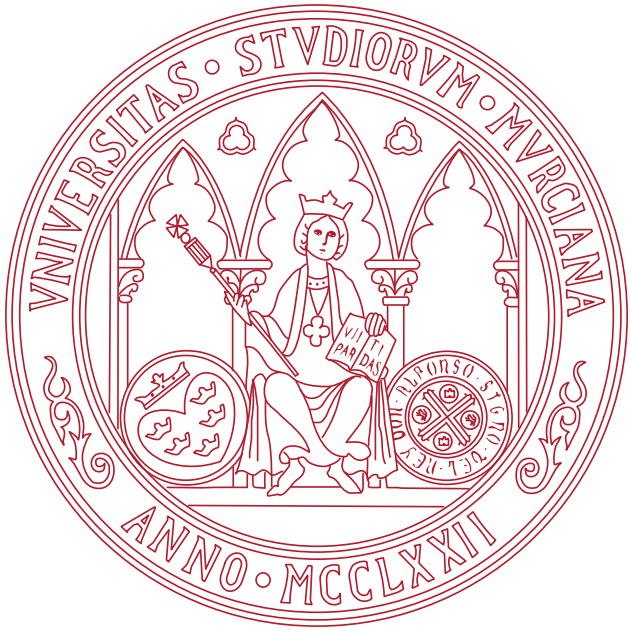

Universidad de Murcia

Trabajo Final de Grado

Supervisión visual de normas COVID

Autor:

Juan José Morell Fernández

juanjose.morellf@um.es

Tutor:

Alberto Ruiz Garcia

a.ruiz@um.es

30 de abril de 2021

Índice general

Índice de figuras

Resumen

En este proyecto se plantea el problema de...

Extended abstract

This project faces the problem of...

CAPÍTULO 1

Introducción

La visión artificial es un ámbito de la informática que surgió hace 60 años, pensado en el estudio del procesamiento digital de las imágenes. En los últimos años ha tomado mucha importancia el reconocimiento facial, algoritmo capaz de identificar rostros humanos dentro de una imagen, gracias a la investigación realizada por *Viola y Jones* en 2001, y la reciente tendencia en *smartphones* con el desbloqueo facial.

1.1. Historia de la Visión Artificial

Uno de los primeros acontecimientos que propició la visión artificial fue la creación del cable Bartlane, capaz de transmitir una imagen a través del mar atlántico en los años 1920, con una duración de cerca a una semana. Pero, dentro del entorno del procesamiento de imágenes digitales, la investigación se centró en recuperar una estructura tridimensional del mundo real a través de una imagen para conseguir un entendimiento total de la escena que plasma la misma. Debido a esto aparecieron varios algoritmos de reconocimiento de lineas, donde uno de ellos fue creado por parte de Huffman en 1971.

Pero el avance de estas investigaciones pronto se ligarían con el del ordenador. Estos, se podrían resumir en varios acontecimientos importantes, tales como: la invención

del transistor por Bell Laboratories en 1948 y de los circuitos integrados en 1958 o la introducción por parte de IBM de los primeros ordenadores personales en 1981. [10]

Uno de los descubrimientos que inicio este movimiento no fue proveniente de la informática, sino de la psicología. Esta sería una de las principales fuentes sobre el entendimiento de como funciona la visión. Un par de psicólogos, David Hubel y Torsten Wiesel, describieron que el comportamiento de las neuronas encargadas de entender el entorno visual siempre empiezan con estructuras simples como vértices. Más tarde esta idea se convertirá en el principio central del *Deep Learning*.

Russell Kirsch, en 1959, es el primero en desarrollar un aparato que traducía las imágenes en datos que las máquinas pudiesen entender. Y, Lawrence Roberts en 1963 publica un estudio sobre como las máquinas perciben objetos sólidos de tres dimensiones, uno de los avances considerados precursores de la visión artificial moderna.

Durante los años 1980s, se desarrolló una red artificial capaz de reconocer patrones, mediante el uso de una red convolucional. Que propició la creación de un modelo llamado LeNet-5, la primera red convolucional moderna. Este modelo se caracteriza por usar la backpropagación. Mientras que en los años 1990s la visión artificial cambia totalmente de rumbo y los investigadores pasaron de intentar reconstruir objetos en 3D a intentar detectar objetos mediante sus características.

A partir del año 2000 se hacen muchos avances importantes y que actualmente son usados en aplicaciones reales. El primer detector facial llegaría en 2001 creado por Paul Viola y Michael Jones. Ambos consiguieron crear el primero que funcionase en tiempo real.

