UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"DETERMINACIÓN DE DISTANCIA DE SEGURIDAD EN AGLOMERACIONES DE PERSONAS"

AUTOR: JEFERSON AGUSTIN CAIZAPASTO GUACHALA

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL AUTOR			
CÉDULA DE IDENTIDAD	1724845829		
APELLIDOS Y NOMBRES	CAIZAPASTO GUACHALA JEFERSON AGUSTIN		
DIRECCIÓN	Ecuador, Pichincha, Pedro Moncayo, Tabacundo		
EMAIL	jacizapastog@utn.edu.ec - jefricaizapasto@gmail.com		
TELÉFONO FIJO	0996031585	TELÉFONO MÓVIL	0996031585
DATOS DE LA OBRA			
TÍTULO	“DETERMINACIÓN DE DISTANCIA DE SEGURIDAD EN AGLOMERACIONES DE PERSONAS ”		
AUTOR	JEFERSON AGUSTIN CAIZAPASTO GUACHALA		
FECHA	16 SEPTIEMBRE 2021		
PROGRAMA	PREGRADO		
TÍTULO POR EL QUE OPTA	INGENIERO EN MECATRÓNICA		
DIRECTOR	CARLOS XAVIER ROSERO CHANDI		

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, 16 de septiembre de 2021

Caizapasto Guachala Jeferson Agustin
C.I.: 1724845829

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado “DETERMINACIÓN DE DISTANCIA DE SEGURIDAD EN AGLOMERACIONES DE PERSONAS”, presentado por el egresado JEFERSON AGUSTIN CAIZAPASTO GUACHALA, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 16 septiembre de 2021

Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

Carlos Xavier Rosero
DIRECTOR DE TESIS

Agradecimiento

Agradezco Dios, por la vida y la salud; por brindarme fortaleza, alegría, por guiar cada decisión y paso que doy. A la Universidad Técnica del Norte, por abrirme las puertas hacia el conocimiento; por contar con docentes que, con su pasión, entrega, sabiduría y conocimientos, nos formaron como profesionales éticos a lo largo de la vida universitaria. Quiénes pese a la situación de la pandemia y modalidad de estudio diferente a la convencional, supieron comprendernos, enseñarnos y solventar oportunamente cada inquietud en el desarrollo del presente trabajo. A mis Padre no solo por darme la vida, sino por el constante esfuerzo, sacrificio y confianza entregada a lo largo de mi vida. Por sus sabios consejos que han permitido alcanzar mis logros, aprender de mis errores y tomar buenas decisiones. A mis compañeros y amigos que con su alegría, amistad y paciencia hicieron que los días de estudio sean llevaderos y dejen gratos recuerdos. Finalmente, a quienes han aportado con un granito de arena en lo largo de mi vida, y han influido positiva mente en mi formación personal y académica. Muchas gracias a todos.

Jeferson Agustin Caizapasto Guachala

Dedicatoria

El presente trabajo de grado dedico a mi padre José Agustin Caizapasto Tituaña que, con su amor incondicional, sus palabras de aliento, consejos y grata compañía ha estado a mi lado en cada paso que doy, en cada noche en vela, por guiarme por el buen camino e inculcarme el deseo de superarme y salir a delante, siendo una buena persona. A mi hermana Geovanna, que con su cariño y ocurrencias me ha sacado una sonrisa incluso en los días malos y ser mi motor para salir adelante con su apoyo, compañía, y guía. A mis hermanos por darme ánimos y sabios consejos. A todas las personas que estuvieron junto a mí, en mis triunfos y ayudarme a sobresalir de los errores y fracasos.

Jeferson Agustin Caizapasto Guachala

Resumen

El principal objetivo de este proyecto es desarrollar un software basado en visión artificial para identificar el número de peatones y distinguir la distancia de seguridad en veredas, parques u salones. De la imagen se extraerá algunas características, tales como el número de personas, la posición relativa para posteriormente calcular la distancia de las personas entre sí.

La información del entorno se puede conseguir gracias a la ayuda de una cámara. Para evaluar el correcto funcionamiento del algoritmo se utilizo una base de datos con imágenes diferente al que se utilizó para entrenar.

Para comenzar el proyecto se estudió las diferentes técnicas existentes de visión artificial sobre la identificación de multitudes de peatones, para estar informado de los nuevos y últimos avances que existe.

Posteriormente se eligió qué camino sería el más adecuado para culminar con éxito el desarrollo del proyecto. Luego de haber elegido el método a seguir, se realizó un filtrado y acondicionamiento de la imagen para eliminar el ruido y otros factores que provoca falsos positivos en la imagen.

Finalmente, al obtener el reconocimiento correcto de los peatones, se calcula los centros de masa de los mismos para conocer la posición relativa. Y calcular las distancias de seguridad correcta entre sí. Además, Con los datos obtenidos de este algoritmo se podrá utilizar para implementar en otros proyectos que necesiten identificar o detectar personas.

Abstract

The main objective of this project is to develop a software based on artificial vision to identify the number of pedestrians and distinguish the safety distance in sidewalks, parks or classrooms. From the image some characteristics will be extracted, such as the number of people, the relative position to later calculate the distance of the people from each other.

The information of the environment can be obtained thanks to the help of a camera. To evaluate the correct functioning of the algorithm, a database with images different from the one used for training was used.

To start the project, the different existing techniques of artificial vision on the identification of crowds of pedestrians were studied, to be informed of the new and latest advances that exist.

Subsequently, it was chosen which path would be the most appropriate to successfully complete the development of the project. After having chosen the method to follow, a filtering and conditioning of the image was carried out to eliminate noise and other factors that cause false positives in the image.

Finally, when obtaining the correct recognition of the pedestrians, the centers of mass of the same are calculated to know the relative position. And calculate the correct safety distances from each other. In addition, with the data obtained from this algorithm, it can be used to implement in other projects that need to identify or detect.

Índice general

Índice de figuras

Índice de cuadros

Lista de Programas

Introducción

Problema

Las autoridades del Ecuador en virtud a la pandemia denominada Covid-19[1, 2] han dispuesto a la ciudadanía un plan de confinamiento y bioseguridad como el distanciamiento físico el fin de evitar aglomeraciones y la fácil propagación de dicha enfermedad[3].

El confinamiento impuesto en el Ecuador ha llevado a que el índice de desempleo [4] suba y con esto muchos hogares se han visto inmersos en la pobreza con dificultades de abastecerse de alimento dando respuesta a ignorar las restricciones de distanciamiento acarreando un posible rebrote de la enfermedad.

Uno de los métodos más efectivos para evitar la propagación del covid-19 es detectar la distancia de seguridad en escenarios donde puedan existir aglomeraciones, siempre y cuando además de incluir la restricción de actividades sociales como viajes, cancelación de eventos a gran escala como conciertos, conferencias, y clases universitarias [5].

Los métodos tradicionales de conteo manual resulta ser el más impreciso debido a que, tras breve tiempo, el cansancio y el hastío hacen que la persona encargada se distraiga por lo que su fiabilidad puede rondar el 50 por ciento además no son factibles para poblaciones grandes porque son ineficientes y difíciles de verificar, mientras que los programas de visión por computadora son capaces de contar un número muy elevado de personas por unidad de tiempo, de manera automática proporcionando una precisión del 97 por ciento [6].

Además, con los avances tecnológicos de la visión por computadora se puede utilizar para llevar a cabo análisis de vídeos. Esto es especialmente útil a la hora de aplicarlo en estrategias de seguridad y control, para monitorear todo tipo de entornos llegando a tener una gran importancia en el ámbito de la bioseguridad [7].

Por lo cual, se pretende desarrollar un sistema que permita determinar la distanciar y el número de personas presentes en situaciones de aglomeraciones, aplicando algoritmos de visión por computador como el reconocimiento de formas y la estimación de distancias. La rapidez con la que se obtienen los resultados utilizando la visión por computadora hace de este sistema sea

potencialmente muy útil y eficiente.

Debido a la nueva revolución tecnológica que combina técnicas de producción y operaciones con la visión por computadora ayuda obtener, procesar y analizar imágenes. Permitiendo automatizar una amplia gama de tareas como por ejemplo en campo de la biología, medicina, seguridad, industria, robótica, geología, meteorología, etc.[8]. Esto hace que esta tecnología llegue a quedarse por un largo tiempo.

Objetivos

Objetivo General

Desarrollar un sistema de conteo de personas en multitudes utilizando un método de aprendizaje por computadora, bajo una plataforma de software libre.

Objetivos Específicos

- Determinar las técnicas de detección y seguimiento de aglomeraciones a través de la revisión del estado del arte para escoger el algoritmo de conteo óptimo.
- Desarrollar el sistema de conteo de peatones en imágenes de multitudes usando software libre para monitorear aglomeraciones.
- Realizar una prueba piloto sobre un ambiente con variables controladas para validad el correcto funcionamiento del algoritmo.

Justificación

El presente trabajo tiene gran importancia para monitorear lugares que presentan aglomeraciones de peatones, debido a la realidad después de la pandemia del COVID-19, es necesario determinar la distancia del tránsito peatonal en un determinado lugar para evitar el contagio[9].

Actualmente existe un gran interés en la tecnología de visión por computadora, debido a los avances considerables en la detección y el seguimiento de personas que es importante para muchas aplicaciones como la vigilancia visual, salas inteligentes, conteo de personas y controlar el tráfico peatonal. Estas son tareas repetitivas de inspección realizadas por operadores que ahora son automatizadas para reducir el tiempo[10].

El trabajo cubre la necesidad de un sistema de conteo de aglomeraciones de peatones basado en visión artificial para estimar el tamaño de las multitudes no homogéneas, es decir que viajan en diferentes direcciones. Ayudando a prevenir más contagio, muertes y un colapso sanitario

por COVID-19. Este trabajo se va a desarrollar por medio de segmentación de movimiento en texturas y regresión de procesos gaussianos en software numérico y software matemáticos.

Además, este trabajo pretende contribuir a la academia información sustentable sobre visión artificial en el tema de aglomeraciones, para posteriormente desarrollar nuevos métodos para estimar el número de personas en multitudes y presentar protocolos de bioseguridad para prevenir un rebrote del COVID-19.

Alcance

El software que se va a desarrollar comprende un conjunto de procesos destinados a realizar el análisis de multitudes para determinar la distancia de seguridad haciendo una mezcla de texturas dinámicas para segmentar las multitudes, extraer características y realizar proceso gaussiano (GP) para hacer retroceder los vectores de características al número de personas por segmento que se mueven en diferentes direcciones además se desarrollará bajo la plataforma de software libre.

Capítulo 1

Revisión Literaria

1.1. Visión por Computadora

La visión por computadora dentro de la comunidad científica tiene la función de reconocer y localizar objetos en diferentes ambientes mediante el procesamiento de imágenes, para enseñar a los ordenadores a realizar tareas asociadas con la inteligencia humana, entre las cuales está la capacidad de resolver problemas, analizar información visual o comprender lenguajes[11]; el procesamiento de imágenes es el área que se encuentra más ligada a la visión por computadora, ya que se utiliza para mejorar la calidad de imágenes y posteriormente se utilizar para es extraer características de una imagen para su descripción e interpretación por la computadora por ejemplo determinar la localización, distancia y tipo de objetos en la imagen.

Dentro de la comunidad científica en los años cincuenta, existió un interrogante de la posibilidad que la visión artificial funciona de manera semejante con la visión humana [12]; ya que las computadoras y la visión humana trabajan de forma similar. El recurso básico para la visión artificial es una imagen obtenida mediante un dispositivo (cámara). Una imagen multiespectral f es una función vectorial con componentes (f1; f2; :::; fn), donde cada una representa la intensidad de la imagen a diferentes longitudes de onda. correspondientes al valor de una función bidimensional que son coordenadas espaciales y el valor de representa el brillo de la imagen. En el caso de las imágenes a blanco y negro e imágenes a color, corresponde a la combinación de tres matrices en el modelo de color RGB[13].

1.1.1. Etapas de la Visión por Computadora

Las etapas de la visión artificial son técnicas para el procesamiento de imágenes o reconocimiento de formas donde se puede incluyen técnicas de modelado geométrico y procesos de conocimiento. Aunque sea diferente un proyecto de otro y tenga otras características, al momento de utilizar la visión por computadora hay una serie de etapas que son muy comunes como:

1.1.1.1. Adquisición

Etapa donde se obtiene las imágenes mediante técnicas fotográficas. Ademas es la etapa más importante para el desarrollo del proceso de la visión por computadora, ya que si se obtiene una buena adquisición de imágenes como muestra la Figura 1.1 . Posteriormente ayudara en las siguientes fases.

Figura 1.1: Imagen original

1.1.1.2. Pre-procesamiento

Etapa donde se pretende mejorar la calidad de la imagen y ayudar a buscar las características especificas en ella. Aquí se utiliza técnicas de eliminación de ruido, realce de contraste, mejoramiento de intensidad, extracción de borde, etc. Como se puede observar en la Figura 1.2 .

Figura 1.2: Imagen procesada de un objeto

1.1.1.3. Segmentación

Aquí se divide la imagen para realizar la extracción de características que tiene mas significado en la imagen. Unas características serán homogéneas como pueden ser en el color, la textura y la intensidad; como ejemplo en la Figura 1.3. La segmentación de lechugas en el suelo.

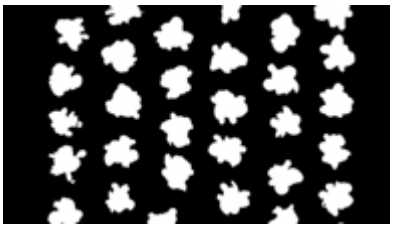

Figura 1.3: Segmentación de lechugas en el suelo

Además, la segmentación de una imagen es el proceso de asignación de una etiqueta para cada píxel, para que los pixeles que comparten la misma etiqueta y que además tengan características visuales similares, para dar lugar a distintas particiones. El objetivo de la segmentación es cambiar la representación de la imagen en otra más significativa y fácil de analizar. Se utiliza tanto para localizar objetos como para encontrar los límites de estos dentro de la imagen. También puede ser extendida como una secuencia de imágenes, teniendo en cuenta sus características visuales, el comportamiento de los pixeles a lo largo del tiempo[16].

Estos modelos de procesos visuales dinámicos se pueden utilizar para la partición del dominio espaciotemporal de una secuencia de vídeo. Este problema no solo es útil para la obtención de vídeo basado en contenido, sino también para otras tareas como navegación robótica, vigilancia por vídeo, restauración, edición y compresión.

La segmentación de un vídeo consiste en particionar el dominio de una imagen, en dos o más regiones disjuntas, donde las regiones no varían en el tiempo. Se espera que los elementos presentes en una región estén relacionados entre sí, es decir, que tengan un comportamiento similar. O sea, cada región debe representar un fenómeno diferente, ya sea, por su dinámica o apariencia. El método de segmentación de imágenes agrupa los pixeles de acuerdo con su intensidad. En este caso, en vez de utilizar la intensidad del píxel, se utiliza la distancia entre el modelo del píxel y un modelo de referencia. En realidad, una vez calculadas las distancias se puede utilizar cualquier técnica de segmentación de imágenes.

- Distancia entre texturas dinámicas
  Para calcular la distancia entre dos texturas dinámicas se utilizan únicamente los parámetros [A;C], ya que se considera que dos procesos con diferente ruido son equivalentes. Como función de distancia se podría utilizar simplemente la norma Frobenius, pero los resultados que se obtienen tomando esta medida son bastante pobres. Debido a esto, se utilizan los ángulos principales que se forman entre los subespacios de los modelos de las texturas dinámicas[17].

