transmití a mis compañeros y Alejandro se puso también a investigar e intentar desarrollar la idea.

En este punto, empecé a realizar la instalación de todo el software necesario para utilizar la API, lo cual en un principio fue duro dado que hay muchas dependencias y no todas ellas están bien especificadas en internet. A la vez que lo instalaba, investigué sobre otros tipos de extractores de características que nos permitieran realizar una experimentación más amplia y objetiva. En esta investigación encontré que, a día de hoy, se pueden utilizar redes neuronales creadas por empresas como Google, sobre las que se pueden realizar técnicas de TL. Durante la lectura y entendimiento del funcionamiento de la API, me di cuenta de que el formato que íbamos a necesitar para los datos era especial. Por ello, mis compañeros y yo comenzamos a buscar conjuntos de datos de nuevo y encontramos el de Wider. Aún así, seguíamos teniendo el problema de conseguir uno con el formato requerido, pero rápidamente encontré en GitHub un repositorio que permitía automatizar el proceso de conversión del formato.

Debido a la situación que generó el COVID-19, el equipo se dispersó un poco —como es obvio— y las reuniones y avances que se estaban produciendo comenzaron a ser menos frecuentes. Por ello, investigué sobre los distintos extractores que podríamos utilizar y realicé un pequeño diagrama de Gantt con todas las tareas que quedaban por hacer,

teniendo en cuenta las posibles prórrogas del estado de alarma y, por tanto, de la entrega del proyecto. Gantt nos permitió ver de una mejor forma el tiempo que nos quedaba y volvimos a trabajar en el proyecto de nuevo.

Con toda esa información, reuní al equipo donde les expliqué mi propuesta para los experimentos del proyecto y comenzamos a asignar las tareas restantes. Discutimos la propuesta y cada uno expuso su opinión e ideas para llevar a cabo la experimentación.

Cuando todos acordamos como iba a ser la experimentación, me dispuse a crear una configuración para todos los modelos que iban a ser probados, de forma que todos estuviesen en las mismas condiciones para ser juzgados objetivamente. Aquí me di cuenta de que sería necesario seleccionar un tipo de métricas para realizar los entrenamientos. Después de investigar lo suficiente, decidí usar las que se describen en la sección 4.3, puesto que eran las que más información nos iban a proporcionar sobre los modelos y las más conocidas en competiciones donde se realizan grandes modelos de detección.

Una vez cree la configuración de todos los modelos, comencé a entrenar el primero de ellos (5.2.1.1) y con ello vi que era necesario tener otra plataforma en la que pudiéramos realizar los entrenamientos, dado que la primera prueba que realicé tardó 5 días en terminar.

En este punto, me di cuenta de que sería una buena oportunidad para aplicar lo que aprendí en la asignatura de *Cloud y Big Data* y utilizar alguna instancia en la nube para realizar los entrenamientos. Tras contarle la situación a mis compañeros y tutores, se nos proporcionó una instancia en *Google Cloud* y con ello conseguimos realizar los entrenamientos incluso 5 veces más rápido en algunos de los casos.

Dado que la instalación de la API no fue trivial, tuvimos una reunión todos y en ella traté de explicarles cómo tenía estructurado el proyecto, los pasos que seguí para realizar la instalación en Windows y los que habría que seguir para realizarla en un sistema basado en Linux, que era el que usaba nuestra instancia en la nube.

Mientras ejecutaba el resto de entrenamientos en *Google Cloud* y monitorizaba con *TensorBoard* tanto el progreso de aprendizaje, como los tiempos de ejecución; pasé a crear un *script* que nos permitiera realizar la detección en rostros usando los modelos que estaba generando con la instancia. Sobre este *script*, posteriormente, mi compañero José, añadió todo lo necesario para la detección de rostros en *streaming*.

Cuando terminó la ejecución de todos los modelos, pasé a analizar los resultados que se habían obtenido con cada uno de ellos y fui apuntándolo todo para pasarlo posteriormente a la memoria. A su vez, desarrollé un script que nos permitió realizar la detección sobre imágenes. Este *script* nos sirvió para poder adjuntar algunos ejemplos más visuales del rendimiento de los modelos en la memoria.

Dado que la implementación del extractor de características que estaba realizando Alejandro se estaba dificultando, intenté ayudarle a la hora de integrarlo con la API en la instancia de *Google Cloud*, pero no fue posible debido a diversas incompatibilidades. De la misma forma, haciendo pruebas con la detección en *streaming* encontré un problema en ello y, tras intentar solucionarlo sin mucho éxito, se lo comuniqué a mis compañeros y José finalmente pudo arreglarlo.

Transversalmente a estas tareas, realicé la completa escritura y creación de figuras de los capítulos 2 y 5, y de las secciones 4.2 y 4.3. Además de ello, he tratado de corregir algunos apartados y revisar la memoria en busca de errores, al igual que el resto de mis compañeros.

6.2. Alejandro Cabezas Garríguez

Las aportaciones personales que he realizado a lo largo del desarrollo de este trabajo han sido las siguientes:

- Realización de un curso *online* de programación con *Python* para aprender los fundamentos de este lenguaje de programación. Con este curso, aprendería todos los aspectos importantes que envuelven este lenguaje, los cuales usaría para construir los extractores y modelos de los que se ha hablado o se hablará en este apartado.
- Investigación personal para conocer y asentar las bases sobre los campos estudiados y aplicados a lo largo de este proyecto para así poder entender mejor los documentos académicos que se estudiarían durante la fase de investigación.
- Realización de un diagrama de Gantt inicial para estimar la duración de todas las fases que iban a transcurrir a lo largo del proyecto. Dicho diagrama fue reconstruido para mejorar su diseño y fue incuido en el capítulo uno de esta memoria.
- Investigación de numerosos trabajos para la aportación de ideas sobre el desarrollo de la idea. Tras leer cada artículo, realizaba un resumen sobre las ideas principales, el cual almacenaba en una carpeta en la nube. De la misma forma, almacenábamos los artículos ya leídos para evitar su relectura. En lugar de realizar una segunda lectura, se optaba por leer el resumen realizado. De esta manera conseguíamos una dinámica de aprendizaje más eficiente.
- Búsqueda e investigación sobre posibles candidatos a ser *datasets* para nuestra implementación. Para evaluarlo, observaba el número de muestras y si otros proyectos ya realizados habían aplicado dicho *datasets*. Si este había sido nombrado en algún artículo, observaba el objetivo de dicho artículo para comprobar si el *dataset* en cuestión podía ser compatible con nuestro objetivo, porque, por ejemplo, podría ser que el *dataset* no contuviese rostros. También comprobaba si el *dataset* era accesible, es decir, si existían enlaces a los que poder acceder para poder extraer los datos de ejemplo.
- Investigación sobre otras implementaciones ya realizadas para poder aplicar dichas técnicas a nuestra implementación. Esto envuelve la investigación de otros proyectos finales de grado realizados sobre una temática parecida, con el objetivo de poder crear unas bases sólidas para la fase de desarrollo.
- Cursé y estudié la asignatura de *Aprendizaje Automático y Big Data* impartida en esta universidad con el objetivo de, no solo aprender más sobre el mundo de la AI, si no también para aplicar dichos conocimientos al desarrollo y experimentación de los modelos y a desarrollo conceptual de la memoria.
- Realización de numerosos tutoriales en los que se creaban modelos muy simples para la clasificación de objetos, con el objetivo de poder aprender las técnicas y *frameworks* utilizados y poder aplicarlos en la posterior fase de desarrollo.
- Realización de resúmenes sobre los tutoriales cursados, con el objetivo de mostrar los problemas que iban surgiendo a medida que realizaba el ejercicio y que en principio no debían pasar. Además de los problemas, también apuntaba las posibles soluciones que encontraba en los distintos portales web. Por último, exponía las ideas que había

aprendido para que mis compañeros conocieran de antemano todos los conceptos que iban a aprender y a los que se iban a enfrentar.