El problema de estos modelos es el uso de información para poder entrenarlos, y para esto se creo un proyecto llamado PASCAL VOC, que creo un dataset estándar para la clasificación de objetos. Posteriormente, apareció en 2010 ImageNet, el cual contiene más de un millón de imágenes para un total de mil objetos. Y junto a este apareció un modelo basado en una red convolucional llamado AlexNet. Desde entonces, el *Deep Learning* se ha convertido en eje central del avance en la visión artificial, conjuntamente con todos los avances matemáticos realizados anteriormente.

1.2. Reconocimiento Facial y COVID-19

En la actualidad, la visión artificial se utiliza en muchos proyectos y esta presente en investigaciones muy importantes para el futuro de la inteligencia artificial. Una de ellas se basa en el reconocimiento facial, y es usado en aplicaciones para reconocer personas, aspectos, contador de personas, etc.

La detección facial se inicia con una imagen arbitraria, con el objetivo de encontrar todas las caras que hay en una imagen, y posteriormente devolver otra con la localización exacta de cada una de las caras. Aunque esta tarea es natural para los humanos, es bastante complicada para los ordenadores. Ya que se encuentran muchos factores que lo dificultan, tales como: la escala, localización, punto de vista, iluminación, lentes, etc. Existen centenares de investigación/proyectos de detección facial, desde uno de los mas influyentes en los años 2000s, como *Viola and Jones face detection*, a proyectos basados en *Deep Learning*, con tecnologías como *Tensorflow* o *YOLO*. [18]

Tras el año 2020 y la aparición del COVID-19, el uso de mascarillas y el cumplimiento de las normas impuestas por la OMS (Organización Mundial de la Salud) están al orden del día. En este trabajo se tendrá como objetivo el estudio de todas estas tecnologías para poder controlar dichas normas, terminando con un prototipo capaz de detectar cuando una personas lleva mascarilla, ya sea al entrar a un comercio, evento u trabajo.

CAPÍTULO 2

Estado del arte

El reconocimiento facial es un ámbito de la visión artificial, que como su nombre indica, se centra en la búsqueda de rostros humanos dentro de imágenes digitales. Durante años se han desarrollado varias tecnologías capaces de realizar dicha acción, de las que se pueden distinguir dos grupos:

- Aplicaciones con sin el uso de *Deep Learning*
- Aplicaciones con uso de *Deep Learning*

Ambos se centran en las características de los rostros humanos para lograr identificarlos. Esto se conoce como *features*, que corresponden con puntos de la cara muy reconocibles, como: el mentón, ojos, cejas, nariz, etc. Esta técnica fue usada por primera vez en 2001, por Paul Viola y Michael Jones, y desde entonces se ha convertido en unas de las técnicas principales en el reconocimiento facial.

Este ejercicio se complica cuando las imágenes presentan errores naturales en su toma (como baja luz, ruido, etc.) o los individuos visten complementos que tapen sus rasgos faciales. Este es el problema que se va a plantear en este trabajo, buscar una posible solución al reconocimiento facial con complementos faciales, en especifico identificar si las personas llevan mascarillas.

2.1. Python y OpenCV

2.2. Aplicaciones sin Deep Learning

HAAR-like Features & ADABOOST

Facial Landmasking

2.3. Aplicaciones con Deep Learning

¿Que es el deep learning?

YOLO

TensorFlow

Keras

MediaPipe

IBM Watson

CAPÍTULO 3

Análisis de objetivos y metodología

El COVID-19 es un problema que ha provocado en la humanidad incontables problemas, y en el mundo de la visión artificial también, por eso me voy a centrar en la creación de un prototipo que sea capaz de revisar el cumplimiento de las normas COVID impuestas en España y en todo el mundo por la OMS. Concretamente, detectar cuando una persona lleva, de manera correcta, una mascarilla al entrar a un comercio, cine, restaurante, etc. Los objetivos que se plantean para llevar esto a cabo son los siguientes:

- Estudio de las tecnologías actuales, para comprobar su comportamiento con uso de mascarilla.

- Creación de un prototipo capaz de reconocer rostros y detectar si se lleva mascarilla.

- Estudiar la capacidad de que el prototipo pueda identificar si se lleva correctamente la mascarilla.

- Poder detectar la mascarilla, independientemente del tipo que se lleve.