- Segmentación basada en regiones
  El procedimiento para llevar a cabo la segmentación de un video que presente, al menos, dos fenómenos relevantes. El método en cuestión primero genera una imagen a partir del video, que luego es segmentada utilizando una técnica de segmentación para imágenes.

Esta imagen codifica en cada uno de sus pixeles la apariencia y dinámica de cada posición del video a lo largo del tiempo, y es por esto por lo que el resultado de la segmentación sobre esta imagen resuelve la segmentación sobre el video.

1.1.1.4. Representación

En este proceso se parametriza las divisiones obtenidas en la etapa anterior.

1.1.1.5. Descripción

Aquí se extraer el objeto de estudio o características que lo diferencian. Para ello se extrae las características invariantes o independientes, como pueden ser patrones de texturas, perímetro del contorno,etc.

La selección de características es el paso fundamental para la aplicación de cualquier técnica de Machine Learning debido que se va utilizando para el entrenamiento y posteriormente en aplicaciones de sistemas inteligentes. Para el contexto de este trabajo, la detección de peatones en multitudes. La etapa de extracción de características sirve para transformar señales cerebrales originales en una representación que facilita la clasificación.

En otras palabras, el objetivo de la extracción de características es eliminar el ruido y otra información innecesaria de las señales de entrada, al mismo tiempo que retener información que es importante para discriminar diferentes clases de señales[18].

Para la selección efectiva de características, existen dos elementos importantes: una función de evaluación que puede ser tipo *filtro* o *wrapper* y un procedimiento iterativo de búsqueda completa o heurística

- Las funciones de costo tipo filtro son aquellas que no consideran el algoritmo de clasificación como generador de la medida que determina la selección de las características para el proceso de reconocimiento de patrones, es decir, filtran características irrelevantes antes que ocurra la etapa de clasificación. A menudo, las funciones de costo tipo filtro se basan en métodos estadísticos que emplean técnicas de análisis univariado o multivariado. Siguen un criterio alternativo, de forma que no hay lazo de realimentación entre uno y otro bloque. Los métodos no guiados tratan de maximizar algún coste que esté relacionado con la capacidad de discriminación entre las clases; de esta forma, se persigue que el clasificador resuelva un problema más sencillo.

- Por otra parte, las funciones tipo wrapper emplean como función de evaluación el propio algoritmo de clasificación para eliminar las características menos influyentes. Los resultados de clasificación son realimentados al módulo de extracción o selección de características, de forma que el criterio que persigue el módulo de reducción de la dimensión

es minimizar la probabilidad de error.

- En cuanto al procedimiento de búsqueda heurística se basa en reglas empíricas que están orientadas a reducir la complejidad computacional, evitando disminuir el rendimiento del sistema. Los métodos heurísticos de búsqueda requieren de una condición de parada para prevenir que la búsqueda de subconjuntos de características se vuelva exhaustiva.

Las características que pueden ser extraídas son:

- Las características globales se obtienen teniendo en cuenta todos los pixeles de una imagen como es el método de extracción de características. Aunque los algoritmos obtenidos con este método, como los propuestos en Wu & Zhang[19], tienen un rendimiento bastante bueno, pero existe un problema que hay que tomar encuneta, y es que las características globales son sensibles a las variaciones en la imagen, como la orientación, los cambios en la iluminación o la oclusión parcial.

- Las características locales, por el contrario, se ven menos afectados a los cambios de iluminación, orientación entre otros. Además, la información geométrica y las restricciones en la configuración de las diferentes características locales pueden ser utilizadas de forma tanto implícita como explicita, es decir, no solo importa su valor, si no, también su posición en la imagen.

1.1.1.6. Reconocimiento

Luego de obtener las características de los descriptores de la etapa anterior, se clasifica los objetos de la imagen. Aquí se puede utilizar son algoritmos genéticos, redes neuronales y métodos estadísticos.

1.1.1.7. Interpretación

En esta etapa final es lograr interpretar el significado a los objetos reconocidos [24]

1.1.2. Aplicaciones de la Visión por Computadora

En la actualidad la visión artificial se usa bastante en diferentes aplicaciones, ya que tiene como finalidad la extracción de información del mundo físico como colores, formas, brillo, etc. a partir de imágenes, utilizando para ello un ordenador[14].
Las aplicaciones que se puede realizar con la visión por computadora son:

- Robótica móvil y vehículos autónomos. En esta área se utiliza cámaras y sensores para localizar obstáculos, idéntica objetos y personas, encontrar el camino, etc.

- Manufactura. Se utiliza para la localización e identificación de piezas, para control de calidad, entre otras tareas.
- Interpretación de imágenes aéreas y de satélite. Aquí se usa para idénticar diferentes tipos de cultivos, también ayuda en la predicción del clima, etc.
- Análisis e interpretación de imágenes médicas. Se aplica para ayudar en la interpretación de diferentes clases de imágenes médicas como rayos-X, tomografía, ultrasonido, resonancia magnética y endoscopia.
- Interpretación de escritura, dibujos, planos. Se utilizan para el reconocimiento de textos, como reconocimiento de caracteres, interpretación automática de dibujos y mapas.
- Análisis de imágenes microscópicas. Se utilizan para ayudar a interpretar imágenes microscópicas en química, física y bióloga.
- Análisis de imágenes para astronomía. Se usa para procesar imágenes obtenidas por telescopios, ayudando a la localización e identificación de objetos en el espacio.

1.2. Procesamiento de la Imagen en Tiempo Real

El objetivo de utilizar técnicas de mejoramiento de imagen es para poder procesar de forma más adecuada la imagen. Debido que depende del problema a resolver, se puede emplear una u otra técnica.

Al trabajar con sistema que son desarrollados en tiempo real se tiene que responder a impulsos que pueden surgir dentro de un plazo especificado y finito. Además, el funcionamiento correcto del sistema depende del tiempo que se demora en obtener estos resultados y no solo en los cálculos. Ya que un sistema a tiempo real tiene que ser más que ser rápido, ser predecible [28].

1.2.1. Imagen en Movimiento

En la actualidad con la ayuda de la tecnología de captura las imagen de cámaras de vídeo o cámaras web es posible realizar procesos de visión artificial. La tecnología de captación de la imagen inicia en el siglo XIX, con la invención de la fotografía.

Posteriormente se comenzó a trabajar en la cronofotografía y otros precursores del cine, que acabaron incentivando las tecnologías de captación de la imagen en movimiento. Ya que una serie de fotografías disparadas a gran frecuencia provocan la ilusión del movimiento [24].

En la primera década del siglo XXI, fue el auge del uso dela fotografía y el vídeo digitales, lo que produjo grabaciones en alta resolución a un relativo bajo coste y con un elevado número

de imágenes por segundo.

1.2.2. Imagen Digital

Una imagen digital es la agrupación de pixeles, cada uno con diferente valor de intensidad o brillo asociado. Esta imagen se representa mediante una matriz bidimensional, de forma que cada elemento de la matriz se corresponde con cada pixel en la imagen Figura 1.4[23].

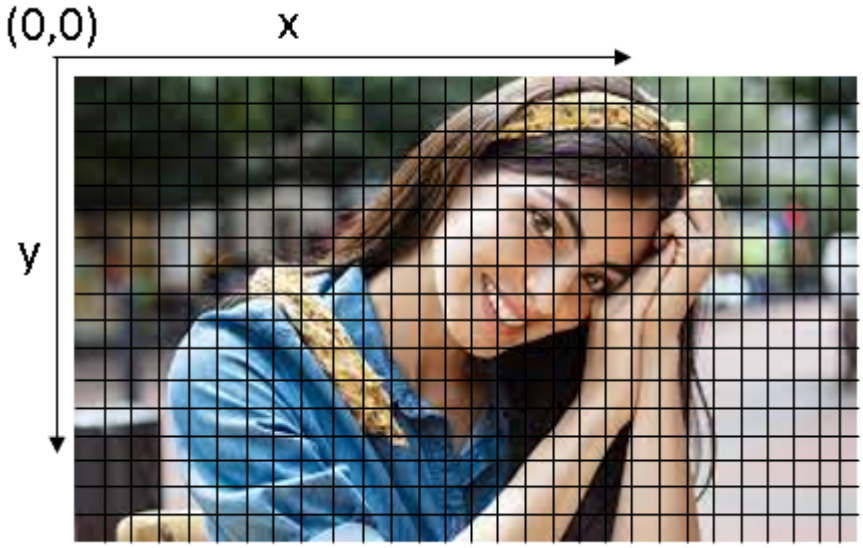

Figura 1.4: Imagen Digital en coordenada (x,y)

1.2.3. Imagen integral

El concepto de imagen integral, ya que su cálculo será fundamental a la hora de extraer las características, reduciendo significativamente el tiempo de procesado de la imagen. Matemáticamente una imagen no es más que una matriz, donde el valor de cada píxel representa un color o tonalidad. La imagen integral es una transformación de la imagen en la que cada elemento es la suma de todos los pixeles que tiene por encima y a la izquierda[20]. Como se muestra en la siguiente ecuación, donde $ii(x,y)$, representa cada uno de los términos de la imagen integral:

$$D_{Ida,vuelta}(cm) = \frac{344m}{s} \times \frac{100cm}{1m} \times T(S) \tag{1.1}$$

La imagen integral se puede calcular rápidamente de una sola pasada por todos los pixeles

de izquierda a derecha y de arriba a abajo de la siguiente manera:

$$ii(x,y) = ii(x,y-1) + i(x-1,y) + i(x,y) - ti(x-1,y-1) \tag{1.2}$$

donde $ii(x,-1)$= 0 y $ii(-1,y) = 0$

A partir de la imagen integral, la suma de píxeles de cualquier rectángulo del interior de la imagen, como se muestra en la figura, se puede calcular con tan solo cuatro elementos.

$$RecSum = ii(x-1,y-1) + ii(x+w-1,y+h-1) - ii(x-1,y+h-1) - ii(x+w-1,y-1) \tag{1.3}$$

Donde w y h son la altura y ancho del rectángulo medida en pixeles

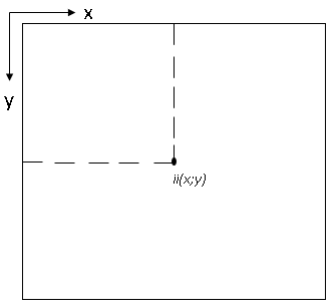

Figura 1.5: Representación de la imagen integral

1.2.4. Geometría de la imagen

Hay dos partes en el proceso de formación de la imagen:

- La geometría de la formación de la imagen, que determina en qué lugar del plano de la imagen se ubicará la proyección de un punto en el escena que sera localizada.
- La física de la luz, que determina el brillo de un punto en el plano de la imagen en función de la iluminación de la escena y las propiedades de la superficie.

Aquí se ve la geometría de la formación de imágenes. Aunque no es necesario comprender la física de la luz para comprender los fundamentos de la mayoría de los algoritmos de visión, este conocimiento suele ser útil para construir sistemas de visión.

El modelo básico para la proyección de puntos en la escena sobre el lugar de la imagen se esquematiza en la Figura 1.6 En este modelo, el centro de proyección del sistema de imágenes

coincide con el origen del sistema de coordenadas tridimensionales. El sistema de coordenadas para los puntos en la escena es el espacio tridimensional abarcado por los vectores unitarios x, y, z que forman los ejes.

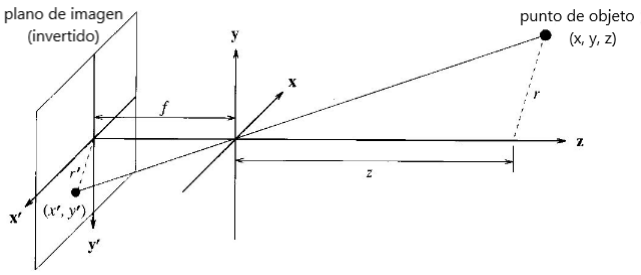

Figura 1.6: El punto en el plano de la imagen que corresponde a un punto particular de la escena se encuentra siguiendo la línea que pasa por el punto de la escena y el centro de proyección. del sistema de coordenadas.

Un punto de la escena tiene coordenadas (x, y, z). La coordenada x es la posición horizontal del punto en el espacio como se ve desde la cámara, la coordenada y es la posición vertical del punto en el espacio como se ve desde la cámara, y la coordenada z es la distancia desde la cámara al punto en espacio a lo largo de una línea paralela al eje z. La línea de visión de un punto de la escena es la línea que pasa por el punto de interés y el centro de proyección. La línea trazada en la Figura 1.6 es una línea de visión.

El plano de la imagen es paralelo a los ejes x y y del sistema de coordenadas a una distancia f del centro de proyección, como se muestra en la figura hay que tener en cuenta que el plano de la imagen en una cámara real está a una distancia de f detrás del centro de proyección y la imagen proyectada está invertida. Es habitual evitar esta inversión asumiendo que el plano de la imagen está enfrente del centro de proyección como se muestra en la Figura 1.7.

El plano de la imagen está atravesado por los vectores x' y y' para formar un sistema de coordenadas bidimensional para especificar la posición de los puntos en el plano de la imagen. La posición de un punto en el plano de la imagen se especifica mediante las dos coordenadas x' e y'.
El punto $(0, 0)$ en el plano de la imagen es el origen del plano de la imagen. La posición en el plano de la imagen de un punto de la escena se encuentra cruzando la línea de visión con el plano de la imagen según el esquema de proyección que se describe

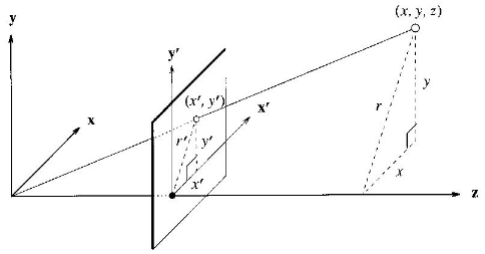

Figura 1.7: Una ilustración que muestra la línea de visión que se utiliza para calcular el punto proyectado (x',y') desde el punto del objeto (x,y,z).

1.2.4.1. Proyección en perspectiva

La posición (x',y') en el plano de la imagen de un punto en la posición (x,y,z) en la escena se encuentra calculando las coordenadas (x',y') de la intersección de la línea de visión que pasa por el punto de la escena (x,y,z) con el plano de la imagen como se muestra en la Figura 1.7

La distancia del punto (x,y,z) desde el eje z es $r = \sqrt{x^2+y^2}$, y la distancia del punto proyectado (r',y') desde el origen del plano de la imagen es $r' = \sqrt{x'^2+y'^2}$. El eje z, la línea de visión al punto (x, y, z), y el segmento de línea de longitud r desde el punto (z,y,z) al eje z (perpendicular al eje z) forman un triángulo.

El eje z, la línea de visión al punto (x',y') en el plano de la imagen, y el segmento de recta de longitud r' desde el punto (x',y') al eje z (perpendicular al eje z) forman otro triángulo. Los dos triángulos son similares, por lo que las proporciones de los lados correspondientes de los triángulos deben ser las mismas.

$$\frac{f}{z} = \frac{r'}{r} \tag{1.4}$$

El triángulo formado por las coordenadas x y y la distancia perpendicular r y el triángulo formado por las coordenadas del plano de la imagen r', y' y la distancia perpendicular r' también son triángulos similares:

$$\frac{x'}{x} = \frac{y'}{y} = \frac{r'}{r} \tag{1.5}$$

La combinación de las ecuaciones y produce las ecuaciones para la proyección en perspectiva:

$$\frac{x'}{x} = \frac{f}{z}$$

and

$$\frac{y'}{y} = \frac{f}{z} \tag{1.6}$$

la posición de un punto (x, y, z) en el plano de la imagen viene dada por las ecuaciones

$$x' = \frac{f}{z}x \tag{1.7}$$

$$y' = \frac{f}{z}y \tag{1.8}$$

1.2.5. Píxel

Por o general todas las imágenes se encuentran formadas por un arreglo o matriz de puntos, denominados pixeles[24]. El pixel se denomina como la unidad de color homogénea más pequeña de una imagen. Como se puede apreciar a continuación las figuras de izquierda a derecha , existe un acercamiento a la imagen hasta llegar a apreciar los pixeles. En este caso se pueden observar dieciocho (18) pixeles, donde cada pixel contiene un color homogéneo, cuando se trabaja con imágenes digital RGB, cada pixel es la combinación de tres colores rojo, verde y azul.