- Implementación de un extractor de características propio para su posterior uso a la hora de la identificación de rostros. Dicho extractor se crearía aplicándole técnicas de TL para que consiguiera extraer características de rostros de manera eficiente. Aunque el extractor pudo implementarse con éxito, la incompatibilidad que existía con la librería que usamos para realización del detector hizo imposible el uso de dicho extractor, por lo que se descartó la idea.

- Para poder entrenar de manera eficiente y sin sobreajuste el extractor explicado anteriormente, realicé un *script* que crearía un conjunto de entrenamiento nuevo a partir de varios conjuntos ya existentes, puesto que no encontré un *dataset* que incluyese lo que buscaba. Dicho *script* extraía un porcentaje de imágenes de cada conjunto y lo clasificaba para el entrenamiento propiamente dicho, la validación y el *testing*. El resultado obtenido fue un *dataset* en el que cierto porcentaje eran imágenes de rostros y otro porcentaje eran objetos o animales, pero no caras propiamente dichas.

- Implementación de un modelo desde cero usando los métodos de las librerías usadas para la codificación del detector. Dicho modelo implementaba un extractor parecido al creado anteriormente y descartado. Debido a problemas de compatibilidad entre distintos componentes y, tras una exhaustiva investigación para encontrar el error que se generaba, se descartó la idea ya que no logramos que funcionase.

- Aportación de ideas durante la fase de experimentación, como los parámetros a evaluar para realizar las mediciones o los conjuntos que se iban a utilizar en el entrenamiento. También participé en la toma de decisión de los mejores modelos.

- Realización de varios capítulos de esta memoria, entre los cuales se encuentran el capítulo uno que consiste en la introducción, el capítulo seis, que consiste en las conclusiones y el trabajo futuro. También he desarrollado los capítulos escritos en inglés, que consisten en la introducción y las conclusiones.

- Aportación de contenido a los distintos capítulos, ayudando en la medida de los posible a los otros compañeros cuando así lo requerían. También he realizado la revisión de todos los capítulos de la memoria buscando posibles errores ortográficos o incongruencias.

- Realización, en conjunto con los demás compañeros, de la presentación posterior a la entrega de esta memoria usada para la defensa del trabajo.

- Realización en conjunto del manual de instrucciones de instalación del código, para así poder ejecutar los *scripts* creados anteriormente para poder realizar las detecciones. De igual forma, configuré un contenedor de *Docker* para poder simplificar la instalación en Linux.

6.3. José Morcuende Sierra

Durante el desarrollo de este trabajo he realizado diversas tareas, sobre todo de investigación y realizado de pruebas.

Lo primero de todo fue aprender *Python*, ya que personalmente no había usado este lenguaje. Por suerte es un lenguaje con una sintaxis relativamente sencilla a la que uno se acostumbra rápidamente. A continuación y relacionado con esto, realice una investigación a grandes rasgos sobre la AI debido a que tampoco tenia conocimientos previos y eran necesarios para comprender los trabajos de la fase de investigación. Además recibimos una pequeña formación sobre los conceptos básicos y los tipos principales de redes.

Lo siguiente que realice fue analizar el estado del arte. Los tutores nos proporcionaron una serie de trabajos para comenzar y nos explicaron el funcionamiento de *Google Scholar*. De los trabajos que nos dieron nos costó al principio entender muchos conceptos, por ello, en la unidad compartida del equipo en *Google Drive* íbamos guardando tanto resúmenes como los propios trabajos con las frases subrayadas que no comprendíamos, para posteriormente ponerlas en común lo aprendido por cada uno y determinar que podíamos usar para nuestra implementación, además, de tratar de solventar las dudas en las reuniones con los tutores.

Algunos de estos trabajos proporcionaban el código en un repositorio, de modo que opté a intentar replicarlos. Esta tarea me llevó más de lo que esperaba ya que las aplicaciones requerían de versiones especificas de las distintas librerías que, además de estar cambiando continuamente por los avances que se hacen el el campo del ML, no son compatibles entre ellas y requieren de componentes *hardware* con altas prestaciones de computo, como tarjetas gráficas, que en el caso de mi equipo al ser portátil no es ni muy potente ni moderna. Aun así, el poder ver la implementación ayudó a comprender como realizar ciertas tareas comunes que podríamos implementar en nuestro trabajo.

Tras ver que las implementaciones de los trabajos eran complicadas de replicar, decidí realizar varios tutoriales más acotados, sobre todo los relacionados con la detección de objetos con *OpenCV*. Empezando con los más básicos como detectar un objeto en una imagen, en este caso fue uno para reconocer si en la imagen aparecía un plato de macarrones. Posteriormente pase a los que se basaban en diferenciar entre varios objetos, como entre perros y gatos, viendo como para una imagen se le asignan los pesos a cada una de las opciones. Para acabar investigue más los detecciones a través de la *webcam* con los clasificadores proporcionados por *OpenCV* para reconocer rostros de frente y de perfil. Esto me sirvió para comprender el ciclo de obtener un fotograma de un vídeo, como procesarlo para extraer las características para posteriormente pintar las detecciones en una copia de ese fotograma e ir construyendo un vídeo con los resultados.

Una vez visto que los clasificadores de *OpenCV* no proporcionaban muy buenos resultados, instigué *framekorks* y modelos ya creados. Estos modelos están preparados para detectar varios objetos, pero no rostros en concreto, por ello estudiamos la posibilidad de realizar realizar TL sobre uno de estos para que se especifique en la detección de caras. La primera prueba fue sobre *Mask R-CNN* [HGDG17], usando uno de los ejemplos que proponen para detectar globos, cambié los métodos para que a la hora de evaluar las etiquetas, en vez de los globos fueran rostros, por lo que etiqueté unas 100 imágenes para realizar la prueba. Dado que el proceso de TL requiere reentrenar el modelo lo que en mi ordenador iba a llevar mucho tiempo. Por eso adapte el proyecto para que se pudiera usar desde remoto, en concreto desde una maquina de *Google Colab*, que permite activar

la aceleración por GPU. Los resultados eran bastante buenos a pesar de tener muy pocos datos de entrenamiento, con esto vimos que el hacer TL era una opción muy viable. Otra prueba que realice en base a este proyecto fue de reconocimiento facial con mi cara, añadiendo más fotos con la etiqueta 'jose' y adaptando la implementación. Pese a que el reconocimiento no fuera la finalidad de este trabajo surgió la idea de intentar realizar la identificación de los miembros del equipo, pero debido a la diferencia que habría con la cantidad de fotos de los conjuntos de datos, no iba a ser muy eficaz por lo que decidimos no incluirlo.