3.1. Prototipo

[Explicar el funcionamiento de como se va a llevar a cabo la creación del prototipo]

- Para cada apartado se creará un prototipo especifico para probar dicha tecnología.
- Se realizará un estudio de los resultado para cada uno de ellos.
- En los anexos se mostrará una explicación de como se ha implementado más centrado en la programación.

CAPÍTULO 4

Diseño y resolución

4.1. Paul Viola and Michael Jones

En 2001, el reconocimiento facial tuvo su primera aparición en el campo de la visión artificial como aplicación en tiempo real. Este avance fue de la mano de Paul Viola y Michael Jones. Análogamente, el punto de partida del estudio de este TFG. Durante este apartado, se estudiará el funcionamiento del algoritmo *Viola-Jones face detector*, ideado por estos dos investigadores y se realizará una implementación del mismo mediante *Python* y *OpenCV* para comprobar como se comporta en la situación actual.

Método de estudio

El trabajo de los expertos fue presentado por parte de la Universidad de Cambridge mediante un *paper* (ensayo de la investigación). Y se introduce como:

> "This paper describes a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates"[20]

Para poder lograr esta afirmación se basan en un procedimiento de trabajo en dos

fases: entrenamiento y detección. Igualmente, Paul y Michael dividen el proyecto en tres ideas principales para poder lograr un detector que se pueda ejecutar en tiempo real. Y estas son: la imagen integral, Adaboost (algoritmo de Machine Learning) y un método llamado *attentional cascade structure*.

Con todos estos puntos combinados lograron ingeniar un prototipo capaz de detectar caras humanas con un *frame rate* de 15 fps. Fue diseñado para la detección de caras frontales, haciéndose difícil para posiciones laterales o inclinadas.

Las imágenes que se toman para realizar la detección pasan por una transformación del espacio de color a *grayscale*. Con el objeto de encontrar características en ellas, llamadas *haar-like features*. Nombradas así por su inventor Alfred Haar en el siglo XIX. En este trabajo se hacen uso de tres tipos de haar-like features, que son las siguientes:

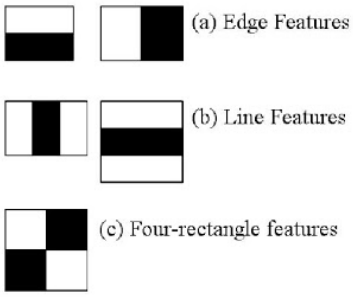

Figura 4.1: Haar-like Features

Las ***Haar-like features***, o también conocidas como *Haar-wavelet* son una secuencia de funciones *rescaled square-shaped*, siendo similares a las funciones de Fourier y con un comportamiento parecido a los *Kernel* usados en las *Redes Convolucionales* (matrices que consiguen extraer ciertas *features* de la imagen de entrada). De manera que, las *Haar Features* serán las características de la detección facial.

En un estudio ideal, los píxeles que forma el *feature* tendrá una división clara entre píxeles de color blanco con los de color negro (Figura 4.1), pero en la realidad eso casi nunca se va a dar.

Más especificamente, las *Haar-like features* están compuestas por valores escalares que representan la media de intensidades entre dos regiones rectangulares de la imagen. Estas capturan la intensidad del gradiente, la frecuencia espacial y las direcciones, me-

diante el cambio del tamaño, posición y forma de las regiones rectangulares basándose en la resolución que se define en el detector. [17]

Estas características van a ayudar al ordenador a entender lo que es la imagen estudiada. Van a ser utilizadas mediante *Machine Learning* para detectar donde hay una cara o no, mediante un recorrido sobre toda la imagen. Esto conlleva una potencia de computación elevada. Para paliar este problema idearon el método de la *Imagen Integral*.

La ***Imagen Integral*** permite calcular sumatorios sobre subregiones de la imagen, de una forma casi instantánea. Además de ser muy útiles para las *HAAR-like features*, también lo son en muchas otras aplicaciones.