El numero de combinaciones posibles que cada pixel, dependen del tipo de dato que se codifique el pixel, por ejemplo un pixel codificado en un byte (8 bit) que corresponden a 256 variaciones de color para cada pixel, que el RGB se traduce a 224(16.777.216) variaciones de color para cada pixel, porque en RGB intervienen tres colores (256*256*256 =16.777.216)[24].

Figura 1.8: Representación de un pixel

Cada pixel en una imagen se identifica mediante un par ordenado de números naturales. Por ejemplo, el pixel (1, 4) es la celda ubicada en la columna 1 (contada de izquierda a derecha) y en la fila 4 (contada de arriba hacia abajo) de la imagen .

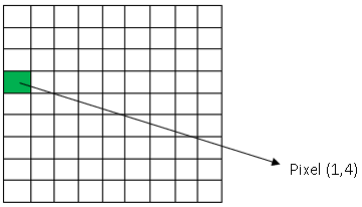

Figura 1.9: Identificación de los pixeles de una imagen

1.2.6. Representación de la imagen

Cuando se representa un imagen por lo general la mayoría de información nos proporciona el sentido de la vista. Pero hay veces que se quiere reafirmar lo observado y se recurre a otro sentidos como es el tacto que nos dice su forma y textura.

Al tener una fotografiar, también se tiene que transmitir toda la información necesaria, como su forma, textura y color, en la imagen.

1.2.6.1. Color

El color es una característica que se encuentra presente en el entorno, lo vemos tanto en los objetos creados como en la naturaleza. Además, es independiente al tamaño de la imagen y a su orientación. Pero el color de un objeto es por la reflexión que existe de las ondas en el espectro electromagnético de luz [25].

Para la vista del ser humano solo es visible el espectro de la luz de longitudes de onda entre 400 nm (violeta) y 700 nm (rojo) como se muestra en la Figura 1.10 .

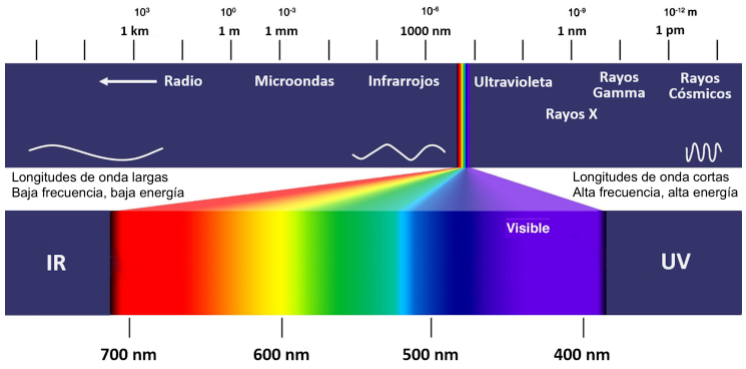

Figura 1.10: Espectro Electromagnético de luz visible

1.2.6.2. Textura

La textura en una imagen se pueden reconoce, pero es difícil describirla. Lo que se diferencia del color, por lo general la textura se puede distingue en una región y no en un punto. Las texturas pueden ser definidas como patrones homogéneos visuales que se presentan en materiales, como madera, piedras, telas, etc. Las texturas no se pueden percibir fácilmente como objetos aislados. La textura tiene cualidades como periodicidad y escala, también se puede distinguir en términos de dirección, contraste y aspereza.

Muchos objetos o regiones no son uniformes, sino están compuestas de pequeños elementos indistinguibles y entrelazados que en general se conoce como “textura”.

La Figura 1.11 . Muestra ejemplos de diferentes tipos de texturas. En algunos casos existe texturas elementos básicos o primitivos que se puede distinguibles, como las texturas de frijoles, ladrillos y monedas.Para los otros casos es más difícil definir.

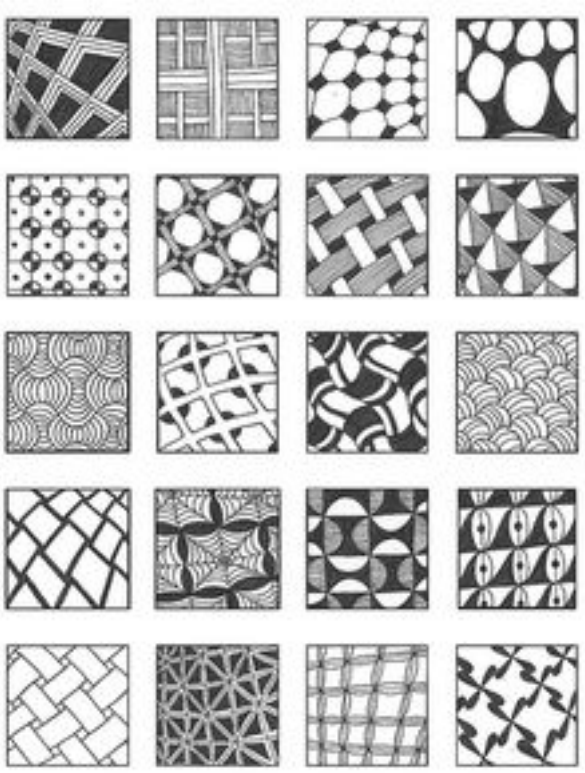

Figura 1.11: Ejemplos de Textura

La textura en una imagen tiene que ver mucho con la resolución. Ya que a una distancia se puede distinguir los objetos, y al tender otras distancias el objeto puede parecer una región uniforme.

Para facilitar los sistemas de vision artificial es recomendable analizar y reconocer diferentes tipos de textura es útil ya que sirve para el reconocimiento de ciertas clases de objetos e incluso en otros aspectos de forma tridimensional[24]. Existen diferentes formas de describir los tipos de textura, que se clasifican en:

- Modelos estructurales

- Modelos estadísticos
- Modelos espectrales

1.2.7. Técnicas para Procesamiento de la Imagen

Las técnicas de procesado de imagen se utiliza para propósitos de mejorar su calidad, Estas técnicas es un conjunto de transformaciones y algoritmos. Por lo tanto la visión artificial consiste en la aplicación de técnicas de procesado de imagen y datos para extraer información de las imágenes

1.2.7.1. Escala de Grises

Para el procesamiento de la imagen en escala de grises se toma en cuenta la luminosidad o intensidad del píxel, que va de 0 que es negro a 255 que es blanco. Aqui el tono de gris de cada píxel se puede obtener bien asignándole un valor de brillo que va de 0 (negro) a 255 (blanco). [23].

1.2.7.2. Binarización

La binarización de una imagen es semegante de la Umbralización debido que crea una imagen de salida binaria a partir de una imagen de grises donde todos los valores de gris cuyo nivel está en el intervalo definido por p1 y p2 son transformados a 255 y todos los valores fuera de ese intervalo a 0 [23].

1.2.7.3. Umbralización

La umbralización es también es conocida como thresholdinges una técnica de segmentación de imágenes en la que cada pixel pertenece obligatoriamente a un segmento y sólo uno. Si el nivel de gris del pixel es menor o igual al umbral se pone a cero si es menor se pone a 255 En función del valor umbral que se escoja el tamaño de los objetos irá oscilando [23].

1.3. Algoritmos de regresión para Detección de Personas

1.3.1. Métodos existentes para la detección

En la actualidad existen diversos algoritmos que puede utilizar para realizar la detección de personas, por lo que es muy importante conocer acerca de las distintas técnicas existentes diseñados principalmente para localización. Por ejemplo, se puede observar en Yang et al[15].

- Métodos basados en conocimientos: Se basado en reglas que codifican el conocimiento humano de lo que constituye una cara típica para captura la relación entre características.

- Enfoques de características invariantes: Tienen como objetivo de encontrar características estructurales que existen, el enfoque o las condiciones de iluminación varia por ejemplo el color de piel, rasgos faciales o texturas.

- Métodos de correspondencia de plantillas: Por lo general los patrones estándar de la cara se almacenan para describir la cara como un todo o como características faciales separadamente. Para la detección se calculan las correlaciones o algún tipo de distancias entre la imagen de entrada y los patrones almacenados.

- Métodos basados en apariencia: Estos métodos son entrenados a partir de un conjunto de imágenes que captura la variabilidad representativa. Estas plantillas aprendidas son utilizadas para la detección. Se basan en técnicas de análisis estadístico o algoritmos de aprendizaje.

Los algoritmos de regresión se utilizan para predecir o estimar cierta una variable o medida partiendo de alguna otra medida. Por lo que la relación entre dos o más variables a través de un modelo formal supone contar con una expresión matemática que permite realizar predicciones de los valores que tomara una de las dos variables que puede ser una variable dependiente a partir de los valores de la otra variable independiente[21].

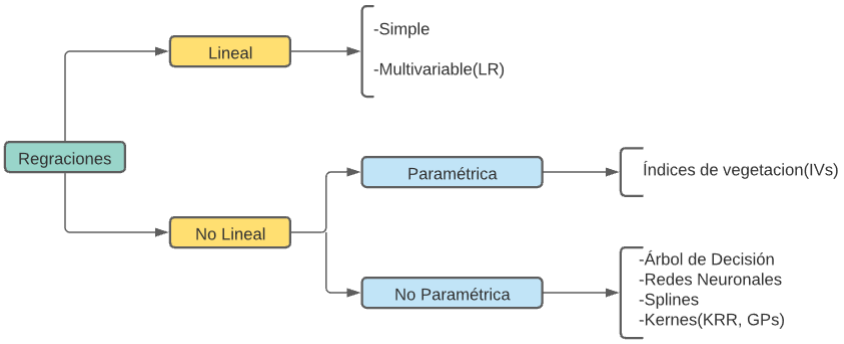

Figura 1.12: Tipos de Regresiones

1.3.2. Regresión lineal

Se produce cuando la relación entre las variables se asume lineal es decir cuando tenemos varias variables independientes, esto se debe a que estas variables independientes, están relacionadas entre sí. Lo que ocasiona un problema de colinealidad entre las variables que pueden enmascarar la relevancia de cada una de ellas, y dificulta la estimación de los pesos del modelo. Este problema se conoce a menudo como sobreajuste y se suele aliviar bien con selección de variables o también con reducción de la dimensionalidad.
La formulación de la regresión lineal regularizada (LR) es la siguiente. Sea $x_i \in R^d$ (espectros) e $y_i \in R^M$(para este caso $M = 3$, ya que tenemos 3 variables de salida a predecir, Chla, LAI y

FVC), donde $i = 1, ..., n$indica el índice de las muestras de entrenamiento. En notación matricial, el modelo viene dado por:

$$Y = XW + b \tag{1.9}$$

Donde X es la matriz de las muestras, $[x_1, x_2, ..., x_n]^T$ y cuya dimensión es $n * d$, y por tanto, contiene los n espectros en las filas de X, la matriz Y contiene los parámetros biofísicos para cada muestra y es de tamaño $n * M$, la matriz W contiene por tanto $d * M$ pesos de las regresiones, y b es un término de sesgo por parámetro biofísico. Por lo cual se asume asumido un modelo de ruido $\hat{Y} = Y + E$ con ruido Gaussiano $E \sim N(0, \sigma_n^2 I)$, cuya media es cero y desviación estándar σ_n.

1.3.3. Regresión no lineal

Cuando nos encontramos con situaciones en las que la relación entre las variables dependiente e independiente se asume no lineal. Este tipo de regresión se puede implementar con muchos métodos: diseñando funciones no lineales ad hoc sobre las variables de entrada, o bien sin asumir una relación explícita entre variables mediante arboles de decisión, redes neuronales, splines, o métodos núcleo (kernel).

- Maquina de Soporte Vectorial
  Las máquinas de soporte vectorial es un algoritmo de clasificación de patrones binarios. Este algoritmo designa para cada patrón una clase a la que mejor se adapte. Para optimizar este algoritmo se utiliza el método de la formulación de LaGrange[26]. este algoritmo tambien se lo conoce como Teoría del Aprendizaje Estadístico ya que al comienzo se utilizaba para problemas de clasificación binaria, pero luego se utilizo problemas de regresión, agrupamiento, clasificación multiclase, regresión ordinal.

  Las máquinas de soporte vectorial fué tomando espacio en proyectos de reconocimiento debido a que mostraron mejores resultados que al utilizar las redes neuronales en el reconocimiento de letra manuscrita. Además, se empezó a utilizar como campo de aplicación para el reconocimiento, la detección facial entre otros.

- Redes Neuronales
  Las Redes neuronales son conjuntos de procesadores muy simples que se encuentran conectados entre si para formar lo que se conoce un modelo simplificado del cerebro. Una neurona artificial tiene, generalmente, varias entradas y una salida. La salida es una función de la suma de las entradas, multiplicadas por "pesos.ªsociados a las conexiones entre neuronas. En algunos casos tienen conexiones al mundo externo [27].

  En la actualidad las redes neuronales se aplica normalmente, como elementos clasificadores o memorias asociativas, Por ellos se debe entrenar a la red, con varios ejemplos

positivos y negativos de los objetos de interés, y luego se aplican a toda la imagen detectando la localización de dichos objetos donde se tenga mayor respuesta. Por o que se recomienda modificando sus pesos de los patrones que se encuentran conectados para encontrar la relación deseada.

Para este trabajo de grado nos centramos en los kernels. A la hora de realizar la regresión se puede utilizar una aproximación estándar como el KRR y una aproximación Bayesiana no paramétrica con GPs, tanto homocedástica como heterocedástica.

1.3.3.1. Kernel Ridge Regresión

El KRR o máquina de vectores soporte de mínimos cuadrados en otras palabras es la versión kernel de la regresión lineal regularizada LR [22]. En este caso se realizar una regresión lineal de mínimos cuadrados en un espacio de Hilbert, H, de dimensión muy grande (posiblemente infinita) d_H, donde las muestras han sido mapeadas mediante una función de transformación $\varphi(xi)$. En notación matricial, el modelo viene dado por:

$$Y = \phi W + b \tag{1.10}$$

donde ϕ es la matriz de las muestras mapeadas,$[\varphi(x_1), \varphi(x_2), ..., \varphi(x_n)]^T$, y cuya dimensión es $n * d_H$. En este caso, la matriz W es de dimensión muy elevada y en principio desconocida porque se define en el espacio H y el mapeado vectorial ϕ no está definido explícitamente. Hemos asumido un modelo de ruido $\hat{Y} = Y + E$ con ruido Gaussiano $E \sim N(0, \sigma_n^2 I)$, cuya media es cero y desviación estándar σ_n.