Tras decidir con el resto del equipo que se iba a usar *Keras* y la API de *TensorFlow*, una de las tareas que surgió era buscar conjuntos de imágenes compatibles. Tras ver que con los parámetros que requería *TensorFlow* solo encontramos un repositorio que automatizaba el proceso para el conjunto de Wider Face [YLLT16], decidí investigar otro candidato al que hacer esta transformación de manera manual. El mejor de los candidatos que encontré fue LFW [HMBLM08], que a pesar de no estar pensado para la detección, vimos en los trabajos analizados que con el se obtenían buenos resultados. Para el proceso de adaptación, realicé el etiquetado a mano de todas las imágenes, para posteriormente transformarlas en formato que requerido pasando todas las fotos del conjunto de imágenes a un solo directorio, realizando el etiquetado con *labelimg* [lab17] en XML pasarlo a CSV para finalmente obtener los archivos *TFRecord* requeridos para el entrenamiento y la validación. Todo el proceso se explica de forma más detallada en la Sección 4.1.1. Los resultados de esta transformación los subí al *Google Drive* del equipo con dos configuraciones: una con una proporción de las imágenes de un 90 %/10 % para el entrenamiento y validación respectivamente; y otra al 80 %/20 %. Tras probar ambas con el modelo vimos que la segunda ofrecía mejores resultados, aunque mucho peores que con el otro conjunto de imágenes, por lo que decidimos no continuar usándolo para la experimentación y dejarlo como documentación para adaptar un conjunto de imágenes al formato de *TensorFlow*.

Por último, tras tener ya la implementación acabada, se nos propuso la idea de añadirle la opción de usar la *webcam*, vídeos de internet o emisiones en directo como entrada para realizar la detección sobre ellos. Gracias a los tutoriales realizados al inicio, el hacerlo compatible con la *webcam* fue inmediato, simplemente añadí lo que había realizado anteriormente en nuestro proyecto. Para los vídeos y las emisiones encontré dos librerías, *Pafy* y *Streamlink* descritas en la Sección 4.4.1, que me permitieron variar la entrada que recibe el proyecto para la detección. Para usar los distintos métodos de entrada la solución implementada fue variar el argumento de entrada donde se indicaba la ruta al fichero de vídeo (con -v) con la URL del vídeo de *Youtube* o de *Twitch* (con los argumentos -y y -t respectivamente) o directamente por -w para hacer uso de la *webcam*.

Paralelamente a lo descrito anteriormente, avancé en algunos de los apartados de este documento con la información que iba consiguiendo, no sin antes compartirla y contrastarla con el equipo, para además de saber con que estábamos cada uno, no introducir información repetida en la misma. Los principales capítulos en los que he trabajado han sido el Capitulo 3 y el resto del Capitulo 4, así como la búsqueda y el formateado de las referencias de la bibliografía que aparecen a lo largo de la memoria. Además, de realizar parte de las correcciones que nos iban sugiriendo nuestros tutores a medida que avanzábamos.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

Una vez que el desarrollo del Trabajo ha concluido, el equipo ha conseguido llegar a las siguientes conclusiones:

Tras la fase de investigación y los resultados obtenidos en los diferentes modelos, los mejores sistemas elegidos han dado un porcentaje de acierto del 74,59 % y 74,83 %, respectivamente. El equipo considera estos resultados bastante aceptables aunque mejorables, teniendo en cuenta, en primera instancia, que es la primera vez que se enfrentaban a un problema como el expuesto en este trabajo. El equipo ha conseguido varios modelos tanto para realizar detecciones en vídeos como en *streaming*. No han tenido en cuenta el preprocesamiento de las imágenes, al igual que han tenido muchos desafíos teóricos a lo largo del proceso y, sin embargo, han conseguido sistemas eficientes en condiciones de imagen pobres, detección de múltiples rostros por fotograma y una velocidad de procesamiento aceptable para poder realizar detecciones en tiempo real. Además, el equipo ha contado con el tiempo suficiente para asegurarse de que los modelos elegidos son los más eficientes, puesto que en la fase de experimentación ha podido realizar múltiples pruebas con diferentes combinaciones, las cuales no han dado los mismos resultados en comparación con los elegidos.

Por otro lado, el haber experimentado con una amplia variedad de arquitecturas, ha permitido que se vea de una manera más clara cuáles son los modelos que se deberían usar dependiendo de los objetivos que se están persiguiendo. Además de esto, los integrantes del grupo creen que puede ser un gran aporte para la comunidad científica el tener una referencia donde se comparen distintas técnicas para que cada persona utilice la que más le convenga. El equipo de trabajo cree firmemente en que los resultados, son bastante positivos teniendo en cuenta la dificultad del problema propuesto y también considerando que se partía de ninguna base, más allá de los conocimientos básicos sobre el tema.

Además, tras una exhaustiva investigación en la que se exploraron una parte de los conceptos que abarcan este área de estudio, muchos fueron los problemas para continuar y hacerle frente a los desafíos que se planteaban a diario, aunque el equipo ha sido capaz de hacerles frente de manera satisfactoria. El equipo de trabajo piensa, por consiguiente, que la complejidad que implica este área de investigación es mucho más de la que se puede imaginar a simple vista. Son muchos los factores a tener en cuenta a la hora de crear modelos precisos y es mucho el tiempo que es necesario invertir para conseguir

resultados fructíferos. Además, y más concretamente en el área de la detección facial, es necesario un conocimiento y experiencia previa para poder entender la problemática en su totalidad. Crear un modelo preciso no consiste únicamente en elegir capas y nodos de manera aleatoria, es necesaria una experimentación muy precisa para conseguir resultados dignos y acordes a los esperados. En lo que se refiere a este trabajo, aunque los resultados son sorprendentemente buenos, el equipo tiene la certeza de que podría ser mucho mejores si se conociese el área y contexto de trabajo y estudio, lo que implicaría una reducción de la complejidad y una mayor facilidad para construir un sistema altamente puntero.

Por último, tras la fase de experimentación, se descubrió la complejidad que supone encontrar el equilibrio entre la efectividad y la eficiencia de un modelo de detección. Lo que quiere decir esto es que cuando un sistema es altamente preciso y efectivo, su eficiencia disminuye exponencialmente, como se ha podido llegar a ver en los resultados obtenidos en el proceso de experimentación. En algunos casos, la eficiencia era tan baja que se procesaba solo una imagen por segundo, y este no es para nada el resultado esperado para conseguir análisis de vídeos de manera rápida. Por otro lado, si se consigue un modelo que procese las imágenes a una gran velocidad, este contará con una tasa de inexactitud más alta y con una precisión final más baja, consiguiéndose así un sistema poco efectivo aunque eficiente. Es por ello que nuestra elección final ha sido aquella que conseguía el equilibrio entre estas dos métricas, aparentemente contrarias e imposibles de explotar en su totalidad de manera simultánea. Aún así, como se explicará en el siguiente apartado, este equilibrio entre efectividad y eficiencia puede llegar a ser mejor de lo que se ha conseguido si se investiga suficientemente en la configuración de los modelos elegidos.