Si se supone una imagen con unas dimensiones de $< w, h >$ (ancho y alto, respectivamente), la imagen integral que la representa tendrá unas dimensiones de $< w+1, h+1 >$. La primera fila y columna de esta son ceros, mientras que el resto tendrán el valor de la suma de todos los píxeles que le preceden. [2] Ahora, para calcular la suma de los píxeles en una región especifica de la imagen, se toma la correspondiente en la imagen integral y se suma según la siguiente fórmula (siguiendo la numeración de la Figura 4.2):

$$sum = L4 + L1 - (L2 + L3)$$

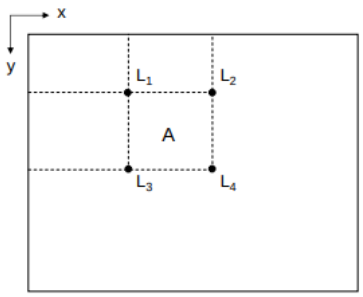

Figura 4.2: Funcionamiento de una *Imagen Integral*

Viola y Jones junta esta propuesta con los filtros *Haar-like features*, y consiguen computar dichas características de manera constante y eficaz. [6]

Una vez estudiada la obtención de características y con un set de entrenamiento, solo queda seleccionar un método de *machine learning* que permita crear una función de clasificación. Concretamente, se plantea el uso de una variante de ***AdaBoost***, que permite seleccionar un pequeño conjunto de características y poder entrenar un clasificador.

Este algoritmo de aprendizaje esta basado en generar una predicción muy buena a partir de la combinación de predicciones peores y más débiles, donde cada uno de estas se corresponde con el *threshold* de una de las características *Haar-like*. La primera vez que aparece este algoritmo, de forma práctica, fue de la mano de *Freund y Schapire* [9]. Sin embargo, el usado por *Viola y Jones* es una modificación de este.

La salida que genera el algoritmo ***AdaBoost*** es un clasificador llamado *Strong Classifier*, como se ha mencionado anteriormente, compuesto por combinaciones lineales de *Weak Classifiers*.

El procedimiento para encontrar *Weak Classifiers* es ejecutar el algoritmo T iteraciones donde T es el número de clasificadores a encontrar. En cada iteración, el algoritmo busca el porcentaje de error entre todas las características y escoge la que menos porcentaje de error presente en dicha iteración. (Como se muestra en la *Figura 4.3*) [15]

Figura 4.3: Construcción del *Strong Classifier*

Con estos clasificadores se procede a la construcción de una estructura en cascada para crear un *Multi-stage Classifier*, que podrá realizar una detección rápida y buena. Por tanto, la estructura de cascada esta compuesta por varios estados de *Strong Classifiers* generados por el algoritmo *AdaBoost*. Donde el trabajo de cada estado será identificar si, dada una región de la imagen, no hay una cara o si hay la posibilidad de que la haya. [9]

Si el resultado de uno de los estados es que no existe una cara en dicha región, esta se descarta directamente. Mientras que, si hay la posibilidad de que exista una, pasa al

siguiente estado de la estructura. De tal forma que, cuantos más estados atraviese una región de la imagen, con más seguridad se podrá afirmar que existe una cara en ella. La estructura completa se refleja en la *Figura 4.4*.

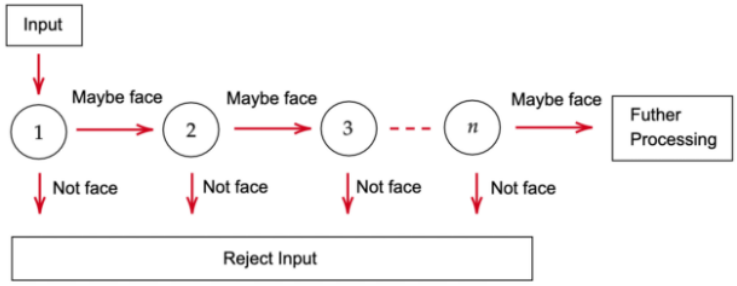

Figura 4.4: Construcción del *Multi-stage Classifier*

Implementación y Experimentación

El prototipo será implementado en *Python*, con el uso de *OpenCV*. Y, el objetivo es construir dos detectores de caras, donde el primero usará un modelo preentrenado de *OpenCV* de caras frontales. Mientras que en el segundo, se intentará modificar el programa, para que mediante el uso de varios modelos preentrenados se pueda detectar una cara con una mascarilla.

La **implementación básica** hace uso de un modelo preentrenado cargado mediante una clase de *OpenCV* llamada *Cascade Classifier*. Esta representa la base de *Machine Learning* explicado en el apartado anterior. Asimismo, *OpenCV* también proporciona una serie de archivos *xml* con diferente modelos preentrenados. En concreto, para este prototipo se hace uso del modelo por defecto, detector de caras frontal, como se muestra en la investigación de *Viola y Jones*.

Finalmente, la detección se realiza, tras realizar una transformación del espacio de color a blanco y negro, mediante la función *detectMultiScale* de la clase, creada anteriormente, *Cascade Classifier*. Concretamente, su funcionalidad será encontrar caras dentro de las imágenes que vaya procesando.