1.3.3.2. Proceso Gaussiano estándar

Cuando muestreamos a partir de una distribución Gaussiana obtenemos escalares que siguen en conjunto esa distribución. En cambio, cuando muestreamos a partir de un proceso Gaussiano, lo que obtenemos son funciones que siguen una distribución Gaussiana. Esto resulta beneficioso en regresión ya que en GPs asumimos que cada punto o muestra proviene de una combinación de un número elevado de escalares que provienen de una función Gaussiana. La regresión basada en un proceso Gaussiano define una distribución sobre funciones $f : X ß R$ completamente descrita por una media m:X $\rightarrow$ Ry una función de covarianza (kernel) $k : X * X ß R$ de tal manera que:

$$m(x) = E[f(x)] \tag{1.11}$$

$$k(x,x') = E[(f(x) - m(x))(f(x') - m(x))] \tag{1.12}$$

A partir de aquí, consideraremos como la función cero para simplificar. Así pues, dado un conjunto finito de datos de muestras $x_1, x_2, ..., x_n$, primero calculamos su matriz de covarianza K

del mismo modo que se calcula la matriz de Gram en KRR. La matriz de covarianza define una distribución sobre el vector de valores de salida $f(x) = (f(x_1), ..., f(x_n))^T$, de tal manera que $f_x \sim N(0; K)$, que es una distribución gaussiana multivariante. Por lo tanto, el tipo de función de covarianza implica la forma de la distribución sobre de las funciones. El papel de la covarianza en un GP es el mismo que el papel del kernel en los KRR: en ambos casos nos da la información de la similitud entre las muestras Asumimos que nuestra variable observada está formada por observaciones ruidosas de la verdadera función subyacente, es decir, tenemos nuestra función, más un ruido:

$$y = f(x) + \in$$

Además, se supone que el ruido añadido es Gaussiano con media cero y varianza $\sigma_n : \rho \sim N(0, \sigma_n^2 I)$. Definimos los valores de salida $y = (y_1, ..., y_n)^T$, los términos de covarianza de testeo $k_* = (k(x_*, x_1), ..., k(x_*, x_n xn)), yk_(**) = k(x_*, x_*)$.
Del modelo anterior, los valores de salida se distribuyen de acuerdo a:

$$\begin{pmatrix} y \\ f(x_*) \end{pmatrix} \in \sim N\left(0 \begin{pmatrix} k + \sigma_n^2 I & k_* \\ k_*^T & k_** \end{pmatrix}\right) \tag{1.13}$$

Para predecir, el GP se obtiene mediante el cálculo de la distribución condicional

$$f(x_*)|y, x_1, ..., x_n; x_* \tag{1.14}$$

que se puede escribir como una distribución Gaussiana de media

$$E[f(x_*)] = k_*^T (K + \sigma_n^2 I)^- 1y \tag{1.15}$$

,y de varianza

$$V[f(x_*)] = k_** - k_*^T (K + \sigma_n^2 I)^- 1k_* \tag{1.16}$$

Capítulo 2

Metodología

Mediante explicaciones del sistema, el usuario puede aprender progresivamente los conocimientos necesarios para resolver el problema ya mencionado, así como su programación en Python.

2.1. Python

Python como se mencionó antes es un lenguaje de programación de código abierto, orientado a objetos, muy simple y fácil de entender. Tiene una sintaxis sencilla que cuenta con una vasta biblioteca de herramientas, que hacen de Python un lenguaje de programación único.

La ventaja principal que se tomó en cuenta para realizar el sistema en este lenguaje de programación es la posibilidad de crear un código con gran legibilidad, que ahorra tiempo y recursos, lo que facilita su comprensión e implementación.

Además, está el hecho que Python es uno de los idiomas de programación más utilizados para el desarrollo de la inteligencia artificial.

2.2. Modelo del Sistema

El análisis y diseño del algoritmo del sistema facilita el uso eficiente del poder de las computadoras para resolver problemas. Para facilitar el desarrollo de esta solución, es adecuado usar un lenguaje computacional simple, general y eficiente como el que ofrece Python. El siguiente gráfico describe los pasos en la solución computacional de un problema

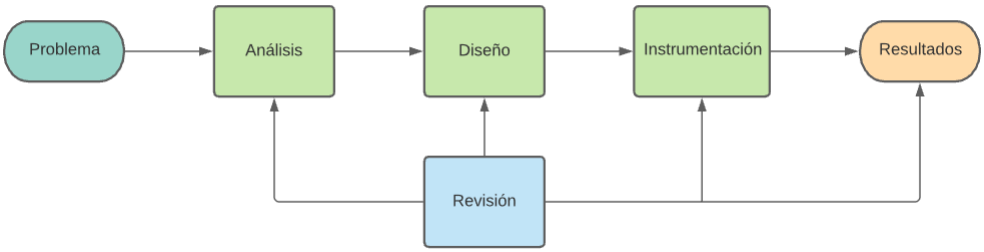

Figura 2.1: Modelo de Sistema

2.2.1. Algoritmo del sistema

En algoritmo tiene como objeto comunicarse con el entorno. Por lo tanto, debe incluir facilidades para el ingreso de datos y la salida de resultado.

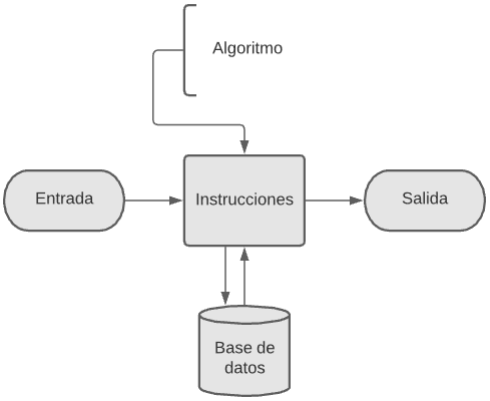

Figura 2.2: Diseño del Algoritmo

Es decir este algoritmo tiene:

- Entrada:
  En la entrada o input del algoritmo será donde se introduzcan todos aquellos datos que el algoritmo necesite para operar en este caso las imágenes de multitudes.

- Procesamiento:
  Con los datos recibidos en la entrada o input, el algoritmo realizará una serie de cálculos

lógicos como reconocimiento, conteo y la distancia de seguridad para resolver el problema.

- Salida:
  Los resultados obtenidos en el procesamiento se mostrarán en la salida u output del algoritmo.

Para este algoritmo se tuvo en cuenta los procesos de forma ordenada es decir realizar uno después del otro ya que cuenta con un número determinado de pasos; así para lograr tener mayor precisión al detectar los peatones, para así cumplir con los objetivos de resolver el problema de aglomeraciones; y por supuesto mostrar el resultado al problema resuelto, ya que los mismos inputs siempre deben obtenerse los mismos outputs.

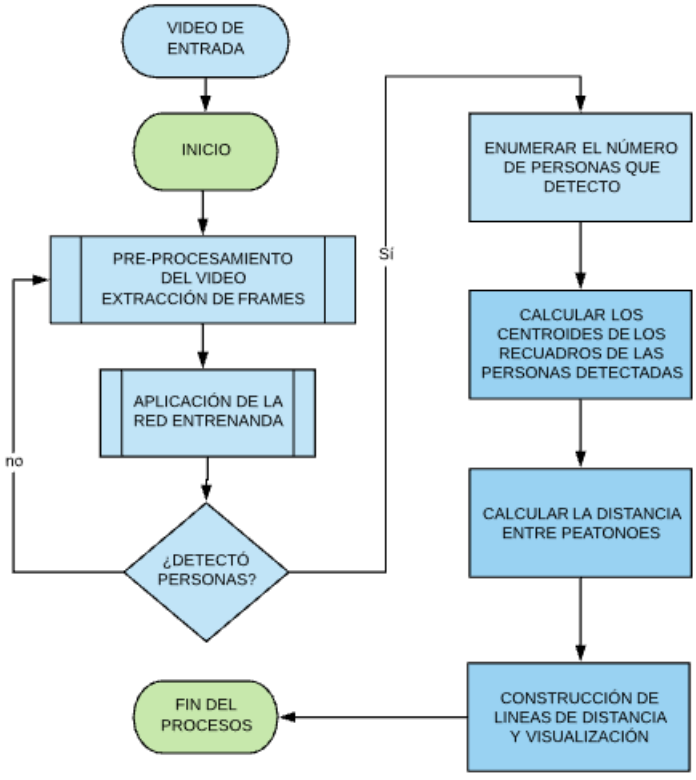

Figura 2.3: Diagrama de Bloque del Algoritmo

En resumen, el algoritmo es suficiente satisfactorio para resolver el problema aglomeraciones y ante varios algoritmos que resuelvan el mismo problema.

2.3. Desarrollo de Algoritmo

Para el desarrollo del algoritmo se trabajo con el diagrama de bloque de la figura 2.3

2.3.1. Vídeo de Entrada

Como inicio del sistema se tiene los datos de entrada. Por ello se generó a partir, de vídeos de una webcam de computadora y vídeos descargados en internet. Ya que para trabajar con estos datos se realiza una preevaluación de la misma, para encontrar la altura máxima que tendría que estar la cámara de vigilancia, para evitar errores de la distancia entre la persona el cual en tendría que será estar una altura máxima de 8 metros

2.3.2. Bases de Datos

Una base datos es una recopilación de datos que corresponde al contenido de una tabla de base de datos única, o una matriz de datos estadística única, donde cada columna de la tabla representa una variable particular y cada fila corresponde a un miembro dado del conjunto de datos en cuestión.

Figura 2.4: Base de Datos

Para realizar proyectos como un algoritmo en Machine Learning, necesitamos un conjunto de datos de capacitación. Es el conjunto de datos real utilizado para entrenar el modelo para realizar varias acciones.

Para el entrenamiento del algoritmo es necesario tener en gran capacidad de datos, debido a que, sin un base de datos, es imposible que un algoritmo pueda aprenda. Por lo que el tamaño del conjunto de datos es indispensable para empezar a entrenar un algoritmo, ya que, si el conjunto de datos no es lo suficientemente bueno, no podría detectar de manera correcta y el proyecto podría funcionar de forma erróneo.

Para la fase de entrenamiento del algoritmo se creó una base de datos con 1900 fotos descargadas de Internet, las que contenía n cantidad de personas en diferentes lugares, en el grupo de fotos que existe en la base de datos existe: en el día en la tarde, en la noche, temporada de lluvia y nieve. Para luego realizar el recorte de las personas llegando a obtener más de 3000 recortes de peatones.

2.3.3. Pre-Procesamiento del Vídeo

2.3.3.1. Proceso de Segmentación del fondo o Background subtractor

La sustracción de fondo o background sustractor es una técnica que nos permite detectar objetos en movimiento (foreground o primer plano), mientras que descarta el resto de la escena (background o fondo), ya que permanecerá sin movimiento. Claro, esto se da al emplearse una cámara estática que estará captando toda la escena.

- BackgroundSubtractorGMG
  Según OpenCV: "Este algoritmo combina la estimación estadística de la imagen de fondo y la segmentación bayesiana por píxel". Algo que hay que tener en cuenta y nos lo dice la documentación, es que este algoritmo utiliza los primeros fotogramas (120 por defecto) para el modelado de fondo, entonces no se podrá observar nada en los primeros fotogramas.

- BackgroundSubtractorMOG
  Según OpenCV: "Es un algoritmo de segmentación de fondo / primer plano basado en una mezcla gaussiana".

- BackgroundSubtractorMOG2
  Para el desarrollo del sistema el más conveniente es este algoritmo ya que Al igual que en el anterior algoritmo, OpenCV señala que: "Es un algoritmo de segmentación de fondo / primer plano basado en una mezcla gaussiana". Pero, además, "Proporciona una mejor adaptabilidad a diferentes escenas debido a cambios de iluminación".

2.3.3.2. Extracción de Frame del Vídeo de Entrada

2.3.4. Conseguir el vídeo y Separación frame a frame

La adquisición del vídeo es la primera parte para obtener los datos de imágenes. Estas imágenes son usadas enseguida para el procesamiento en los distintos bloques. Ya que un vídeo es una secuencia de imágenes consecutivas, se divide el vídeo en frames y se trabaja en cada uno de ellos. Para realizar la división del vídeo en frames se requiere un proceso el cual se relata

a continuación:

En primer lugar se obtiene las imágenes a procesar con ayuda de una interfaz gráfica que permite subir un archivo de vídeo previamente existente o capturar una señal de vídeo en vivo. Luego de haber sido cargado el vídeo al módulo, se continua a capturar cada una de las imágenes que componen la secuencia de vídeo.

En esta etapa luego de haber cargado el vídeo al sistema. Se convierte el vídeo original preparado para ser procesado frame a frame (ver Figura 3.5). Por último se implementa la división del mismo en imágenes para ir moviéndose por todas y cada una de ellas [24].

Figura 2.5: División del vídeo en Fotogramas

2.3.5. Metodológica del entrenamiento de la red

2.3.5.1. Entrenador

El reconocimiento de peatones mediante un algoritmo es utilizar una herramienta como una computadora, que pueda identificar a las personas de la misma forma como lo hacen los ojos

humanos para poder analizar e interpretar.

Debido al uso de la computadora se puede afirmar que es más eficientes que el ojo humano debido que puede funcionar sin descanso y procesar innumerables imágenes de manera rápida y arrojar datos necesarios para tomar decisiones.

Además, como objetivo principal del proyecto es detectar los peatones en aglomeraciones, para ellos se plateo las deferentes tareas que debe cumplir el algoritmo como es:

- Problemas de detección para detectar a los peatones
- Problemas de distanciamiento como indicar la distancia entre peatones menor a 2 metros.

Arquitectura de la red YOLO es un tipo de inteligencia artificial que procesa la imagen para extraer características de imágenes. Lo que hace el algoritmo es procesar la imagen, pasándola a escala de grises y detectando los bordes de los objetos con un filtro.

Además, los anchos que obtiene en el entrenamiento sirven para calcular un rectángulo que se adapte a la persona en la imagen Reduciendo varios cuadros a uno solo. para comparar donde podría estar el objeto y optimiza el cuadro de detección.

2.3.6. Cálculo del Centroide del Recuadro de la Persona Detectada

Para encontrar el centro de cada persona se usa los momentos de cada contorno. Los momentos son una medida particular que indica la dispersión de una nube de puntos, y matemáticamente se definen como:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y) \tag{2.1}$$

donde x,y son las coordenadas de un píxel y la función I(x,y) indica su intensidad. Estamos trabajando sobre una máscara binaria, por tanto la función I(x,y) sólo puede valer 0 ó 1.

Bien, ¿cómo puede ayudarnos esta fórmula extraña a saber el centro? Hay tres momentos que nos interesan: M00, M01 y M10. Fijémonos que, si estamos calculando M00, la fórmula anterior se transforma en:

$$M_{00} = \sum_x \sum_y x^0 y^0 I(x,y) = \sum_x \sum_y I(x,y) \tag{2.2}$$

(recordad que $a^\circ = 1$, y aquí definimos $0^\circ = 1$)
Como I(x,y) sólo puede dar 0 ó 1, la fórmula de M00 es equivalente al número de píxeles cuyo valor es 1. Por tanto, M00 es el área en píxeles de la región blanca. Para M10, la fórmula original se transforma en esta:

$$M_{10} = \sum_x \sum_y x^1 y^0 I(x,y) = \sum_x \sum_y x I(x,y) \tag{2.3}$$

Esto es igual a la suma de las coordenadas x de los píxeles cuya intensidad sea 1. Por tanto, dividiendo M10 por M00 obtendríamos el centroide para la componente x:

$$\frac{M_{10}}{M_{00}} = \frac{\sum_x \sum_y x I(x,y)}{\sum_x \sum_y I(x,y)} = \frac{x_1 + x_2 + ... + x_n}{n} = \bar{x} \tag{2.4}$$

Y lo mismo con M01:

$$\frac{M_{01}}{M_{00}} = \frac{\sum_x \sum_y y I(x,y)}{\sum_x \sum_y I(x,y)} = \frac{y_1 + y_2 + ... + y_n}{n} = \bar{y} \tag{2.5}$$

Por tanto: para obtener los centroides calcularemos los momentos M00, M10 y M01 de cada contorno y haremos las divisiones anteriores. Después los pintaremos (dibujando un círculo pequeño en su posición) y escribiremos las coordenadas sobre la imagen

2.3.7. Cálculo de la Distancia Entre Personas

2.3.7.1. Pixel metro

Para calcular la distancia entre los peatones en primer lugar se utilizo es la unidad básica de longitud en el Sistema Internacional de Unidades (SI) que es el metro (m), el cual se define como la longitud de la trayectoria recorrida por la luz en el vacío durante un intervalo de tiempo que es de 1/299. 792.458 de un segundo.