7.2. Trabajo Futuro

Como posibles trabajos futuros pueden señalarse los siguientes:

- **LSTM para la extracción de características**: Aunque durante toda la fase de experimentación el equipo estuvo probando con numerosas configuraciones de modelos, es cierto que no se probaron las técnicas usadas en el artículo académico [LLT17], relacionadas con el uso de LSTM, debido a la inexperiencia en implementar este tipo de sistemas por la complejidad que esto conllevaba, tanto a nivel teórico como a nivel técnico. Como se puede observar en el estado del arte de esta memoria situado en el capítulo 3, este artículo muestra una posible arquitectura aplicando este tipo de Red Neuronal, con la que se consiguen unas extracciones de características de mejor calidad. Es por ello que el equipo cree que podrían mejorar las tasas de aciertos de los modelos si se aplicaran dichas soluciones, y, por tanto, se considera que podría ser un buen punto de partida de cara a mejorar la precisión final de los modelos elegidos.

- **Mejora de los parámetros de configuración**: Aunque, como se ha explicado anteriormente, el equipo piensa que los resultados que se han obtenido son satisfactorios, y ha conseguido sistemas eficiente a la par que efectivos, el equipo cree que la configuración de estos modelos podría mejorarse de cara a conseguir un mejor equilibrio entre la rapidez de detección y la efectividad a la hora de realizarla. Por ello, se cree que un proceso de investigación sobre los parámetros de configuración de los modelos puede ser muy positivo de cara a mejorar la tasa de aciertos sin que esto afecte a la velocidad de procesamiento.

- **Preprocesado de las imágenes**: Tras realizar la experimentación, el equipo cree que es posible que, aunque se mitigue en cierta medida, el problema con la calidad de la imagen puede seguir siendo un factor que disminuya la efectividad de la detección. Si una imagen posee ciertos aspectos como baja iluminación, pixelación, difuminación o, simplemente, resolución pobre, la tarea de la detección del rostro puede suponer un gran reto incluso para una red muy puntera y bien entrenada.

  Es por ello que se piensa que la idea de un preprocesamiento de la imagen puede suponer un aumento significativo en la precisión y efectividad del modelo en cuestión. Dicho sistema de preprocesamiento se encargaría de calibrar ciertas propiedades de la imagen, como el brillo, el contraste o el color. También podría encargarse de añadir píxeles extras a la imagen final, para añadirle calidad a esta y poder así añadir más información para el modelo. Esto podría realizarse de manera programática, es decir, con algún tipo de algoritmo que realice la calibración en función de los parámetros iniciales de la imagen, o, por otro lado, se puede crear un modelo más que esté entrenado para aumentar la calidad final de la composición.

- **Incorporación de la estimación de la edad**: Tras realizar este trabajo, el equipo siente que el proyecto final realizado podría servir como un gran sistema de detección facial para el análisis forense. Se cree que los modelos finales pueden llegar a ser, si bien no perfectos, bastante certeros a la hora de realizar detecciones faciales en vídeo e, incluso, en *streaming*. Sin embargo, no se ha podido tiempo a crear un sistema que realice predicciones de la edad de manera certera y eficiente. Es posible que dicho sistema, en combinación con el trabajo final, pueda ser de mucha mayor utilidad para el análisis forense, puesto que, en numerables ocasiones, la edad de los sujetos implicados es determinante para poder aclarar la gravedad o la legalidad de los hechos ocurridos en el metraje. Es por ello que el equipo cree que una estimación de la edad puede ser un gran paso a realizar para poder conseguir un sistema plenamente capaz de ser usado para el análisis forense.

- **Etiquetado de las detecciones**: A la hora de realizar el análisis forense sobre vídeo es posible que sea necesario saber, no solo el o los rostros que aparecen en ella, sino también ciertas características sobre ellos que puedan ser cruciales para el proceso de investigación, como puede ser el nombre, el color de piel, la raza o, incluso, el color del pelo. Por ello, el equipo afirma que podría ser interesante añadir etiquetado en las detecciones de los rostros, dotando a los modelos de la capacidad de identificación. Actualmente, cuando los modelos finales consiguen detectar una cara en una imagen, esta aparece con la etiqueta de *Face*. Aunque dicha etiqueta deja bastante claro lo que está detectando, es cierto que podrían aparecer ciertas propiedades del rostro en cuestión, como un conjunto de etiquetas tales como *Face, blue eyes, caucasian* o algo relacionado. Añadir más información a la detección podría llegar a ser muy útil para, por ejemplo, saber en cuestión de muy poco tiempo si un individuo en concreto aparece en los metrajes observados por el modelo.

- **Software de soporte**: Tras la fase de experimentación, se contaba con los modelos más precisos para realizar las detecciones. Para crear las salidas del modelo, se codificaron diferentes *scripts* que permitían que el modelo consiguiese detectar los rostros en una imagen o un vídeo, el cual era la entrada de dichos *scripts*. Para la ejecución de estos, era necesario que el proceso se realizase en una terminal, puesto que no existía ningún soporte software para la ejecución de dichos *scripts*. Es por ello que el equipo cree oportuno el desarrollo de un componente software que permita

utilizar los modelos finales de este trabajo para realizar la detección podría ser muy interesante para poder así simplificar la complejidad que ahora supone poder realizar detección en un vídeo.

- **Más aplicaciones**: El objetivo que se ha mantenido desde el comienzo de este trabajo ha sido el de crear un sistema óptimo para la detección de rostros a la hora de realizar análisis forense sobre vídeos. Aunque este es un campo muy amplio que abarca multitud de utilidades concretas, el equipo cree que este sistema puede tener más aplicaciones en el mundo actual. Con la reciente situación que se está sufriendo a causa del COVID-19, una utilidad nueva podría ser el del control de las medidas de seguridad aplicadas para evitar el contagio. Concretamente, el sistema podría detectar rostros para así poder identificar, de manera manual con intervención humana o incluso también automática si se aplica el etiquetado en las detecciones, si el individuo está respetando las medidas de seguridad establecidas, específicamente si el rostro lleva consigo una mascarilla o no. Esto concienciaría a la sociedad de cara a respetar las medidas propuestas y así asegurar que se cumplen, consiguiendo así un impacto positivo en la evolución de la situación.

Resumen del Trabajo en Inglés

Capítulo 8

Introduction

8.1. Motivation

Over the past 70 years, numerous scientific advances have led to exponential growth in the world of technology. Specifically, in the audiovisual field, these developments have led to numerous advances that have had a positive impact on the quality, processing speed and storage capacity of images and audio. Until recently, a specialized camera was required to record a video and an innumerable amount of extra components for its correct reproduction. In addition, the quality obtained was much lower than today, resulting in a loss of important information when performing forensic analysis on video.

With the arrival of the digital era, digital cameras emerged and this managed to fix the problems discussed above. With them, no special equipment was needed and the quality obtained was notably superior. In addition, over time, new forms of storage emerged that allowed for longer-lasting video, making video forensics a more efficient and productive task.