[Explicacion del prototipo custom]

OpenCV y Haar-like features + Machine Learning con PCA y SVM

Se implementa un identificador de caras conjunto a un modelo de *Machine Learning* que identifica cuando una persona lleva o no mascarilla, mediante una toma de muestras anterior. Gracias a los modelos PCA y SVM, se puede crear un modelo para su identificador. Lo malo: Solo funciona con los rostros/rostro que se toma como referencia para construir el modelo, igualmente pasa con el tipo de mascarilla (siendo la quirúrgica la que mejor funciona con este prototipo). Asimismo, su funcionamiento es de manera frontal y cercana, ya que si se coloca la cámara en la borde superior de una puerta o similar, el detector se pierde mucho y crea identificaciones falsa o no llega a reconocer nada.

Este procedimiento se podría llegar a usar con otra implementación, específicamente con HOG.

[Resultados] Bien, es un comienzo. Pero mal para un mercado amplio.

4.2. Facial Landmark

Con el objetivo de ampliar la idea anterior, se plantea el uso de Facial Landmark, una tecnología que nos permite el reconocimiento de puntos de interés en las caras que se han detectado en la imagen. Sus pasos de ejecución son: detectar cara dentro de la imagen (En este caso, se usará *Haar-like features*) y obtener dichos puntos de interés.

La implementación que se va a utilizar es la estudiada por *Kazemi* y *Sullivan* en 2014, con el paper *One Millisecond Face Alignment with an Ensemble of Regression Trees* [13], y usado en el *toolkit Dlib*. Este método se centra en localizar las siguientes zonas faciales: boca, cejas, ojos, nariz y mentón, gracias al uso de un conjunto de árboles de regresión. Estos son entrenados mediante un modelo formado por puntos de interés de un grupo de imágenes, etiquetados a mano y especificadas como coordenadas (x,y).

Dlib será el *toolkit* (*Open Source*) utilizado para la implementación de dicho método. Este contiene algoritmos de *Machine Learning* y herramientas capaces de crear software complejo en *C++* y *Python* para resolver problemas reales. Sobre todo centrado en robótica, dispositivos embebidos, móviles y ordenadores de gran capacidad. [7]

HOG - *Histogram of Oriented Gradients*

El primer paso en esta solución es encontrar una cara dentro de la imagen de entrada, y el encargado será el método HOG. El cual, sigue una idea similar al método de *Haar-like*, ya que se basa en la detección de *features* (características).

La idea teórica tras *HOG* es encontrar la apariencia y forma de un objeto, en este caso caras, mediante la distribución de la intensidad de gradientes locales o direcciones de los bordes. Mientras que la implementación, divide la imagen en pequeñas regiones (llamadas celdas) y se calculo un histograma de gradientes de 1 dimensión para cada uno de los píxeles de cada celda. Para un mejor estudio se normaliza el contraste. [4]

Con este procedimiento se obtiene un *feature vector* a partir de la imagen de entrada.

In the HOG feature descriptor, the distribution (histograms) of directions of gra-

dients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions.

La implementacion de HOG se podria dividir en 5 pasos generales:

1. Preprocesado.

   HOG feature descriptor used for pedestrian detection is calculated on a 64x128 patch of an image. Of course, an image may be of any size. Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches being analyzed have a fixed aspect ratio.

2. Calculo de las imagenes de gradiente.

   To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels. Next, we can find the magnitude and direction of gradient using the following formula.If you are using OpenCV, the calculation can be done using the function *cartToPolar*. Notice, the x-gradient fires on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires where ever there is a sharp change in intensity. None of them fire when the region is smooth. I have deliberately left out the image showing the direction of gradient because direction shown as an image does not convey much. The gradient image removed a lot of non-essential information (e.g. constant colored background), but highlighted outlines. In other words, you can look at the gradient image and still easily say there is a person in the picture. At every pixel, the gradient has a magnitude and a direction. For color images, the gradients of the three channels are evaluated (as shown in the figure above). The magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.