El píxel en la tecnología de computadores e imágenes digitales es un punto físico en una imagen de trama, o el elemento identificable más pequeño en un dispositivo de visualización. Su longitud es diferente para los diferentes monitores. La ubicación de cualquier píxel corresponde a sus coordinadas físicas en la pantalla.

Para calcular la distancia entre los peatones el tamaño del píxel es 0,265 mm y corresponde aproximadamente al tamaño de un píxel de un monitor de 17 pulgadas de 1280 * 1024 La fórmula para convertir Metro a Píxel es 1 Metro = 3779.52755905488 Píxel. Es decir el metro es 3779.5276 veces más grande que Pixel.

2.4. Librerías Utilizadas

Para el desarrollar un sistema de visión artificial es imprescindible elegir las librerías adecuadas. Ya que no se va trabajar con solo una, debido que los defectos de una puede ser resueltos con otra.

A continuación se detallan las librerías más utilizadas para el desarrollo del sistema de Visión Artificial.

2.4.1. OpenCV

OpenCV es una bibliotecalibre de visión artificial originalmente desarrollada por Intel. Esta librería proporcionaría un marco de trabajo de nivel medio-alto que ayudaría al personal docente e investigador a desarrollar nuevas formas de interactuar con los ordenadores. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos.

La librería OpenCV es una API de aproximadamente 300 funciones escritas en lenguaje C que se caracterizan por lo siguiente:

- Su uso es libre tanto para su uso comercial como no comercial [29].
- No utiliza librerías numéricas externas, aunque puede hacer uso de alguna de ellas, si están disponibles, en tiempo de ejecución.

El módulo OpenCV se importa como:

import cv2

2.4.2. Numpy

NumPy es una librería de Python que proporciona un objeto de matriz multidimensional, varios objetos derivados como matrices. Tiene una variedad de rutinas para operaciones rápidas en matrices, incluidas matemáticas, lógicas, manipulación de formas, clasificación, selección, Transforma Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.[30]

Entre las matrices NumPy y las secuencias estándar de Python son:

- Los arreglos NumPy tienen un tamaño fijo en el momento de la creación, a diferencia de las listas de Python (que pueden crecer dinámicamente). Cambiar el tamaño de un array creará una nueva matriz y eliminará la original.

- Se requiere que todos los elementos en una matriz NumPy sean del mismo tipo de datos y, por lo tanto, tendrán el mismo tamaño en memoria.
- Los arreglos NumPy facilitan operaciones matemáticas avanzadas y otros tipos de operaciones en grandes cantidades de datos. Típicamente, tales operaciones se ejecutan de manera más eficiente y con menos código de lo que es posible usando las secuencias integradas de Python.

El módulo Numpy se importa como:

import numpy as np

2.4.3. SYS

El módulo sys proporciona información acerca constantes, funciones y métodos del intérprete de Python. Puede ofrece un resumen de las constantes, funciones y métodos[31].
El módulo sys se importa como:

import sys

2.4.4. Tensorflow

TensorFlow es un software de computación numérica, creado por Google, orientado a problemas de Deep Learning. Deep Learning es un área específica de Machine Learning que está tomando gran relevancia dentro del mundo de la Inteligencia Artificial y que está detrás de algunos de las novedades tecnológicas más sorprendentes de los últimos años.

En febrero de 2017, Google liberó la versión 1.0, que incorpora multitud de mejoras. Algunas de ellas es el hecho que mejora el rendimiento, es decir permiten acelerar hasta 58 veces los tiempos de ejecución de algunos algoritmos y aprovechar las ventajas de ejecutar sobre GPU[32].

En este caso, he usado TensorFlow 1 con la versión 1.13.2 de TF Object Detection AP

El módulo tensorflow se importa como:

import tensorfloe

2.4.5. Keras

Keras es un módulo de aprendizaje profundo escrita en Python, que se ejecuta sobre la plataforma de aprendizaje automático TensorFlow .El módulo Keras es una secuencia de capas que cada una de ellas va "destilando" gradualmente los datos de entrada para obtener la salida deseada[33].

En Keras podemos encontrar todos los tipos de capas requeridas y se pueden agregar fácilmente al modelo mediante el método add( ). Además,Keras ayuda a mejorar la ejecución de nuevos experimentos, de manera mas rapida.

En este caso, he usado Keras con la versión 2.0.8

El módulo Keras se importa como:

import keras

2.4.6. PIL

PIL es una librería gratuita que permite la edición de imágenes directamente desde Python. Además este modulo soporta una variedad de formatos, como es el caso como GIF, JPEG y PNG. Una gran parte del código está escrito en C, por cuestiones de rendimiento[34].

Esta librería soporta únicamente hasta la versión 2.7 de Python.

El módulo PIL se importa como:

import pil

2.4.7. Time

Es un módulo llamado time que sirve para manejar tareas relacionadas con el tiempo. Se puede usar las funciones definidas en el módulo.

El módulo Time se importa como:

import time

Capítulo 3

Implementación y Pruebas

En el siguiente capítulo se detalla el procedimiento para realizar la implementación y pruebas del algoritmo que se propuso en la tesis. Así como el análisis de los resultados obtenidos en dichas pruebas.

3.1. Implementación

Para realizar la implementación del algoritmo se basó en los resultados de métodos de seguimiento de personas que en la actualidad existen. Pero se puede deducir que existe un gran problema en cuanto a comparación de métodos debido a que es difícil de encontrar las mismas características de secuencias de imágenes evaluados en métodos anteriores. Ya que para poder hacer una comparación justa es necesario utilizar el mismo grupo de secuencias. Sin embargo, actualmente no existen disponibles de forma pública las características que se proponen en esta tesis, que es la presencia de múltiples personas en veredas o calles.

Debido al problema ya mencionado se buscó una alternativa que es crear una base de datos propia descargando imágenes y videos de internet. Luego probar el funcionamiento del algoritmo implementado y observar los resultados del trabajo de detección de personas.

Las características de las imágenes y videos mencionadas son de seguimiento de personas. Sin embargo, la mayoría de las veces es necesario para el seguimiento de personas un lugar específico como en hospitales, campos militares o escuelas, donde se puede observar aglomeraciones de personas. Ya que los resultados de esta tesis pretenden mostrar el número de personas en la imagen y su distancia.

3.1.1. Metodología de Implementación

Para la implementación del algoritmo se utilizó 5 distintos vídeos con secuencias de imágenes captadas por cámaras de vigilancia y vídeos de internet que utilizan una taza baja fps para disminuir el espacio de memoria. Además, los vídeos tienen una duración entre 6.5s a 11.25s de duración.

Cabe mencionar que los vídeos son captados en diferentes sitios y ambientes con una iluminación no controlada lo que significa que el clima controla el factor de iluminación. En estos vídeos se puede apreciar desde niños hasta personas adultas con diferente postura como puede ser erguido, sentado y desplazándose a diferentes direcciones.

Los peatones que se pueden observar en los vídeos tienen un rango de 1 a 10 metros de distancia a partir de la ubicación de las cámaras. El cual es reflejados en el plano X-Z de sistema de coordenadas con respecto a la cámara como se muestra en la siguiente Figura 3.1.

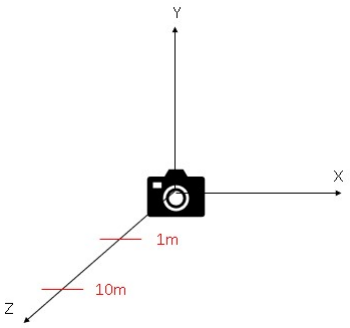

Figura 3.1: Ejes vistos desde el entorno con respecto a la cámara

3.1.2. Pruebas del Entrenamiento

Las pruebas de funcionamiento del algoritmo entrenado se probaron diferentes métodos de identificación de objetos que tras diferentes pruebas se acabaron descartando. Al final se acabó utilizando, en una primera fase, diferentes operaciones secuenciales sobre imágenes de peatones en dos dimensiones, que se realizó de manera personalmente y del mismo modo se creó una base de dato con 100 imágenes descargadas de internet. Para obtener la información necesaria de las imágenes en 2D se prosigue a calcular la distancia real a la que se encuentran los peatones entre sí para ellos se utilizando la información obtenida y procesada previamente en el detector de peatones para obtener los puntos centros de las personas.

Para el Procesamiento de imágenes de multitudes en 2D e identificar a los peatones, en primera instancia se utilizó el proceso que más se ha usado en los últimos años para detectar

diferentes tipos de objetos o personas, que es Machine Learning. Pero como bien se conoce existe otros proceso que funciona de mejor manera como es el Deep Learning, pero utilizar este tipo de proceso es proporcionalmente al coste computacional que tiene que ser mucho mayor y una base de datos bastante grande debido a que el mismo programa detectara y enumerara a los peatones y del mismo modo indicara la distancia de seguridad que se encuentra cada personas. Sin embargo, en el Machine Learning el programador decide qué es peatón y el detector aprende.

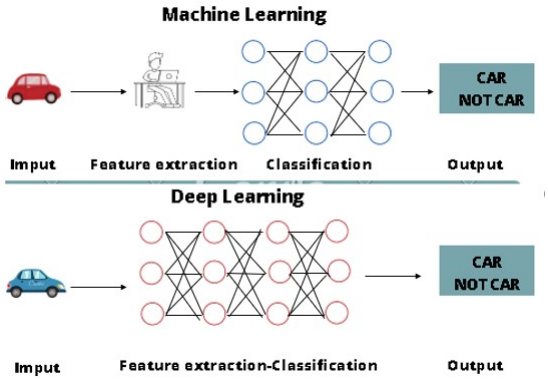

Figura 3.2: Diferencia entre Machine Learning y Deep Learning

Entre los tipos de Machine Learning que existen, se probó primero el basado en Bayes, donde había que enseñar a un detector a un detector introduciendo descriptores que por lo general son numéricos. Pero el problema comenzó al elegir los descriptores, ya que, al intentar incluir los descriptores de perímetro y área, tiene inconveniente porque los valores varían mucho según el tamaño y la orientación del peatón, así como a la distancia a la que estuviera de la cámara. pero no se acabó utilizando debido al gran coste computacional que presenta.

Por lo que se probó posteriormente un Machine Learning basado en los gradientes de las imágenes, el Histogram of Oriented Gradients(HOG). En este nuevo detector, en lugar de introducir las características del peatón, se deben introducir imágenes positivas y negativas. Las imágenes positivas se introducen en primer lugar para “indicarle” al detector que las personas son las personas que va a tener que detectar, o similares. A continuación, se introducen imágenes negativas, que suelen ser objetos que podrían aparecer con los peatones, para “indicarle” en este caso que estos objetos no son peatones, y que no debe detectarlos como tal. Las imágenes positivas debían ser imágenes cuadradas obligatoriamente, del mismo tamaño y donde únicamente apareciera el peatón. Para realizar estas pruebas se utilizaron los recortes de peatones, de más de 1900 imágenes, obteniendo un total de aproximadamente mas de 3000 recortes de imágenes positivas.

Figura 3.3: Tipos de imágenes para el test

En un comienzo se realizó el entrenamiento con 500 imágenes, lo que genero problemas al detectar ya que sobre escribía al peatón ya detectado. Además, obtenía varios falsos positivos lo que se tuvo que corregir los recortes de los peatones para tener los recuadros uniformes de las personas y aumentar la base de datos para prepararle al entrenador.

Con estos últimos valores se consiguió, finalmente, entrenar un detector. Sin embargo, la detección no fue tan buena como se esperaba. Las detenciones sobre escrita el mismo peatón formaban parte del número de peatones detectados, incluso a veces daba como positivo a una sombra, como se puede apreciaren las siguientes imágenes.

Figura 3.4: Errores de la detección

La solución más conveniente fue realizar un nuevo recorte de los peatones en las fotos mezclando escenarios ya sea de día, tarde y algunas en la noche. además, se aumentó en número de fotos a 1900 imagenes obtenidas de buscador de Google, lo que conlleva el aumento de peatones recortadas para entrenar.

Luego de realizar el recorte todas las nuevas imágenes, se entrenó nuevamente al detector, ya obteniendo una salida donde solo se detectaba al peatón y no la sombra, disminuyendo un 99 % las detectaba falsos positivos frecuentemente, por lo que el resultado obtenido fue el esperado y se podía utilizar para calcular las distancias entre personas. Al final se terminó aprobando este este método.

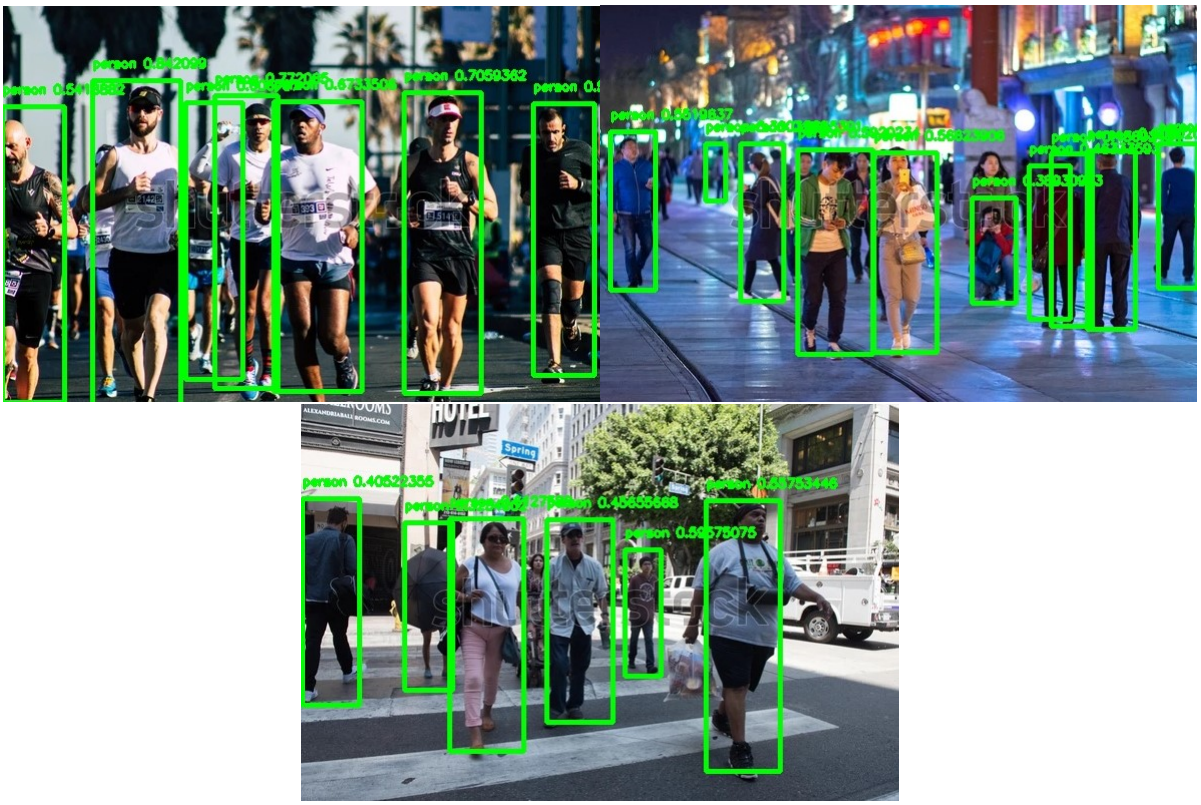

Figura 3.5: Detecciones Finales

por ultimo se realizo la implementación del medidor de distancias en el algoritmo entrenado en detección de personas como se puede observar en la siguiente figura obteniendo un excelente resultado

Figura 3.6: Implementación de la Distancia entre Peatones

3.1.3. Numero de Personas Detectadas

En esta etapa, luego de haber aplicado la red entrenada. Ya se observa los cuadros de personas detectadas el cual se enumera con ayuda del comando ,previamente inicializado la variable en cero

```
AllBoxesNumber = AllBoxesNumber + 1          # Cuenta cuantos rectangulos
```

Figura 3.7: Comando para contar los recuadros

3.1.4. Construcción de Lineas de Distancia y Visualización

Finalmente, luego de haber calculado todos los centroides para nuestras predicciones y la distancia entre ellos. Se puede crear las líneas dx,dy que conectan los centroides y mostrar la distancia en el centro .