Even so, the quality of the video was not as high as desired, and the task of identification was difficult when the quality of the environment in which the video was made was under very unfavorable conditions, such as dark places. With the advancement of software and the emergence of early image and video editing programs, footage could be modified to improve certain aspects of composition, such as brightness, saturation, contrast, and so on. Forensic analysis thus became an even more productive task, but there was still a problem that hindered its evolution: it required too much manual work and human intervention for the identification work, and this slowed down the process.

On the other hand, the field of AI was still under investigation. It was not until 1958 that the first neural network architecture appeared, [Sim20] Perceptron. Since then, neural networks and AI began to grow at great speed, largely thanks to the evolution of hardware that allowed the creation of more powerful machines capable of supporting the construction of more complex and efficient architectures.

Software development changed completely with the incorporation and development of Neural Networks. As their utility covers any task, many companies developed neural networks for the identification of objects in images to implement autonomous systems that react and make decisions in real time taking into account the environment in which they were located.

It might seem that the use of AI in forensic analysis would solve all the problems. This task would be completely automated, and extensive analysis would be performed in a matter of minutes and in a massive way. Image quality becomes a lesser priority issue, because systems analyze images differently than the human eye does, finding patterns and anomalies that are imperceptible to anyone. But, even so, a new type of problem arises: the need for a cutting-edge and extremely efficient network, to avoid erroneous identifications which, for the identification of objects and more specifically faces, is a very complex job.

Therefore, a good face identification system, which is able to make very reliable and accurate detections, as well as to make identifications in low quality videos, is crucial to be able to incorporate artificial intelligence in the forensic analysis in a full way and thus be able to dispense with human intervention, which is an incredible slowing down of the process. The search for such systems is the driving force behind this work.

8.2. Context

This Final Degree Project has been carried out within the Analysis, Security and Systems Group (GASS Group, https://gass.ucm.es/, Group 910623 of the catalogue of groups recognized by the UCM) as part of the activities of the research project THEIA (Techniques for Integrity and Authentication of Multimedia Files of MobileDevices) with reference FEI-EU-19-04

8.3. Object of the Investigation

A picture contains many simple features which can be seen by human eye, such as the silhouettes of objects or the color they have. These patterns together make it possible to identify objects in an image and therefore to draw conclusions about it, such as the number of people in it or the time of day it was made. However, it has been found that a sufficiently trained AI can identify certain patterns in images that are beyond human understanding, and these are critical in identifying objects or any other task involving the analysis of a photograph.

In addition, there are many already created and tested identifiers dedicated to identifying whole bodies, and not faces specifically, and this can be a disadvantage when a body is not fully displayed in an image.

This paper proposes different techniques, algorithms and DL models that try to make an efficient face identification in streaming images and videos.

8.4. Workplan

The development of this work has been carried out in four phases:

1. **Investigation**: During the first three months, a period of adaptation to the work context and acquisition of the necessary knowledge was carried out in order to start with the subsequent development. At the beginning, a general meeting was held in which the starting point of the work was explained, what the objectives were to be achieved and what the necessary knowledge was to be. After this, numerous meetings

were agreed upon with the directors to follow up on the progress of the research and to resolve any doubts that might arise. Another reason why these meetings were arranged was to receive talks on the basic concepts of the different fields involved in this work. These will be discussed in the following chapters. Some members of the team already had such knowledge thanks to the subjects of Artificial Intelligence and Machine Learning, taught in this Faculty. During the meetings, many tools that would facilitate the research work were also explained. One of these tools was 'Google Scholar', which the team would use to search for scientific articles on previous research in the same field of study. Finally, once several conclusions had been reached about what was wanted and how it was to be achieved, the team began to prioritise development.

2. **Developing**: Once the necessary knowledge was acquired to be able to start creating a solid version of the project, it was decided to spend less time on research and to start with the codification of the project. Although the research phase did not cease, it was rather put in the background, with research being carried out on specific concepts that emerged during development. Therefore, during this phase, advanced concepts of the *Python* programming language and libraries such as *Keras* and *TensorFlow* were investigated. During this phase the training sets for the model were also built, obtained from the sources provided by the articles read during the previous phase.

3. **Experimenting**: In this phase, the evaluation of the developed prototype was carried out and the results obtained were analysed. Different tools provided by the same libraries explained above were used, the results were compared with those obtained in the different works reviewed and the configuration of the parameters for the model refinement was carried out. For example, it was decided to adjust the training data sets to analyze their influence on the improvement of the results. It should be noted that the development phase did not cease during the experimental phase, since the model was constantly compared with others and modified for optimization.

4. **Documentation**: Halfway through the development, the team began the phase that would consist of writing and constantly reviewing the final project report. This phase ran in parallel between the development and experimentation stages. For the writing of the report, the team made constant updates of the content, as the development included some new technology, the experimentation concluded in different conclusions, or even new articles were investigated. After an update, the team reviewed the changes made for errors or possible sections to improve. Finally, this phase ended after completing the last milestone of the experimental stage, compiling all the results obtained from that phase and adding them to the Work.

In the Figure 8.1 you can see in more detail the organization that the team has followed during the course of this project, as well as the duration of each phase and the sub-tasks that compose them. Also, below is a table showing the milestones followed by the team followed by an identifier, used in the Figure 8.1 for better understanding.

Table 8.1: Table of milestones followed by the team followed by their identifiers

Id	Task
1	Investigation
1.1	Introduction to the working context
1.2	Individual learning of basic concepts
1.3	Introduction to tools and software
1.4	Research of academic articles
1.5	Extraction of conclusions and common ideas
2	Developing
2.1	Construction of datasets
2.2	Creation of the prototype (architecture)
2.3	Training of the models
3	Experimenting
3.1	Video experimenting
3.2	Optimization of parameters and inputs
3.3	Extraction of metrics and conclusions
4	Documentation
4.1	Report development