3. Calculo de los histogramas de gradientes en celdas (8x8).

In this step, the image is divided into 8x8 cells and a histogram of gradients is calculated for each 8x8 cells. We will learn about the histograms in a moment, but before we go there let us first understand why we have divided the image into 8x8 cells. One of the important reasons to use a feature descriptor to describe a patch of an image is that it provides a compact representation. An 8x8 image patch contains 8x8x3 = 192 pixel values. The gradient of this patch contains 2 values (magnitude and direction) per pixel which adds up to 8x8x2 = 128 numbers. By the end of this section we will see how these 128 numbers are represented using a 9-bin histogram which can be stored as an array of 9 numbers. Not only is the representation more compact, calculating a histogram over a patch makes this represenation more robust to noise. Individual graidents may have noise, but a histogram over 8x8 patch makes the representation much less sensitive to noise. But why 8x8 patch ? Why not 32x32 ? It is a design choice informed by the scale of features we are looking for. The contributions of all the pixels in the 8x8 cells are added up to create the 9-bin histogram.

4. Normalización de bloques (16x16).

   In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half. Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations. A 16x16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3x1 vector is normalized. The window is then moved by 8 pixels (see animation) and a normalized 36x1 vector is calculated over this window and the process is repeated.

5. Calculo del vector de *HOG*: To calculate the final feature vector for the entire image patch, the 36x1 vectors are concatenated into one giant vector.

Sullivan Paper

El *paper* de Kazemi y Sullivan presenta la implementación usada en el *toolkit Dlib* del algoritmo que estima de forma precisa y eficiente de los puntos de interes faciales. Para ello se hará uso de una *cascada de regresores*. [14]

Prototipo

Próximos pasos

Con el objetivo de mejorar este prototipo se incorpora el *Deep Learning*. En las últimas décadas, se ha convertido en uno de los métodos mas usados para la creación de aplicaciones de visión artificial. [18]. Esta basado en las estructuras neuronales CNN (*Convolutional Neural Network*), principales responsables de que un ordenador pueda procesar de forma sencilla una imagen. *CNN* es una combinación entre capas neuronales y convolcionales, siendo estas últimas filtros con los que se obtendrán características de la imagen de entrada, los conocidos *features*. La principal función del Deep Learning es clasificar, ya sea una imagen, un objeto o incluso varios objetos a la vez. Esto es gracias al uso de modelos, entrenados con miles de imágenes, que consiguen clasificar imágenes en varias categorías, por ejemplo *MobileNet*. [8]

En los siguientes apartados se probará implementar prototipos con el uso de Deep Learning con el objetivo de mejorar los resultado obtenidos hasta ahora. Se va a tratar con la API de Google llamada *MediaPipe* y la tecnologia de *Transfer Learning* conjuntamente con *TensorFlow*, para poder reentrenar modelos para nuestro objetivo, detectar mascarillas.

4.3. Mediapipe

Mediapipe es una API *open-source* creada por *Google*, que ofrece servicios de *Machine Learning* para vídeos y fuentes multimedia. Entre ellas, se encuentra un servicio llamado *Face Mesh* que ofrece una solución que estima 468 puntos de interés de un rostro, que conforman una malla 3D en tiempo real. Este usa aceleración GPU conjuntamente con un modelo y el uso de una *pipeline*.

La *pipeline* que se utiliza en esta API consiste en dos modelos de *Deep Learning* que trabajan al mismo tiempo. Su funcionalidad es realizar una detección a partir de una imagen de los puntos de interés sobre una cara y construir un modelo *face landmark* 3D que aproxima la superficie de esta mediante regresión sobre dichos puntos. Esta tarea es facilitada si la cara, donde se tienen que detectar los puntos de interés, se encuentra recortada, haciendo así que el modelo se centre solamente en buscar los puntos, aumentando la precisión de la predicción. Asimismo, los recortes de las caras se puede generar a partir de las predicciones anteriores realizadas por el mismo modelo, y solamente es llamada la predicción nuevamente cuando no se consigue detectar la presencia de la cara.[11]

Todo esto es implementado gracias al framework *MediaPipe*, con la herramienta *MediaPipe graph*. Arquitectura caracterizada por estar formada por componentes llamados *Calculator*, nodos del grafo que tras la entrada de cero o más inputs generan cero o mas salidas. Todos estos nodos estan conectados mediante datos en forma de *Streams*, donde cada uno representa un conjunto de datos-tiempo en *Packets*. Por tanto, los *calculators* y *steams* definen el flujo de datos del *Graph*. [16]

El *pipeline* puede ser definido mediante la adición/modificación de *calculators* dentro del *graph*. Especificamente, el pipeline que se utiliza en esta solución (*FaceMesh*) esta formada por un *graph* compuesto por un *subgraph* de *face landmark* (proveniente del modulo, ya implementado, de face landmark de Mediapipe), donde a su vez usa otro subgraph proveniente de face detection module para la detección de caras, y un *face renderer subgraph* para mostrar el resultado. [11] En concreto el *graph* que se usa en esta implementación es el mostrado en la Figura 4.5