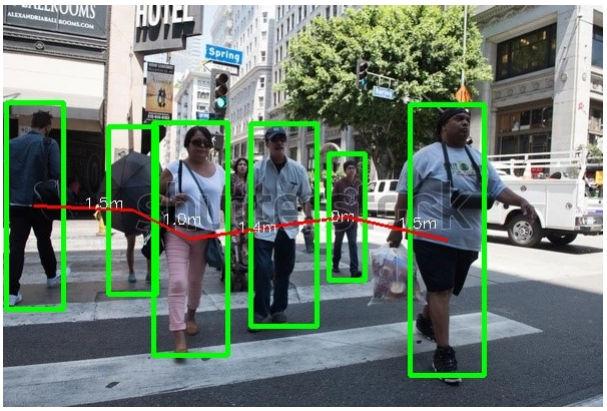

Figura 3.8: Visualización del lineas y distancia

3.2. Pruebas

3.2.1. Evaluación

Cada vídeo fue procesado por el algoritmo propuesto en la tesis donde se obtuvo falsos positivos, falsos negativos y verdadero positivo. En el proceso del seguimiento de personas en imágenes, se define como verdadero positivo al seguimiento correcto de una persona, es decir, la cantidad de veces en una secuencia que la persona es detectada y clasificada correctamente.

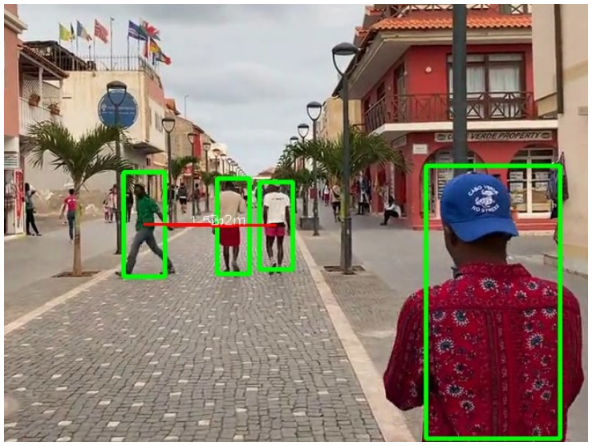

Figura 3.9: Verdadero Positivo

Los falsos positivos son definidos como la cantidad de seguimiento que el algoritmo lo realiza de forma incorrecto o de otro modo es el número de veces confunde una persona con otro objeto.

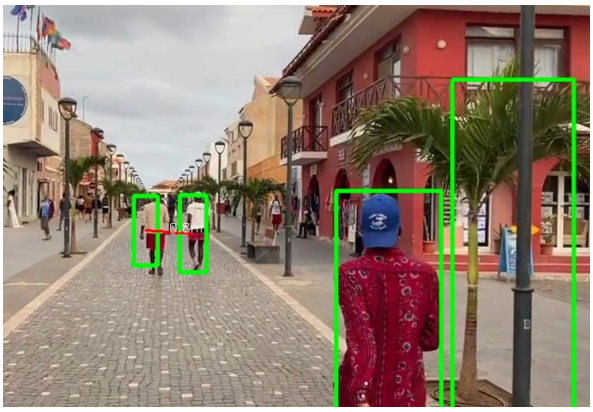

Figura 3.10: Falso Positivo

Por último, se define a los falsos negativos como el número de objetivos o personas que no ha sido detectado por el algoritmo.

Figura 3.11: Falso Negativo

Al final los resultados arrogados por el algoritmo fueron cuantificados de forma visual, conociendo de antemano el número de cada persona en la imagen y comparándola con los resultados arrojados por el algoritmo. Así como las coordenadas del centroide del cuadro de detección para dibujar las líneas de distancia de las personas.

Los videos para la experimentación tienen diferentes características por ellos se realizó una tabla marcando con una “X” donde se especifica las características propias de los videos para la fase de pruebas, entre las características se presenta el número de fotogramas que las conforman el vídeo.

Cuadro 3.1: Características de los Vídeos de Prueba

Videos / Características	1	2	3	4	5
Vídeos capturados en ambientes exteriores	X	X	X	X	
Vídeos con iluminación no controlada	X	X	X		X
Vídeos que tiene múltiples objetivos		X	X		X
Personas entre sí	X	X	X	X	X
Movimiento de los objetivos sobre el eje X	X	X	X	X	X
Movimiento de los objetivos sobre el eje Z	X	X	X		X
Movimiento repentino de los objetivos	X	X	X		X
Número de fotogramas	155	110	60	45	65

Para las pruebas de funcionamiento del algoritmo propuesto para la tesis se realizó con 5 secuencias de vídeo.

3.3. Resultados

Los siguientes recuadros es el resultado de cada secuencias de video de prueba:

Cuadro 3.2: Resultados del experimento con el vídeo número 1

	Algoritmo Propuesto
Verdadero Positivo	**90.72 %**
Falso Positivo	**0 %**
Falso Negativo	**13.26 %**

Cuadro 3.3: Resultados del experimento con el vídeo número 2

	Algoritmo Propuesto
Verdadero Positivo	**78.67 %**
Falso Positivo	**22.72 %**
Falso Negativo	**29.2 %**

Cuadro 3.4: Resultados del experimento con el vídeo número 3

	Algoritmo Propuesto
Verdadero Positivo	**79,51 %**
Falso Positivo	**8.33 %**
Falso Negativo	**13.26 %**

Cuadro 3.5: Resultados del experimento con el vídeo número 4

	Algoritmo Propuesto
Verdadero Positivo	**100 %**
Falso Positivo	**0 %**
Falso Negativo	**0 %**

Cuadro 3.6: Resultados del experimento con el vídeo número 5

	Algoritmo Propuesto
Verdadero Positivo	**78.71 %**
Falso Positivo	**7.69 %**
Falso Negativo	**10.76 %**

El promedio y la desviación estándar de los resultados de cada experimento se presenta en en el siguiente recuadro:

Cuadro 3.7: Promedio de todos los resultados

	Algoritmo Propuesto
Verdadero Positivo	**85.52 %**
Falso Positivo	**7.75 %**
Falso Negativo	**13.29 %**

Cuadro 3.8: Desviación estándar de los Resultados

	Algoritmo Propuesto
Verdadero Positivo	**8.56 %**
Falso Positivo	**8.30 %**
Falso Negativo	**9.33 %**

3.4. Discusión

Como se puede observar de la sección de resultados el seguimiento correcto de peatones o verdadero positivo esta entre 78.67 % y 100 %. Pero esto depende del tipo de secuencias de video debido que la diferencia que puede existir entre una secuencia y otra es la dificultad que estás presentan ya sea falta de luz u movimientos bruscos de las personas, aunque al final se tiene un promedio de 85.52 %. La desviación estándar de los verdaderos positivos es de 8.52 esto hace referencia al grado de precisión del algoritmo entre las 5 secuencias de vídeos.

El algoritmo propuesto se tiene resultados significativos en el porcentaje de falsos positivos y de falsos negativos. También muestra mejores resultados en la desviación estándar lo que se puede asegurar una constancia en el rango de los resultados obtenidos. Además, se puede

notar, que en las secuencias de imágenes donde se aprecia mejores resultados es donde se observa cambios repentinos de trayectoria de los objetivos, por ello el sistemas de seguimiento de personas, se basa en predictores de trayectoria porque los movimientos de las personas suelen cambiar bruscamente de un momento a otro.

Capítulo 4

Conclusiones y Trabajo Futuro

4.1. Conclusiones

En este trabajo se presentó un sistema de conteo de multitudes para en el futuro implementar en las cámaras, para el seguimiento de personas a través de estas. En el sistema se aplicó diferentes técnicas de sustracción de fondo para detectar objetos de interés con las personas y luego aplicar una proyección de los puntos, para ser visualizados la distancia entre sí en un espacio georreferenciado.

El sistema es muy fácil de configurar, permitiendo mejorar la funcionalidad de procesamiento, visualización y seguimiento en cualquier equipo. Analiza técnicas de filtrado de los datos, a fin de obtener un informe del número de personas con mayor precisión y compararlo con los resultados obtenidos del análisis de imágenes.

Además, se concluye con respecto al desarrollo del algoritmo realizado que existe resultados favorables debido al uso de filtro de profundidad que permitió eliminar de manera precisa el fondo de la imagen, para conseguir una correcta segmentación las personas.

Tras culminar el sistema se puede concluir que se ha sido capaz de diseñar un algoritmo de visión artificial eficiente que cumple con los objetivos propuestos al inicio del proyecto: la detección y conteo de personas y obtención de la distancia de seguridad entre sí. Sin embargo, se encontraron varios inconvenientes al trabajar con imágenes que, al trabajar en tiempo real con una cámara, pero el trabajar con este proyecto me ayudo a aumentar mis conocimientos en el ámbito de la programación y visión artificial. De igual forma me dio la oportunidad de comprobar la importancia de las asignaturas vistas en la carrera y para prepararme para futuras experiencias en este campo.

Se concluye que el algoritmo de reconocimiento de peatones tiene un mejor funcionamiento al trabajar en rangos entre 1 m a 10 m de distancia a partir de la cámara, después de la medida máxima el algoritmo sigue funcionando, pero observa mínimos registros de falsos positivos y

falsos negativos es decir no detecta algunos peatones en la imagen.

4.2. Trabajos Futuros

Para trabajos futuros, se podría perfeccionar las técnicas de sustracción de fondo en las imágenes o videos para que no contemplen la aparición de sombras, para llegar a obtener valores más precisos. Ya que se ha trabajado sobre el seguimiento de muchas personas de manera simultánea, además se podría realizar integrar otras técnicas de reconocimiento de objetos basadas en Deep-Learning; a fin de optimizar la clasificación de los objetos.

No obstante, si se tiene en cuenta la línea de investigación del trabajo de fin de grado se podría utilizar para diferentes desarrollos de proyectos que se encuentren relacionados con la visión artificial como por ejemplo se podría se tendría que encontrar una forma de identificar de manera más rápida la ubicación de los peatones. Para ello se tendría que obtener las imágenes de mayor resolución de la cámara, para lograr disminuir el tiempo de espera y usar de forma más optima a tiempo real.

Otra posible solución sería desarrollar otro algoritmo que permita identificara mejor manera las personas en diferentes ángulos de la cámara u trabajar modificando el algoritmo de tal forma que pueda detectar todo tipo de personas en diferentes temporales climáticas ya sea en lluvia, nieve o vientos con polvo ya que actualmente el algoritmo trabaja de forma eficiente a días normales con luz ya que los peatones con los temporales climática tienden a tomar tonalidades muy oscuras, cercanas al negro, por lo que es difícil que consiga identificar correctamente al peatón.