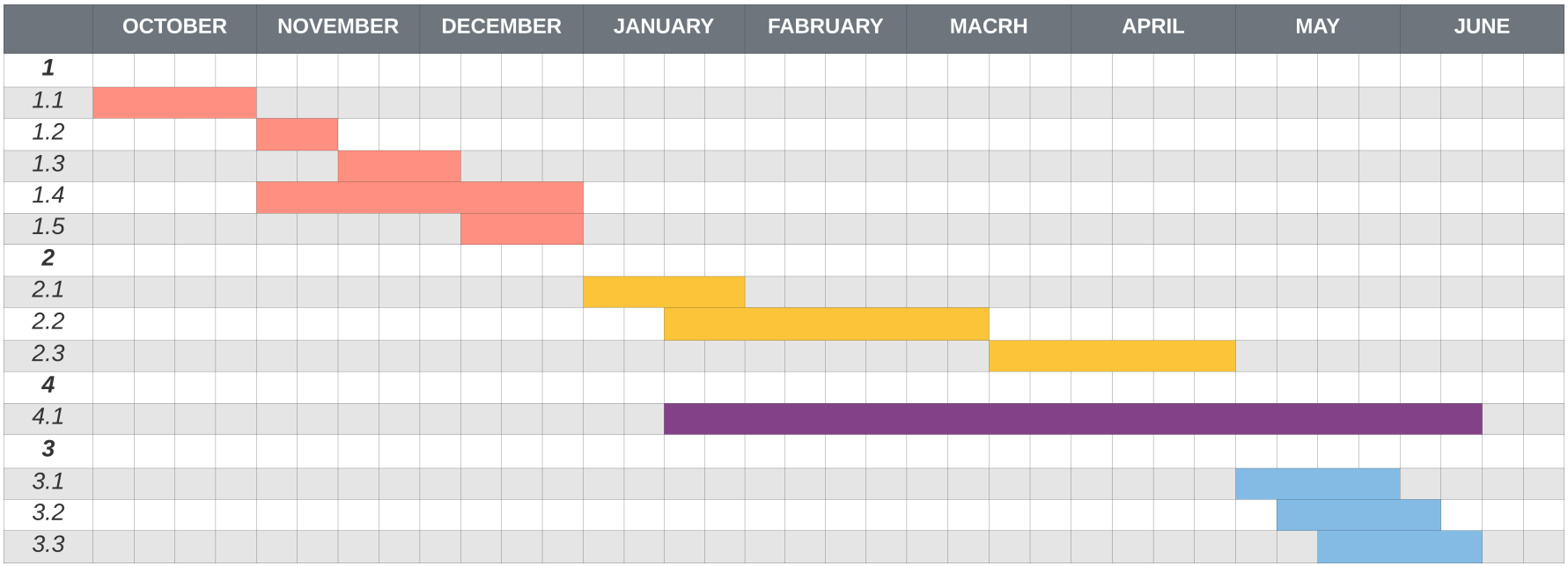


Figure 8.1: Gantt diagram followed by the team

8.5. Struture of the Work

The rest of the work is organized into 9 chapters with the structure described below:

Chapter 2 explains basic concepts both in the basics of AI and in object detection. Specifically, this chapter focuses both fields of study on the term that concerns this work, which is the detection of faces in digital images and in videos. Therefore, in this chapter we talk about the history that involves the AI as well as the techniques used and the different models of neural networks.

Chapter 3 begins by showing a state of the art in which different techniques and research carried out in the field of face identification in images are presented, as well as different architectures and models presented. These works represent the current situation in the field of face detection in images and videos. From each article, we extract both the idea and the techniques used for the development of this, as well as the training set used and the conclusions to be highlighted based on the results obtained. These results are stored in different tables to be able to obtain comparisons between the different works. Finally, the different articles are shared in order to draw conclusions about the current state of the object of study in this work, the identification of faces in digital images and videos.

Chapter 4 presents the contributions of this work. Thus, firstly, an extensive description of the technologies applied for the creation of the models is presented, which will be subsequently tested in subsequent chapters. It also explains in detail the different sets of images used in the training of the models and the different metrics used in the evaluations of the experimental phase.

Chapter 5 describes the experiments carried out to evaluate the effectiveness of the proposed models based on the technologies explained in Chapter 4 and presents the results obtained, as well as the variation of these results depending on the optimization of the initial configuration.

Chapter 7 shows the main conclusions of this work, the future lines of research.

Chapters 8 and 9 are the English translations of the Introduction and the Conclusions.

Finally, Chapter 6 describes the personal contributions of each member of the team to the development of this work.

Capítulo 9

Conclusions and Future Work

9.1. Conclusions

After the research phase and the results obtained in the different models, the best systems chosen have given a success rate of 74.59 % and 74.83 % respectively. The team considers these results to be quite acceptable, although they can be improved, taking into account, in first instance, that this is the first time they have faced a problem like the one described in this paper. The team has obtained several models for both video and streaming detections. They have not taken into account the pre-processing of the images, as they have had many theoretical challenges along the process and, nevertheless, they have achieved efficient systems in poor image conditions, detection of multiple faces per frame and an acceptable processing speed to be able to make detections in real time. In addition, the team has had enough time to ensure that the models are the most efficient ones, since in the experimentation phase they have been able to perform multiple tests with different combinations, which have not given the same results compared to the selected ones.

On the other hand, the fact of experimenting with a wide variety of architectures have allowed the team to see more clearly which models should be used depending on the objectives being pursued. Besides this, the members of the group believe that it can be a great contribution for the scientific community to have a reference where different techniques are compared so that each person can use the one that is most convenient for him/her. The work team firmly believes that the results are quite positive, considering the difficulty of the proposed problem and also considering that it was based on no basis, beyond basic knowledge on the subject.

Furthermore, after an exhaustive investigation in which a part of the concepts that cover this area of study were explored, many were the problems to continue and to face the challenges that were posed daily, although the team has been able to face them in a satisfactory way. The team therefore believes that the complexity involved in this area of research is much more than can be imagined at first glance. There are many factors to consider when creating accurate models and much time needs to be invested to achieve successful results. In addition, and more specifically in the area of face detection, previous knowledge and experience is necessary in order to understand the problem in its totality. Creating an accurate model is not just about choosing layers and nodes at random, it requires very precise experimentation to achieve worthy results in line with expectations.

As far as this work is concerned, although the results are surprisingly good, the team is certain that it could be much better if the area and context of work and study were known, which would imply a reduction in complexity and greater ease in building a highly advanced system.

Finally, after the experimental phase, the complexity of finding a balance between the effectiveness and efficiency of a detection model was discovered. What this means is that when a system is highly precise and effective, its efficiency decreases exponentially, as has been seen in the results obtained in the experimental process. In some cases, the efficiency was so low that only one image per second was processed, and this is not at all the expected result to achieve fast video analysis. On the other hand, if a model is achieved that processes the images at a high speed, it will have a higher inaccuracy rate and a lower final accuracy, thus achieving a not very effective but efficient system. That is why the final choice was the one that achieved the balance between these two metrics, apparently opposite and impossible to exploit in their entirety simultaneously. Even so, as will be explained in the following section, this balance between effectiveness and efficiency can be better than what has been achieved if enough research is done on the configuration of the chosen models.

9.2. Future Work

As possible future works following this investigation line, there are proposed the following ones:

- **LSTM for feature extraction**: Although during the entire experimental phase the team was testing with numerous model configurations, it is true that the techniques used in the academic article [LLT17], related to the use of glsLSTM, were not tested due to inexperience in implementing this type of system because of the complexity involved, both at a theoretical and technical level. As can be seen in the state of the art of this report located in the chapter Capitulo3, this article shows a possible architecture applying this type of Neural Network, with which better quality features can be extracted. Therefore, the team believes that the success rates of the models could be improved if such solutions were applied, and therefore it is considered that it could be a good starting point to improve the final accuracy of the chosen models.
- **Improvement of the configuration parameters**: Although, as explained above, the team thinks that the results obtained are satisfactory, and has achieved efficient systems at the same time as effective ones, the team believes that the configuration of these models could be improved in order to achieve a better balance between the speed of detection and the effectiveness when carrying it out. Therefore, it is believed that a research process on the configuration parameters of the models can be very positive in order to improve the success rate without affecting the processing speed.
- **Image pre-processing**: After experimentation, the team believes that it is possible that image quality problem could affect to the detection effectiveness. If an image has certain aspects such as low lighting, pixelation, blurring or simply poor resolution, the task of face detection can be very challenging even for a very advanced and well-trained network.