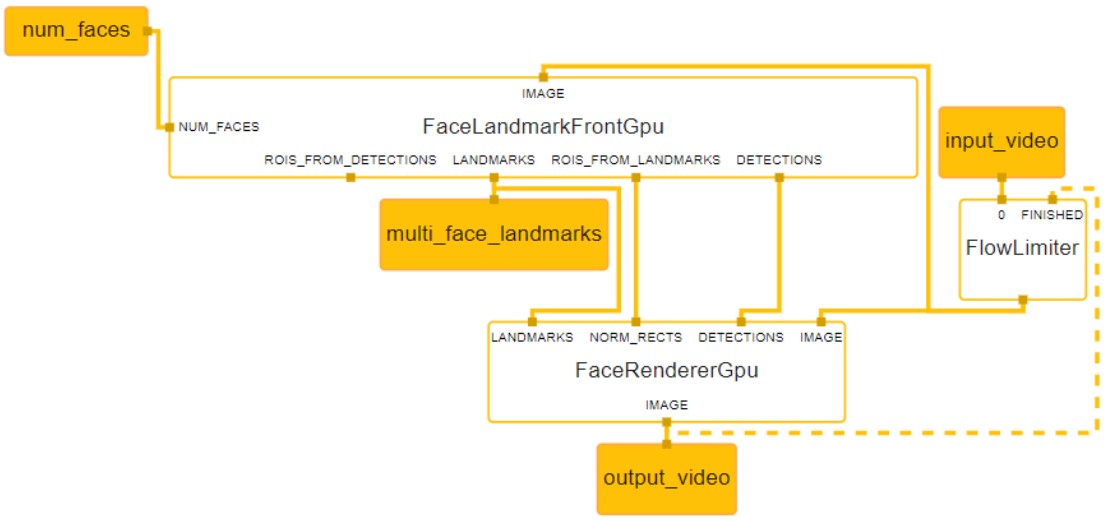

Figura 4.5: MediaPipe Graph utilizado en FaceMesh

Por tanto, el procesamiento de una imagen en este modelo sigue dos pasos. El primero (1) toma la imagen de entrada, capturada por la cámara, es procesada por un detector de caras *lightweigth*, llamado *BlazeFace*, y produce unos rectángulos que definen el perímetro donde se encuentra la cara, conjuntamente con un par de puntos de interés superficiales (ojos, boca y nariz). Estos puntos se utilizarán para alinear la cara para el siguiente paso. Y el segundo (2), mediante el rectángulo obtenido en el paso anterior, se recorta la cara de la imagen inicial y es reescalado para utilizarse como entrada de la red neuronal que realiza la prediccion de la malla. (El tamaño de reescalado será entre 256x256 en un modelo completo, hasta 128x128 en el modelo más pequeño). Tras la predicción, se obtiene como salida un vector de coordenadas *landmark 3D*, que serán mapeadas y dibujadas en la imagen original. [12]

Las coordenadas que se obtienen como salida estan compuestas por unas coordenadas x e y provenientes de localizaciones del plano 2D propio de la imagen. Mientras que, la coordenada z es interpretada como una profundidad relativa a un centro de masa que compone la malla de la cara.

Se utilizan dos modelos para el funcionamiento de *FaceMesh*. El primero de ellos dedicado a la detección facial, llamado BlazeFace (mencionado anteriormente). Este es un modelo *lightweight* creado para GPU móviles, llegando a una velocidad de procesamiento de entre 200 a 1000 fps en dispositivos móviles punteros. Es inspirado en los

modelos *MobileNet*, tanto la primera version como la segunda, provenientes del *framework* SSD (*Single Shot Multibox Detector*). Este modelo produce una salida compuesta por un rectángulo perimetral y 6 puntos de interés faciales. [3]

El segundo modelo, *Face Landmark Model*, es generado mediante *Transfer Learning* buscado una serie de objetivos: crear coordenadas 3D (mencionadas anteriormente) y conseguir mostrarlas en la imagen de salida de forma correcta. [11] El *Transfer Learning* es una técnica de *Machine Learning* donde se puede hacer uso de un modelo preentrenado para personalizarlo y usarlo en una tarea determinada. Conviene destacar que un modelo entrenado es una red almacenada, entrenada previamente con un conjunto de datos con el objetivo de realizar una tarea de clasificación de imágenes a gran escala. [19]