Capítulo 5

Código del Programa

5.1. Código

```
    import numpy as np
import json
import cv2
import copy
import sys
stderr = sys.stderr
sys.stderr = open(os.devnull, 'w')
sys.stderr = stderr
import keras
import imgaug as ia
import xml.etree.ElementTree as ET
from imgaug import augmenters as iaa
from keras.utils import Sequence
from tkinter import *
import threading
from PIL import Image, ImageTk
import numpy as np
import time
import cv2
from math import sqrt
from keras.models import Model
import tensorflow as tf
from keras.layers import Reshape, Activation, Conv2D, Input, MaxPooling2D,
BatchNormalization, Flatten, Dense, Lambda
from keras.layers.advanced_ activations import LeakyReLU
from keras.layers.merge import concatenate
from keras.optimizers import SGD, Adam, RMSprop
from keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard
# Configuracion de la red **********
```

```
tamanio = 416
labels = [”person”]
mejores_ pesos = red_ person.h5”
anchors = [0.5811406319167913,
2.0145195200638706,
1.1052373546216343,
4.25112899786784,
2.572279531227269,
8.945986984631663,
1.5124197503880255,
7.535693455688639,
5.291787564439895,
10.043146284514377]
# Funciones adicionales

    def _ sigmoid(x):
        return 1. / (1. + np.exp(-x))
def _ softmax(x, axis=-1, t=-100.):
        x = x - np.max(x)
        if np.min(x) ¡t:
                x = x/np.min(x)*t
        e_ x = np.exp(x)
        return e_ x / e_ x.sum(axis, keepdims=True)
def _ interval_ overlap(interval_ a, interval_ b):
        x1, x2 = interval_ a
        x3, x4 = interval_ b
        if x3 ¡x1:
                if x4 ¡x1:
                        return 0
                else:
                        return min(x2,x4) - x1
        else:
                if x2 ¡x3:
                        return 0
                else:
                        return min(x2,x4) - x3
def compute_ overlap(a, b):
        area = (b[:, 2] - b[:, 0]) * (b[:, 3] - b[:, 1])
        iw = np.minimum(np.expand_ dims(a[:, 2], axis=1), b[:, 2]) - np.maximum(np.expand_ dims(a[:,
0], 1), b[:, 0])
        ih = np.minimum(np.expand_ dims(a[:, 3], axis=1), b[:, 3]) - np.maximum(np.expand_ dims(a[:,
1], 1), b[:, 1])
        iw = np.maximum(iw, 0)
        ih = np.maximum(ih, 0)
        ua = np.expand_ dims((a[:, 2] - a[:, 0]) * (a[:, 3] - a[:, 1]), axis=1) + area - iw * ih
```

```
        ua = np.maximum(ua, np.finfo(float).eps)
        intersection = iw * ih
        return intersection / ua

    def compute_ap(recall, precision):
        # correct AP calculation
        # first append sentinel values at the end
        mrec = np.concatenate(([0.], recall, [1.]))
        mpre = np.concatenate(([0.], precision, [0.]))
        # compute the precision envelope
        for i in range(mpre.size - 1, 0, -1):
            mpre[i - 1] = np.maximum(mpre[i - 1], mpre[i])
        # to calculate area under PR curve, look for points
        # where X axis (recall) changes value
        i = np.where(mrec[1:] != mrec[:-1])[0]
        # and sum ( Delta recall) * prec
        ap = np.sum((mrec[i + 1] - mrec[i]) * mpre[i + 1])
        return ap

    def decode_netout(netout, anchors, nb_class, obj_threshold=0.3, nms_threshold=0.3): grid_h, grid_
w, nb_box = netout.shape[:3]
        boxes = []
        # decode the output by the network
        netout[..., 4] = _sigmoid(netout[..., 4])
        netout[..., 5:] = netout[..., 4][..., np.newaxis] * _softmax(netout[..., 5:])
        netout[..., 5:] *= netout[..., 5:] ¿obj_threshold
        for row in range(grid_h):
            for col in range(grid_w):
                for b in range(nb_box):
                    # from 4th element onwards are confidence and class classes
                    classes = netout[row,col,b,5:]
                    if np.sum(classes) ¿0:
                        # first 4 elements are x, y, w, and h
                        x, y, w, h = netout[row,col,b,:4]
                        x = (col + _sigmoid(x)) / grid_w # center position, unit: image width
                        y = (row + _sigmoid(y)) / grid_h # center position, unit: image height
                        w = anchors[2 * b + 0] * np.exp(w) / grid_w # unit: image width
                        h = anchors[2 * b + 1] * np.exp(h) / grid_h # unit: image height
                        confidence = netout[row,col,b,4]
                        box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, confidence, classes)
                        boxes.append(box)
        # suppress non-maximal boxes
        for c in range(nb_class):
            sorted_indices = list(reversed(np.argsort([box.classes[c] for box in boxes])))
            for i in range(len(sorted_indices)):
```

```
                index_ i = sorted_ indices[i]
                if boxes[index_ i].classes[c] == 0:
                        continue
                else:
                        for j in range(i+1, len(sorted_ indices)):
                                index_ j = sorted_ indices[j]
                                if bbox_ iou(boxes[index_ i], boxes[index_ j]) ¿= nms_ threshold:
                                        boxes[index_ j].classes[c] = 0
    # remove the boxes which are less likely than a obj_ threshold
    boxes = [box for box in boxes if box.get_ score() ¿obj_ threshold]
    return boxes

def bbox_ iou(box1, box2):
    intersect_ w = _ interval_ overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
    intersect_ h = _ interval_ overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
    intersect = intersect_ w * intersect_ h
    w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
    w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin
    union = w1*h1 + w2*h2 - intersect
   return float(intersect) / union

class BoundBox:          def __ init__ (self, xmin, ymin, xmax, ymax, c = None, classes = None):
            self.xmin = xmin
            self.ymin = ymin
            self.xmax = xmax
            self.ymax = ymax
            self.c = c
            self.classes = classes
            self.label = -1
            self.score = -1
    def get_ label(self):
            if self.label == -1:
                    self.label = np.argmax(self.classes)
            return self.label
    def get_ score(self):
            if self.score == -1:
                    self.score = self.classes[self.get_ label()]
            return self.score

class BatchGenerator(Sequence):
    def _ _ init_ _ (self, images,
                config,
                shuffle=True,
                jitter=True,
                norm=None):
```

```
        self.generator = None
        self.images = images
        self.config = config
        self.shuffle = shuffle
        self.jitter = jitter
        self.norm = norm
        self.anchors = [BoundBox(0, 0, config['ANCHORS'][2*i], config['ANCHORS'][2*i+1])
for i in range(int(len(config['ANCHORS'])//2))]
        sometimes = lambda aug: iaa.Sometimes(0.5, aug)
        self.aug_pipe = iaa.Sequential(
            [
                sometimes(iaa.Affine()),
                iaa.SomeOf((0, 5),
                    [
                        iaa.OneOf([
                            iaa.GaussianBlur((0, 3.0)), # blur images with a sigma be-
tween 0 and 3.0
                            iaa.AverageBlur(k=(2, 7)), # blur image using local means
with kernel sizes between 2 and 7
                            iaa.MedianBlur(k=(3, 11)), # blur image using local me-
dians with kernel sizes between 2 and 7
                        ]),
                        iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)), # sharpen ima-
ges
                        iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_ chan-
nel=0.5), # add gaussian noise to images
                        iaa.OneOf([
                            iaa.Dropout((0.01, 0.1), per_ channel=0.5), # randomly re-
move up to 10% of the pixels
                        ]),
                        iaa.Add((-10, 10), per_ channel=0.5), # change brightness of ima-
ges (by -10 to 10 of original value)
                        iaa.Multiply((0.5, 1.5), per_ channel=0.5), # change brightness
of images (50-150% of original value)
                        iaa.ContrastNormalization((0.5, 2.0), per_ channel=0.5), # im-
prove or worsen the contrast
                    ],
                    random_ order=True
                )
            ],
            random_ order=True
        )
        if shuffle: np.random.shuffle(self.images)
    def __ len__ (self):
        return int(np.ceil(float(len(self.images))/self.config['BATCH_ SIZE']))
```

```
        def num_ classes(self):
                return len(self.config['LABELS'])
        def size(self):
                return len(self.images)
        def load_ annotation(self, i):
                annots = [ ]
               for obj in self.images[i]['object']:
                        annot = [obj['xmin'], obj['ymin'], obj['xmax'], obj['ymax'], self.config ['LABELS'].index(obj
['name'])]
                        annots += [annot]
                if len(annots) == 0: annots = [[ ]]
                return np.array(annots)
        def load_ image(self, i):
                return cv2.imread(self.images[i]['filename'])
        def __ getitem__ (self, idx):
                l bound = idx*self.config['BATCH_ SIZE']
                r_ bound = (idx+1)*self.config['BATCH_ SIZE']
                if r_ bound ¿len(self.images):
                        r_ bound = len(self.images)
                        l_ bound = r_ bound - self.config['BATCH_ SIZE']
                instance_ count = 0
                x_ batch = np.zeros((r_ bound - l_ bound, self.config ['IMAGE_ H'], self.config ['IMAGE_
W'], 3)) # input images
                b_ batch = np.zeros((r_ bound - l_ bound, 1, 1, 1, self.config ['TRUE_ BOX_ BUFFER'],
4)) # list of self.config ['TRUE_ self.config ['BOX']_ BUFFER'] GT boxes
                y_ batch = np.zeros((r_ bound - l_ bound, self.config ['GRID_ H'], self.config ['GRID_ W'],
self.config ['BOX'], 4+1+len(self.config ['LABELS']))) # desired network output
                for train_ instance in self.images[l_ bound:r_ bound]:
                        # augment input image and fix object's position and size
                        img, all_ objs = self.aug_ image(train_ instance, jitter=self.jitter)
                        # construct output from object's x, y, w, h
                        true_ box_ index = 0
                        for obj in all_ objs:
                                if obj['xmax'] ¿obj['xmin'] and obj['ymax'] ¿obj['ymin'] and obj['name'] in
self.config['LABELS']:
                                        center_ x = .5*(obj['xmin'] + obj['xmax'])
                                        center_ x = center_ x / (float(self.config ['IMAGE_ W']) / self.config
['GRID_ W'])
                                        center_ y = .5*(obj['ymin'] + obj['ymax'])
                                        center_ y = center_ y / (float(self.config['IMAGE_ H']) / self.co- nfig['GRID_
H'])
                                         grid_ x = int(np.floor(center_ x))
                                         grid_ y = int(np.floor(center_ y))
                                        if grid_ x ¡self.config['GRID_ W'] and grid_ y ¡self.config['GRI-
D_ H']:
```

```
                                        obj_ indx = self.config['LABELS'].index(obj['name'])
                                        center_ w = (obj['xmax'] - obj['xmin']) / (float(self.config['I-
MAGE_ W']) / self.config['GRID_ W']) # unit: grid cell
                                        center_ h = (obj ['ymax'] - obj ['ymin']) / (float(self.config ['IMA-
GE_ H']) / self.config ['GRID_ H']) # unit: grid cell
                                        box = [center_ x, center_ y, center_ w, center_ h]
                                        # find the anchor that best predicts this box
                                        best_ anchor = -1
                                        max_ iou = -1
                                        shifted_ box = BoundBox(0,
                                                                0,
                                                                center_ w,
                                                                center_ h)
                                        for i in range(len(self.anchors)):
                                                anchor = self.anchors[i]
                                                iou = bbox_ iou(shifted_ box, anchor)
                                                if max_ iou ¡iou:
                                                        best_ anchor = i
                                                        max_ iou = iou
                                        # assign ground truth x, y, w, h, confidence and class probs to y_
batch
                                        y_ batch[instance_ count, grid_ y, grid_ x, best_ anchor, 0:4] = box
                                        y_ batch[instance_ count, grid_ y, grid_ x, best_ anchor, 4 ] = 1.
                                        y_ batch[instance_ count, grid_ y, grid_ x, best_ anchor, 5+obj_
indx] = 1
                                        # assign the true box to b_ batch
                                        b_ batch[instance_ count, 0, 0, 0, true_ box_ index] = box
                                        true_ box_ index += 1
                                        true_ box_ index = true_ box_ index % self.config['TRUE_ BOX_
BUFFER']
                        # assign input image to x_ batch
                        if self.norm != None:
                                x_ batch[instance_ count] = self.norm(img)
                        else:
                                # plot image and bounding boxes for sanity check
                                for obj in all_ objs:
                                        if obj['xmax'] ¿obj['xmin'] and obj['ymax'] ¿obj['ymin']:
                                                cv2.rectangle(img [:,:,::-1], (obj ['xmin'], obj ['ymin']), (obj ['xmax'],
obj ['ymax']), (255,0,0), 3)
                                                cv2.putText(img[:,:,::-1], obj['name'],
                                                        (obj['xmin']+2, obj['ymin']+12),
                                                        0, 1.2e-3 * img.shape[0],
                                                        (0,255,0), 2)
                                x_ batch[instance_ count] = img
                        # increase instance counter in current batch
```

```
                    instance_ count += 1
                # print(' new batch created', idx)
                return [x_ batch, b_ batch], y_ batch
        def on_ epoch_ end(self):
                if self.shuffle: np.random.shuffle(self.images)
        def aug_ image(self, train_ instance, jitter):
                image_ name = train_ instance['filename']
                image = cv2.imread(image_ name)
                if image is None: print('Cannot find ', image_ name)
                h, w, c = image.shape
                all_ objs = copy.deepcopy(train_ instance['object'])
                if jitter:
                        ### scale the image
                        scale = np.random.uniform() / 10. + 1.
                        image = cv2.resize(image, (0,0), fx = scale, fy = scale)
                        ### translate the image
                        max_ offx = (scale-1.) * w
                        max_ offy = (scale-1.) * h
                        offx = int(np.random.uniform() * max_ offx)
                        offy = int(np.random.uniform() * max_ offy)
                        image = image[offy : (offy + h), offx : (offx + w)]
                ### flip the image
                flip = np.random.binomial(1, .5)
               if flip ¿0.5: image = cv2.flip(image, 1)
               image = self.aug_ pipe.augment_ image(image)
                # resize the image to standard size
                image = cv2.resize(image, (self.config ['IMAGE_ H'], self.config ['IMAGE_ W']))
                image = image[:,:,::-1]
                # fix object's position and size
                for obj in all_ objs:
                        for attr in ['xmin', 'xmax']:
                                if jitter: obj[attr] = int(obj[attr] * scale - offx)
                                obj[attr] = int(obj[attr] * float(self.config ['IMAGE_ W']) / w)
                                obj[attr] = max(min(obj [attr], self.config ['IMAGE_ W']), 0)
                        for attr in ['ymin', 'ymax']:
                                if jitter: obj[attr] = int(obj[attr] * scale - offy)
                                obj[attr] = int(obj [attr] * float(self.config ['IMAGE_ H']) / h
)                                 obj[attr] = max(min(obj [attr], self.config ['IMAGE_ H']), 0)
                        if jitter and flip ¿0.5:
                                xmin = obj['xmin']
                                obj['xmin'] = self.config ['IMAGE_ W'] - obj['xmax']
                                obj['xmax'] = self.config ['IMAGE_ W'] - xmin
                return image, all_ objs

    def draw_ boxes(image, boxes, labels):
```

```
        image_ h, image_ w, _ = image.shape
        for box in boxes:
                xmin = int(box.xmin*image_ w)
                ymin = int(box.ymin*image_ h)
                xmax = int(box.xmax*image_ w)
                ymax = int(box.ymax*image_ h)
                cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (0,255,0), 3)
        return image
def distancia(x1,y1,x0,y0):
        return sqrt((x1-x0)**2 + (y1-y0)**2)

    def draw_ lines(image, boxes, Maximo, Minimo, escala): # Argumentos: Image = imagen a dibujar
lineas; boxes = los rectangulos que reconocio la red,
       image_ h, image_ w, _ = image.shape # Maximo = distancia máxima que puede tener la linea
                                           # Minimo = distancia minima que puede tener la linea
                                           # Escala = Factor de conversion de pixeles a metros
        puntosx = [] # Array que contiene todos los centros de los rectangulos en X
        puntosy = [] # Array que contiene todos los centros de los rectangulos en Y
        AllBoxesNumber = 0 # Variable usada para contar todos los rectangulos que hay
        font = cv2.FONT_ HERSHEY_ SIMPLEX
        for box in boxes:
                xmin = int(box.xmin*image_ w) # Obtiene el punto mínimo del rectangulo en X
                ymin = int(box.ymin*image_ h) # Obtiene el punto mínimo del rectangulo en Y
                xmax = int(box.xmax*image_ w) # Obtiene el punto máximo del rectangulo en X
                ymax = int(box.ymax*image_ h) # Obtiene el punto máximo del rectangulo en Y
                puntosx.append(xmin + int(abs((float(xmax) - float(xmin)) / 2))) # Obtiene el centro del
rectangulo en X
                puntosy.append(ymin + int(abs((float(ymax) - float(ymin)) / 2))) # Obtiene el centro del
rectangulo en Y
                AllBoxesNumber = AllBoxesNumber + 1 # Cuenta cuantos rectangulos hay
        cuenta = 0 # Variable usada para iterar (para la combinatoria empleada para obtener las líneas)
        # Código de iteración que emplea combinatoria para trazar la trayectoria de todas las líneas posi-
bles entre cuadros
        for box in boxes:
                for j in range(cuenta + 1, AllBoxesNumber):
                        # Almacena la distancia entre dos centros de rectangulos
                        TemporalDistanciaPixel = distancia(puntosx[cuenta], puntosy[cuenta], puntosx[j],
puntosy[j]) # Distancia entre dos puntos en pixeles
                        # Discriminador de líneas. Solo traza las líneas cuya distancia cumpla la condición
de rango: Minimo ¡Distancia ¡Máximo
                        if (Minimo ¡TemporalDistanciaPixel) and (TemporalDistanciaPixel ¡Maximo):
                                metros = distancia(puntosx[cuenta], puntosy[cuenta], puntosx[j], puntosy[j])
# Almacena la distancia actual (pixeles) de la línea a trazar
                                cv2.line(image, (puntosx[cuenta], puntosy[cuenta]), (puntosx[j], puntosy[j]),
(0, 0, 255), 2) # Traza la línea de un centro a otro
```

```
                        # Fragmento de codigo cuya función es ubicar un texto en la mitad de la línea
trazada ******
                        PX = int(abs(float(puntosx[cuenta]) - float(puntosx[j])) / 2) # Obtiene la dis-
tancia media en X de la línea
                        PY = int(abs(float(puntosy[cuenta]) - float(puntosy[j])) / 2) # Obtiene la dis-
tancia media en Y de la línea
                        # Código que identifica el lugar en la línea donde poner el texto
                        if puntosx[cuenta] ¿= puntosx[j]:
                                PX = PX + puntosx[j]
                        else:
                                PX = PX + puntosx[cuenta]
                        if puntosy[cuenta] ¿= puntosy[j]:
                                PY = PY + puntosy[j]
                        else:
                                PY = PY + puntosy[cuenta]

                            # Escribe el texto en metros de acuerdo al factor de conversión en la posi-
ción calculada *****
                        cv2.putText(image, str(round(escala * metros, 1)) + ”m”, (PX, PY), font, 0.5,
(255, 255, 255),1)
                cuenta = cuenta + 1
        return image

    FULL_ YOLO_ BACKEND_ PATH = ”full_ yolo_ backend.h5”# should be hosted on a server
class BaseFeatureExtractor(object):
        ”docstring for ClassName”
        # to be defined in each subclass
        def __ init__(self, input_ size):
                raise NotImplementedError(.error message”)
        def normalize(self, image):
                raise NotImplementedError(.error message”)
        def get_ output_ shape(self):
                return self.feature_ extractor.get_ output_ shape_ at(-1)[1:3]
       def extract(self, input_ image):                    return self.feature_ extractor(input_ image)

    class FullYoloFeature(BaseFeatureExtractor):
        ”docstring for ClassName”
       def __ init__ (self, input_ size):
                input_ image = Input(shape=(input_ size, input_ size, 3))
               def space_ to _ depth_ x2(x):
                        return tf.space_ to_ depth(x, block_ size=2)

                    # Layer 1
                x = Conv2D (32, (3,3), strides = (1,1), padding = ’same’, name= ’conv_ 1’, use_ bias =
False) (input_ image)
```

```
x = BatchNormalization(name='norm_ 1')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_ size=(2, 2))(x)

    # Layer 2
x = Conv2D (64, (3,3), strides=(1,1), padding='same', name='conv_ 2', use_ bias = False)(x)
x = BatchNormalization(name='norm_ 2')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_ size=(2, 2))(x)

    # Layer 3
x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_ 3', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 3')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 4
x = Conv2D (64, (1,1), strides= (1,1), padding='same', name= 'conv_ 4', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 4')(x)
x = LeakyReLU(alpha=0.1)(x)
# Layer 5
x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_ 5', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 5')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_ size=(2, 2))(x)

    # Layer 6
x = Conv2D(256, (3,3), strides=(1,1), padding='same', name='conv_ 6', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 6')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 7
x = Conv2D(128, (1,1), strides=(1,1), padding='same', name='conv_ 7', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 7')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 8
x = Conv2D(256, (3,3), strides=(1,1), padding='same', name='conv_ 8', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 8')(x)
x = LeakyReLU(alpha=0.1)(x)
x = MaxPooling2D(pool_ size=(2, 2))(x)

    # Layer 9
x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_ 9', use_ bias=False)(x)
x = BatchNormalization(name='norm_ 9')(x)
```

```
x = LeakyReLU(alpha=0.1)(x)

    # Layer 10
x = Conv2D(256, (1,1), strides=(1,1), padding='same', name='conv_10', use_bias=False)(x)
x = BatchNormalization(name='norm_10')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 11
x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_11', use_bias=False)(x)
x = BatchNormalization(name='norm_11')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 12
x = Conv2D(256, (1,1), strides=(1,1), padding='same', name='conv_12', use_bias=False)(x)
x = BatchNormalization(name='norm_12')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 13
x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_13', use_bias=False)(x)
x = BatchNormalization(name='norm_13')(x)
x = LeakyReLU(alpha=0.1)(x)

    skip_connection = x
x = MaxPooling2D(pool_size=(2, 2))(x)

    # Layer 14
 x = Conv2D (1024, (3,3), strides= (1,1), padding = 'same', name = 'conv_14', use_
bias=False)(x)
x = BatchNormalization(name='norm_14')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 15
 x = Conv2D (512, (1,1), strides= (1,1), padding = 'same', name='conv_15', use_bias =
False)(x)
x = BatchNormalization(name='norm_15')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 16
x = Conv2D (1024, (3,3), strides = (1,1), padding = 'same', name = 'conv_16', use_bias =
False)(x)
x = BatchNormalization(name='norm_16')(x)
x = LeakyReLU(alpha=0.1)(x)

    # Layer 17
x = Conv2D (512, (1,1), strides = (1,1), padding = 'same', name = 'conv_17', use_bias =
```

```
False)(x)
                x = BatchNormalization(name='norm_ 17')(x)
                x = LeakyReLU(alpha=0.1)(x)

                    # Layer 18
                x = Conv2D (1024, (3,3), strides = (1,1), padding = 'same', name = 'conv_ 18', use_
bias=False)(x)
                x = BatchNormalization(name='norm_ 18')(x)
                x = LeakyReLU(alpha=0.1)(x)

                    # Layer 19
                x = Conv2D (1024, (3,3), strides = (1,1), padding='same', name = 'conv_ 19', use_ bias =
False)(x)
                x = BatchNormalization(name='norm_ 19')(x)
                x = LeakyReLU(alpha=0.1)(x)

                    # Layer 20
                x = Conv2D (1024, (3,3), strides = (1,1), padding = 'same', name = 'conv_ 20', use_
bias=False)(x)
                x = BatchNormalization(name='norm_ 20')(x)
                x = LeakyReLU(alpha=0.1)(x)

                    # Layer 21
                skip_ connection = Conv2D (64, (1,1), strides = (1,1), padding = 'same', name='conv_ 21',
use_ bias = False)(skip_ connection)
                skip_ connection = BatchNormalization(name='norm_ 21')(skip_ connection)
                skip_ connection = LeakyReLU(alpha=0.1)(skip_ connection)
                skip_ connection = Lambda(space_ to_ depth_ x2)(skip_ connection)
                x = concatenate([skip_ connection, x])

                    # Layer 22
                x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_ 22', use_ bias=False)(x)
                x = BatchNormalization(name='norm_ 22')(x)
                x = LeakyReLU(alpha=0.1)(x)

                    self.feature_ extractor = Model(input_ image, x)
                self.feature_ extractor.load_ weights(FULL_ YOLO_ BACKEND_ PATH)

            def normalize(self, image):
                return image / 255.

    class YOLO(object):
        def __ init__ (self,
                            input_ size,
                            labels,
```

```
                    max_ box_ per_ image,
                    anchors):
    self.input_ size = input_ size
    self.labels = list(labels)
    self.nb_ class = len(self.labels)
    self.nb_ box = len(anchors)//2
    self.class_ wt = np.ones(self.nb_ class, dtype='float32')
    self.anchors = anchors
    self.max_ box_ per_ image = max_ box_ per_ image
    # Make the model
    # make the feature extractor layers
    input_ image = Input(shape=(self.input_ size, self.input_ size, 3))
    self.true_ boxes = Input(shape=(1, 1, 1, max_ box_ per_ image , 4))
    self.feature_ extractor = FullYoloFeature(self.input_ size)
    print(self.feature_ extractor.get_ output_ shape())
    self.grid_ h, self.grid_ w = self.feature_ extractor.get_ output_ shape()
    features = self.feature_ extractor.extract(input_ image)
    # make the object detection layer
    output = Conv2D(self.nb_ box * (4 + 1 + self.nb_ class),
                    (1,1), strides=(1,1),
                    padding='same',
                    name='DetectionLayer',
                    kernel_ initializer='lecun_ normal')(features)
    output = Reshape((self.grid_ h, self.grid_ w, self.nb_ box, 4 + 1 + self.nb_ class))(output)
    output = Lambda(lambda args: args[0])([output, self.true_ boxes])
    self.model = Model([input_ image, self.true_ boxes], output)
    # initialize the weights of the detection layer
    layer = self.model.layers[-4]
    weights = layer.get_ weights()
    new_ kernel = np.random.normal(size=weights[0].shape)/(self.grid_ h*self.grid_ w)
    new_ bias = np.random.normal(size=weights[1].shape)/(self.grid_ h*self.grid_ w)
    layer.set_ weights([new_ kernel, new_ bias])
    # print a summary of the whole model
    self.model.summary()

def load_ weights(self, weight_ path):
    self.model.load_ weights(weight_ path)

def predict(self, image):
    image_ h, image_ w, _ = image.shape
    image = cv2.resize(image, (self.input_ size, self.input_ size))
    image = self.feature_ extractor.normalize(image)
    input_ image = image[:,:,::-1]
    input_ image = np.expand_ dims(input_ image, 0)
    dummy_ array = np.zeros((1,1,1,1,self.max_ box_ per_ image,4))
```

```
                netout = self.model.predict([input_ image, dummy_ array])[0]
                boxes = decode_ netout(netout, self.anchors, self.nb_ class)
                return boxes

    video_ path = 'vtest.avi'
video_ out = video_ path[:-4] + '_ detected' + video_ path[-4:]
video_ reader = cv2.VideoCapture(video_ path)

    # Iniciar red neuronal **********
mi_ yolo = YOLO(input_ size = tamanio,
                labels = labels,
                 max_ box_ per_ image = 5,
                anchors = anchors)
mi_ yolo.load_ weights(mejores_ pesos)

    # Configuracion de la cámara ***************************
frameWidth= 480 # CAMERA RESOLUTION 480 pixeles ancho
frameHeight = 720 # CAMERA RESOLUTION 720 pixeles alto
brightness = 100
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# Set setup of the video camera
cap = cv2.VideoCapture(0) # Camara computadora
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10, brightness)

    from tqdm import *
font = cv2.FONT_ HERSHEY_ SIMPLEX
Fotogramas = 0
while True:
                # READ IMAGE
                Fotogramas = Fotogramas + 1
                _ , image = cap.read()
                boxes = mi_ yolo.predict(image)
               image = draw_ boxes(image, boxes, labels)
                image = draw_ lines(image, boxes, 280, 80, 0.0028)
                cv2.putText(image, 'Detectados: ' + str(len(boxes)) , (10, 40), font, 0.75, (0, 255, 0))
                cv2.putText(image, 'Fotograma # ' + str(Fotogramas) , (10, 80), font, 0.75, (0, 255, 0))
                cv2.imshow("Detecta personas", image)
                if cv2.waitKey(1) and 0xFF == ord('c'):
                        break
cap.release()
cv2.destroyAllWindows()
```