It is therefore thought that the idea of image pre-processing can significantly increase the accuracy and effectiveness of the model in question. This pre-processing system would be responsible for calibrating certain image properties such as brightness, contrast or colour. It could also add extra pixels to the final image, to add quality to the image and provide more information to the model. This could be done in a programmatic way, that is, with some kind of algorithm that performs the calibration according to the initial parameters of the image, or, on the other hand, you can create one more model that is trained to increase the final quality of the composition.

- **Incorporation of age estimation**: After doing this work, the team feels that the final product could serve as a great face detection system for forensic analysis. It is believed that the final models can be, although not perfect, quite accurate performing video and streaming face detection. However, there has been no time to create a system that makes accurate and efficient age predictions. It is possible that such a system, in combination with the final work, could be much more useful for forensic analysis, since, on a number of occasions, the age of the subjects involved is a determining factor in order to clarify the seriousness or legality of the events that occurred in the footage. That is why the team believes that an age estimation can be a great step forward to achieve a system fully capable of being used for forensic analysis.

- **Labeled detections**: When carrying out forensic analysis on video, it may be necessary to know not only the face or faces that appear on it, but also certain characteristics about them that may be crucial to the investigation process, such as name, skin colour, race or even hair colour. Therefore, the team says it could be interesting to add labelling to the face detections, giving the models the ability to identify them. Currently, when the final models manage to detect a face in an image, it appears with the label of *Face*. Although this label makes it quite clear what it is detecting, it is true that certain properties of the face in question could appear, such as a set of labels like *Face, blue eyes, caucasian* or something related. Adding more information to the detection could be very useful to, for example, know in a very short time if a particular individual appears in the footage observed by the model.

- **Supporting software**: After the experimental phase, the most accurate models were available to perform the detections. To create the outputs of the model, different text scripts were coded that allowed the model to detect the faces in an image or a video, which was the input of those text scripts. In order to execute them, it was necessary for the process to be carried out in a terminal, since there was no software support for the execution of these texts. That is why the team believes that the development of a software component that allows the use of the final models of this work to perform the detection could be very interesting to simplify the current complexity.

- **More applications**: The objective that has been maintained from the beginning of this work has been to create an optimal system for face detection when performing forensic analysis on videos. Although this is a very wide field that covers many specific utilities, the team believes that this system can have more applications in today's world. With the recent situation that is being suffered because of the COVID-19, a new utility could be the control of the security measures applied to avoid the contagion. Specifically, the system could detect faces in order to be able to identify, manually with human intervention or even automatically if labelling is applied in

the detections, whether the individual is respecting the established safety measures, specifically whether the face is wearing a mask or not. This would make society aware of the need to respect the proposed measures and thus ensure that they are complied with, thus achieving a positive impact on the evolution of the situation.

Bibliografía

[AZ13] R. Arandjelovic and A. Zisserman. All About VLAD. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1578–1585, 2013.

[BEP+08] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[Bre20] C. Breazeal. MIT Team Building Social Robot. http://news.mit.edu/2001/kismet, January 2020.

[Bro20a] J Browniee. How to use Learning Curves to Diagnose Machine Learning Model Performance. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/, January 2020.

[Bro20b] J. Brownlee. A Gentle Introduction to Transfer Learning for Deep Learning. https://machinelearningmastery.com/transfer-learning-for-deep-learning/, January 2020.

[CBL+16] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent Batch Normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[CCG+13] PL. Carrier, A. Courville, IJ. Goodfellow, M. Mirza, and Y. Bengio. FER-2013 Face Database. *Universit de Montral*, 2013.

[Cla20] L. Clark. Google's Artificial Brain Learns to Find Cat Videos. https://www.wired.com/2012/06/google-x-neural-network/, January 2020.

[CRW+14] D. Chen, S. Ren, Y. Wei, X. Cao, and J. Sun. Joint Cascade Face Detection and Alignment. In *European conference on computer vision*, pages 109–122. Springer, 2014.

[DAHG+15] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[DGG15] A. Dhall, R. Goecke, and T. Gedeon. Automatic Group Happiness Intensity Analysis. *IEEE Transactions on Affective Computing*, 6(1):13–26, 2015.

[DGJ+16] A. Dhall, R. Goecke, J. Joshi, J. Hoey, and T. Gedeon. Emotiw 2016: Video and Group-Level Emotion Recognition Challenges. In *Proceedings of the 18th ACM international conference on multimodal interaction*, pages 427–432, 2016.

[DGLG12] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. Collecting Large, Richly Annotated Facial-Expression Databases from Movies. *IEEE multimedia*, (3):34–41, 2012.

[F.18] Chollet F. *Deep Learning with Python*. Manning, 2018.

[FH99] H. Feng-Hsiung. IBM's Deep Blue Chess Grandmaster Chips. *IEEE Micro*, 19(2):70–81, March 1999.

[FLLL16] Y. Fan, X. Lu, D. Li, and Y. Liu. Video-Based Emotion Recognition Using CNN-RNN and C3D Hybrid Networks. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pages 445–450, 2016.

[GDDM14] R Girshick, J Donahue, T Darrell, and J Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[Gir15] R Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[GLF+08] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008.

[GWD14] A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

[HGDG17] K. He, G. Gkioxari, P Dollár, and R Girshick. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[HMBLM08] GB. Huang, M. Mattar, T. Berg, and E. Learned-Miller. Labeled Faces in the Wild: A Database For Studying Face Recognition in Unconstrained Environments. 2008.

[HOT06] G. E. Hinton, S. Osindero, and Y. Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527—1554, May 2006.

[HS97] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

[Hui18] J. Hui. mAP (mean Average Precision) for Object Detection. 2018.

[HZRS16] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JPD+11] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating Local Image Descriptors Into Compact Codes. *IEEE transactions on pattern analysis and machine intelligence*, 34(9):1704–1716, 2011.

[JSD+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[JZL+19] Licheng J., Fan Z., Fang L., Shuyuan Y., Lingling L., Zhixi F., and Rong Q. A Survey of Deep Learning-Based Object Detection. *IEEE Access*, 7:1—22, October 2019.

[KCL+15] S. Kwak, M. Cho, I. Laptev, J. Ponce, and C. Schmid. Unsupervised Object Discovery and Tracking in Video Collections. In *Proceedings of the IEEE international conference on computer vision*, pages 3173–3181, 2015.

[KOLW16] K. Kang, W. Ouyang, H. Li, and X. Wang. Object Detection from Video Tubelets with Convolutional Neural Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 817–825, 2016.

[KSH12] A. Krizhevsky, I. Sutskever, and GE. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[lab17] LabelImg. https://github.com/tzutalin/labelImg, October 2017.

[LAE+16a] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, pages 21—-37, November 2016.

[LAE+16b] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A.C. Berg. SSD: Single Shot Multibox Detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[LdMM17] R. López de Mántaras and P. Meseguer. *Inteligencia Artificial*, chapter 1, pages 7–29. CSIC Consejo Superior de Investigaciones Científicas, 2017.

[LGRYTN11] H. Lee, R. Grosse, R. Ranganath, and A. Yan-Tak Ng. Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks. *Communications of the ACM*, 54(10):95–103, October 2011.