FaceMesh propone una implementación mediante *TensorFlow Lite* que dispone de dos formatos: CPU y GPU, que presentan un rendimiento en dispositivos moviles (*Pixel3*, *Pixel2* y *iPhoneX*) muy fluido, impresionado sobre todo su funcionamiento sobre CPU. (Figura 4.6) [1]

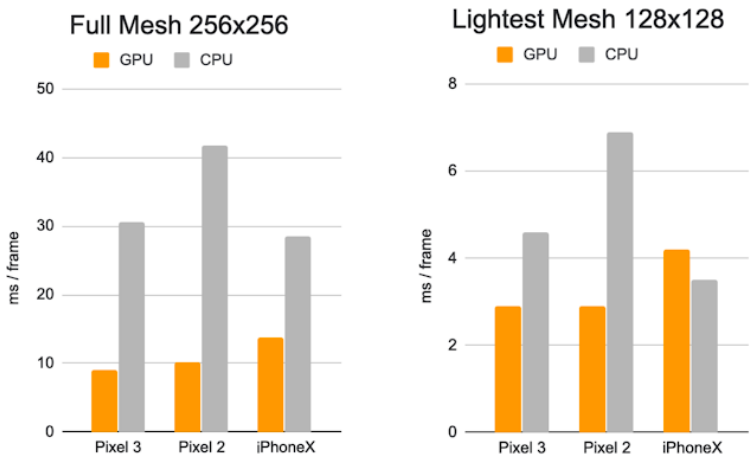

Figura 4.6: Rendimiento de FaceMesh sobre dispositivos móviles

Prototipo

Uso de *FaceMesh* del framework *MediaPipe* y un detector basado en *Haar-like features*, como el mencionado en el *paper* de Viola & Jones (4.1).

4.4. Tensorflow

Posible añadido de AlexNet si es útil.

- Plantear idea
- Procedimiento
- Diferentes tipos de modelos
- Mostrar funcionamiento y aplicación al objetivo

4.5. Comparación

CAPÍTULO 5

Conclusiones y vías futuras

Finalizamos el trabajo estableciendo las conclusiones y vías futuras...

Bibliografía

[1] Artsiom Ablavatski and Google AI Ivan Grishchenko, Research Engineers. Real-time ar self-expression with machine learning. 03 2019. Acceso: 25-04-2021.

[2] AIShak. Integral images in opencv. https://aishack.in/tutorials/integral-images-opencv/, Jun 2010. Acceso: 07-04-2021.

[3] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus. 07 2019.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005. doi:10.1109/CVPR.2005.177.

[5] Rostyslav Demush. A brief history of computer vision (and convolutional neural networks). 02 2019. URL: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3.

[6] Konstantinos Derpanis. Integral image-based representations. 1, 01 2007.

[7] Dlib. Dlib library, 2021. Acceso: 25-04-2021. URL: http://dlib.net.

[8] Govinda Dumane. Introduction to convolutional neural network (cnn) using tensorflow. 2020. Acceso: 28-04-2021. URL: https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83.

[9] Robert E. Schapire. Explaining adaboost. 2020. Acceso: 11-04-2021. URL: https://www.math.arizona.edu/~hzhang/math574m/.

[10] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Pearson, 2018.

[11] Google. Mediapipe face mesh. 2020. Acceso: 21-04-2021. URL: https://google.github.io/mediapipe/solutions/face_mesh.

[12] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus. 07 2019.

[13] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. 06 2014. doi:10.13140/2.1.1212.2243.

[14] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014. doi:10.1109/CVPR.2014.241.

[15] Socret Lee. Understanding face detection with the viola-jones object detection framework. 2020. Acceso: 11-04-2021. URL: https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc

[16] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for building perception pipelines. 06 2019.

[17] Takeshi Mita, Toshimitsu Kaneko, and Osamu Hori. Joint haar-like features for face detection. *IEEE Int Conf Comp Vis*, 2:1619 – 1626 Vol. 2, 11 2005. doi:10.1109/ICCV.2005.129.

[18] R. Szeliski. *COMPUTER VISION: Algorithms and aplications.* SPRINGER NATURE, 2021.

[19] TensorFlow. Transferir el aprendizaje y la puesta a punto, 2021. Acceso: 25-04-2021. URL: https://www.tensorflow.org/tutorials/images/transfer_learning.

[20] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *IEEE Conf Comput Vis Pattern Recognit*, 1:I–511, 02 2001. doi:10.1109/CVPR.2001.990517.