Bibliografía

[1] GOMEZ ORTEGA JOSE, *PLAN DE ACTUACIÓN. CORONAVIRUS (COVID-19)*, Murcia, CONSEJERÍA DE AGUA, AGRICULTURA, GANADERÍA, PESCA Y MEDIO AMBIENTE, 2020

[2] M. Palacios, E. Santos, M.A.Velázquez, M. León, *COVID-19,una emergencia de salud pública mundial*, Veracruz, México, Revista Clínica Española, 2020

[3] Pacheco Mayra, *El Ministerio de Salud aprobó un reglamento para que las universidades en Ecuador investiguen sobre covid-19*, Quito, El Comercio, 15 de abril de 2020

[4] Silva Vanessa, *11 000 trabajadores del Ecuador fueron desvinculados por fuerza mayor; los empleadores deberán justificar motivos*, Quito, El Comercio, 2020

[5] Organización Panamericana de la Salud, *CONSIDERACIONES SOBRE MEDIDAS DE DISTANCIAMIENTO SOCIAL Y MEDIDAS RELACIONADAS CON LOS VIAJES EN EL CONTEXTO DE LA RESPUESTA A LA PANDEMIA DE COVID-19*, Estados Unidos, Organización Panamericana de la Salud, 2020

[6] Vargas Nestor, Andrés González, *Sistema de visión por computadora para la medición de distancia e inclinación de obstáculos para robots móviles*, Cali, Universidad Autónoma de Occidente, 2005

[7] Vasconcelos Antoni B., Chan Zhang-Sheng, John Liang Nuno, *Privacy Preserving Crowd Monitoring:Counting People without People Models or Tracking*, San Diego, 2008

[8] Iván García, Víctor Caranqui, *LA VISIÓN ARTIFICIAL Y LOS CAMPOS DE APLICACIÓN*, Tulcán, Universidad Politécnica Estatal del Carchi, 2015

[9] L. Kingo, «*pactomundial.org,*», GreenBiz, 13 abril 2020, [En línea]. Available: https://www.pactomundial.org/2020/04/como-sera-el-mundo-despues-de-la-covid-19/, [Último acceso: 2020 abril 2020]

[10] Antoni Chan, Nuno Vasconcelos, *Modeling, Clustering, and SegmentingVideo with Mixtures of Dynamic Textures*, IEEE Trans. on Pattern Analysis and Machine Intelligence, pp. vol. 30(5), pp. 909–26, 2008

[11] Poppe, R. *A survey on vision-based human action recognition*, Image and vision computing, 28(6): 976 – 990, 2010

[12] Moeslund T., Hilton A., Krüger V. *A survey of advances in vision-based human motion capture and analysis*, Computer vision and image understanding, 104(2 – 3): 90 – 126, 2006

[13] L. Enrique, Giovani Gómez, *Visión Computacional*, Instituto Nacional de Astrofísica, Óptica y Electrónica Puebla, México, 2015

[14] Vélez J. F., Moreno A. B., Sánchez Á., Sánchez J. L., *2º edición. Visión por computador*

[15] Yang M.-H., Kriegman D., Ahuja N., *faces in images: a survey,"Transactions on Pattern Analysis and Machine Intelligence"*, vol. 24, no. 1, pp. 3458, Jan 2002

[16] Rojas Jessica,*CONTEO DE PERSONAS MEDIANTE VIDEOCÁMARAS UNIVERSIDAD AUTÓNOMA METROPOLITANA*, México , 2013

[17] Amini Z., V. Abootalebi, M.T. Sadegui , *A Comparative Study of Feature Extraction Methods in P300 Detection*. Proceedings of the 17th Iranian Conference of Biomedical Engineering (ICBME2010), Teheran, 2010

[18] Wu J., Zhang X., *A PCA classifier and its application in vehicle detection*, Preceedings of the IEEE International Joint Conference on Neural Networks, 600-604, 2001

[19] Garrido C., *Extracción de Características para algoritmos de Aprendizaje Automático*, Universidad Politécnica de Madrid, España, 2017

[20] Campos M., *Evaluación de procesos Gaussianos en la estimación de parámetros biofísicos*, Universidad de Valencia, España, 2013

[21] Shawe-Taylor J., Cristianini N., *Kernel Methods for Pattern Analysis. Cambridge*, U.K.: Cambridge Univ. Press, 2004

[22] B. Scholkopf, A. Smola, *Learning With Kernels—Support Vector Machines, Regu-larization, Optimization and Beyond*. Cambridge, MA: MIT Press, 2002

[23] Aponte P., Armijos A., *Desarrollo de un sistema de visión artificial para la detección de accidente y/o congestionamiento vehicular*, Universidad Nacional de Loja, Ecuador, 2015

[24] Jiménez M., *Desarrollo de un sistema de visión artificial para la detección de aglomeración de personas en un semáforo*, UNIVERSIDAD NACIONAL DE LOJA, Ecuador, 2015

[25] García M., *El color como recurso expresivo:Análisis de las series de televisión Mad Men y Breaking Bad*, UNIVERSIDAD COMPLUTENSE DE MADRID, España, 2016

[26] Niebles E., Muñoz V.; Pacheco J., *Aplicación de las ecuaciones de lagrange para un sistema rotatorio con desalineamiento angular*, Universidad Autónoma del Caribe, Colombia, 2011

[27] Basogain X., *REDES NEURONALES ARTIFICIALES Y SUS APLICACIONES*, Escuela Superior de Ingeniería de Bilbao, España.

[28] López P., *Desarrollo de sistemas de tiempo real basados en componentes utilizando modelos de comportamiento reactivos*, Universidad de Cantabria, España, 2010.

[29] Open Source Initiative, *The BSD License*, www.opensource.org/licenses/bsd-license.php, 2003.

[30] NumPy community, *NumPy User Guide*, 2021

[31] Bernd Klein, Bodenseo, *Diseño de Denise Mitchinson adaptado para python-course.eu*, 2011-2020

[32] Manuel Zaforas, *TensorFlow, o cómo será el futuro de la Inteligencia Artificial según Google.*https://www.paradigmadigital.com/dev/tensorflow-sera-futuro-la-inteligencia-artificial-segun-google/, España

[33] Keras community, *Keras Sencillo. Flexible, Poderoso*,https://keras.io,

[34] Recursos Python, emphInstalar PIL / Pillow y aplicar efectos visuales, 2014