[LHS+14] H. Li, G. Hua, X. Shen, Z. Lin, and J. Brandt. Eigen-Pep for Video Face Recognition. In *Asian conference on computer vision*, pages 17–33. Springer, 2014.

[LLT17] Y. Lu, C. Lu, and C. Tang. Online Video Object Detection Using Association LSTM. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2344–2352, 2017.

[LMB+20] TY Lin, M Maire, S Belongie, J Hays, P Perona, D Ramanan, P Dollár, L Zitnick, G Patterson, MR Ronchi, Y Cui, L Bourdev, and R Girshick. COCO - Common Objects in Context. http://cocodataset.org/#detection-eval, March 2020.

[LWZ11] J. Li, T. Wang, and Y. Zhang. Face Detection Using Surf Cascade. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 2183–2190. IEEE, 2011.

[LZLY14] L. Liu, L. Zhang, H. Liu, and S. Yan. Toward Large-Population Face Identification in Unconstrained Videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(11):1874–1884, 2014.

[MMdRM16] N. McLaughlin, J. Martinez del Rincon, and P. Miller. Recurrent Convolutional Network for Video-Based Person Re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1325–1334, 2016.

[num20] NumPy, The Fundamental Package for Scientific Computing with Python. https://numpy.org/, March 2020.

[NZH+17] G. Ning, Z. Zhang, C. Huang, X. Ren, H. Wang, C. Cai, and Z. He. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[Ola20] C Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 2020.

[ope20a] Open Computer Vision Library (OpenCV). https://opencv.org/, March 2020.

[ope20b] Open Images Dataset V6. https://storage.googleapis.com/openimages/web/challenge_overview.htmle, March 2020.

[paf20] Pafy - A Python library to download YouTube content and retrieve metadata. https://pythonhosted.org/pafy/, March 2020.

[pas20] The PASCAL Visual Object Classes Homepage. http://host.robots.ox.ac.uk/pascal/VOC/, March 2020.

[PLC+12] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari. Learning Object Class Detectors from Weakly Annotated Video. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3282–3289. IEEE, 2012.

[PSVZ14] OM. Parkhi, K. Simonyan, A. Vedaldi, and A. Zisserman. A Compact and Discriminative Face Track Descriptor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1693–1700, 2014.

[PVZ15] OM. Parkhi, A. Vedaldi, and A. Zisserman. Deep Face Recognition. 2015.

[RCWS14] S. Ren, X. Cao, Y. Wei, and J. Sun. Face Alignment at 3000 FPS Via Regressing Local Binary Features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1685–1692, 2014.

[RDGF16] J Redmon, S Divvala, R Girshick, and A Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RHGS15] S Ren, K He, R Girshick, and J Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[Sar20] D. Sarkar. A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. https://tinyurl.com/y7ehja7h, January 2020.

[Sch97] RR. Schaller. Moore's Law: Past, Present and Future. *IEEE Spectrum*, 34(6):52—59, June 1997.

[SF18] N. Shukla and K. Fricklas. *Machine Learning With TensorFlow*, chapter 1, pages 5–20. Manning, 2018.

[Sha18] R. Shanmugamani. *Deep Learning for Computer Vision*, chapter 1, pages 8–25. Packt, 2018.

[Sil20] V Silaparasetty. How to Handle Overfitting and Underfitting in Machine Learning. https://medium.com/datadriveninvestor/how-to-handle-overfitting-and-underfitting-470a1f7389fe, January 2020.

[Sim20] Simplilearn. What is Perceptron. https://www.simplilearn.com/what-is-perceptron-tutorial, January 2020.

[SKP15] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A Unified Embedding for Face Recognition and Clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[SLJ+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SPVZ13] K. Simonyan, OM Parkhi, A. Vedaldi, and A. Zisserman. Fisher Vector Faces in the Wild. In *BMVC*, volume 2, page 4, 2013.

[SSB14] H. Sak, AW. Senior, and F. Beaufays. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling. 2014.

[str20] Streamlink - A CLI utility which pipes video streams from various services into a video player. https://streamlink.github.io/, March 2020.

[SVI+15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. December 2015.

[SVZ14] K. Simonyan, A. Vedaldi, and A. Zisserman. Learning Local Feature Descriptors Using Convex Optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1573–1585, 2014.

[SWF15] S. Sukhbaatar, J. Weston, and R. Fergus. End-To-End Memory Networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[SWT15] Y. Sun, X. Wang, and X. Tang. Deeply Learned Face Representations are Sparse, Selective, and Robust. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2892–2900, 2015.

[SZ14] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TBF+15] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[TBHN16] S. Tripathi, S. Belongie, Y. Hwang, and T. Nguyen. Detecting Temporally Consistent Objects in Videos Through Object Class Label Propagation. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9. IEEE, 2016.

[ten20] Tensorflow Object Detection API. https://github.com/tensorflow/models/blob/master/research/object_detection/README.md, January 2020.

[TLBN16] S. Tripathi, ZC. Lipton, S. Belongie, and T. Nguyen. Context Matters: Refining Object Detection in Video with Recurrent Neural Networks. *arXiv preprint arXiv:1607.04648*, 2016.

[Tor18] J. Torres. *Deep Learning. Introducción Práctica con Keras*. Watch This Space, 2018.

[Tur50] AM. Turing. Computing Machinery and Intelligence. *Mind*, 49:433–460, 1950.

[TYRW14] Y. Taigman, M. Yang, MA. Ranzato, and L. Wolf. Deepface: Closing the Gap to Human-Level Performance in Face Verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[TYRW15] Y. Taigman, M. Yang, MA. Ranzato, and L. Wolf. Web-Scale Training for Face Identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2746–2754, 2015.

[Val20] S. Valdarrama. Confusion Matrix in Object Detection with TensorFlow. Technical Report, GitHub, March 2020.

[VBK15] O. Vinyals, S. Bengio, and M. Kudlur. Order Matters: Sequence to Sequence for Sets. *arXiv preprint arXiv:1511.06391*, 2015.

[WHM11] L. Wolf, T. Hassner, and I. Maoz. Face Recognition in Unconstrained Videos with Matched Background Similarity. In *CVPR 2011*, pages 529–534. IEEE, 2011.

[wid20] Wider Face Dataset to TF Record. https://github.com/iitzco/widerface-to-tfrecord, March 2020.

[WR10] AN. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1910.

[WZLQ16] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A Discriminative Feature Learning Approach for Deep Face Recognition. In *European conference on computer vision*, pages 499–515. Springer, 2016.

[XZLT15] Y. Xiong, K. Zhu, D. Lin, and X. Tang. Recognize Complex Events from Static Images by Fusing Deep Channels. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015.

[YLLT16] S. Yang, P. Luo, C. Change Loy, and X. Tang. WIDER FACE: A Face Detection Benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[YRZ+17] J. Yang, P. Ren, D. Zhang, D. Chen, F. Wen, H. Li, and G. Hua. Neural Aggregation Network for Video Face Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4362–4371, 2017.

[YSMC15] A. Yao, J. Shao, N. Ma, and Y. Chen. Capturing AU-Aware Facial Features and Their Latent Relations for Emotion Recognition in the Wild. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 451–458, 2015.