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:

"Estimación de Número de Personas en Imágenes de Multitudes usando Aprendizaje de Máquina"

Autor: Deysi Steffany Gualoto López

Director: Carlos Xavier Rosero Chandi

Ibarra-Ecuador

Agosto 2021

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	1003862248
APELLIDOS Y NOMBRES	GUALOTO LÓPEZ DEYSI STEFFANY
DIRECCIÓN	IBARRA, AV. LOS GALEANOS Y ANALÍA BERNAL
EMAIL	dsgualotol@utn.edu.ec
TELÉFONO	0985775943
DATOS DE LA OBRA	
TÍTULO	"ESTIMACIÓN DE NÚMERO DE PERSONAS EN IMÁGENES DE MULTITUDES USANDO APRENDIZAJE DE MÁQUINA"
AUTOR	DEYSI STEFFANY GUALOTO LÓPEZ
FECHA	12/08/2021
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, 12 de agosto de 2021

Deysi Steffany Gualoto López

C.I.: 1003862248

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado "ESTIMACIÓN DE NÚMERO DE PERSONAS EN IMÁGENES DE MULTITUDES USANDO APRENDIZAJE DE MÁQUINA", presentado por la egresada DEYSI STEFFANY GUALOTO LÓPEZ, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 12 de agosto de 2021

Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

Carlos Xavier Rosero
DIRECTOR DE TESIS

Agradecimiento

Antes que nada, doy gracias a Dios por bendecirme y guiarme en cada paso de este camino y permitirme cumplir una de las metas más anheladas.

Gracias a mis padres y hermana, por la confianza y el apoyo que me han brindado durante todo este tiempo, por todo el esfuerzo que han realizado para ayudarme económicamente en mis estudios, quiero que sepan que siempre sentí su mano protectora y su voz de aliento en los momentos más difíciles. Los quiero mucho.

Quiero expresar un sincero agradecimiento a mi director Carlos Xavier Rosero, principal colaborador durante todo este proceso, quien con su dirección y cooperación permitió el desarrollo de este trabajo y la culminación del mismo.

Agradezco mucho a todos mis profesores que compartieron sus conocimientos y experiencias durante toda mi carrera, aportando a mi formación profesional y personal.

A mis compañeros y a quienes más que compañeros se convirtieron en verdaderos amigos, gracias por hacer de esta etapa de vida en una de las mejores experiencias.

Deysi Steffany Gualoto López

Dedicatoria

Este trabajo de titulación se lo dedicó de manera especial a mis padres y hermana, por todo su amor, trabajo y sacrificio en todos estos años, siendo dignos de admiración y ejemplos de esfuerzo. Gracias a ustedes he logrado llegar hasta aquí y convertirme en una profesional. Este logró no solo es mío, también es de ustedes.

Con amor,
Deysi Steffany Gualoto López

Resumen

Actualmente, existe una gran demanda de sistemas de monitoreo de multitudes para su análisis y seguimiento. El conteo de personas en aglomeraciones ha adquirido una gran relevancia en el campo de la seguridad pública. Por lo tanto, el objetivo del presente trabajo de grado es desarrollar un sistema que permita detectar y estimar el número de personas presentes en imágenes de multitudes, usando aprendizaje de máquina y visión por computador. El núcleo de este enfoque se basa principalmente en el uso de un descriptor de características Histogramas de Gradientes Orientados (HOG) y un modelo de clasificación supervisado Máquinas de Vectores de Soporte (SVM). Estos algoritmos son implementados en una plataforma de software libre, utilizando el lenguaje de programación Python y bibliotecas libres. La evaluación del rendimiento del modelo de clasificación se realiza usando las métricas orientadas a estos modelos, por otro lado, para el algoritmo de detección y conteo se realiza pruebas experimentales en imágenes de un conjunto de datos de libre acceso, orientado al análisis de multitudes de diferentes densidades. Estas imágenes son de distintas dimensiones y presentan diferentes perspectivas de cámara, variaciones de iluminación y diversos fondos. Por último, mediante las pruebas realizadas se puede deducir que el sistema presenta un mejor desempeño en imágenes de multitudes con un ángulo de cámara normal y una densidad uniforme.

Abstract

Today, there is a high demand for crowd monitoring systems for analysis and tracking. Crowd counting has become very important in the field of public safety. Therefore, the objective of the present degree work is to develop a system that allows detecting and estimating the number of people present in crowd images, using machine learning and computer vision. The core of this approach is mainly based on the use of a feature descriptor Histograms of Oriented Gradients (HOG) and a supervised classification model Support Vector Machines (SVM). These algorithms are implemented on a free software platform, using the Python programming language and free libraries. The performance test of the classification model is performed using metrics oriented to these models, on the other hand, for the detection and counting algorithm, experimental tests are performed on images of a freely available dataset, oriented to the analysis of crowds of different densities. These images are of different dimensions and present different camera perspectives, illumination variations and diverse backgrounds. Finally, from the tests performed it is deduced that the system performs better on images of crowds with a normal camera angle and uniform density.

Índice general

Índice de figuras

Índice de tablas

Introducción

Problema

Eventos culturales, socioeducativos, deportivos, empresariales, entre otros, se han venido desarrollando a lo largo de los años [1]. Sin embargo, en la actualidad debido a la pandemia del COVID-19 [2], este tipo de actividades son consideradas de alto riesgo, ya que muchos sitios carecen de sistemas de control de aforo de personas [3].

En Ecuador, según la Cámara de Comercio de Quito, una de las medidas que se están llevando a cabo para prevenir la propagación del virus son los controles de ingeniería que reducen la exposición a los riesgos. Entre ellos se encuentra evitar las aglomeraciones de personas [3].

En la actualidad, la implementación de métodos de visión por computador orientados al monitoreo de aglomeraciones, han logrado potencializar el uso de videocámaras permitiendo generar aplicaciones tales como video vigilancia, control de accesos, análisis de flujos de movimiento de personas o el conteo de personas. Este tipo de aplicaciones han adquirido cada vez más importancia en los últimos años, dadas las necesidades de prevención y detección de situaciones peligrosas, como la que se vive por el COVID-19 [4].

Sin embargo, las técnicas de visión por computador tradicionales tienen sus limitaciones en situaciones de hacinamiento, ya que es difícil segmentar y rastrear a cada individuo con alta precisión, a causa de las oclusiones severas [5]. Pero gracias al avance de las tecnologías de Industria 4.0, es posible implementar técnicas de aprendizaje de máquina, que cuentan con una capacidad de procesamiento bastante alta y pueden lograr una precisión sorprendente, a menudo alcanzando o superando las capacidades de visión humanas, con el fin de resolver problemas de ingeniería y logística [6].

Por lo cual, se pretende desarrollar un sistema que permita detectar y contabilizar el número de personas presentes en grandes aglomeraciones, usando aprendizaje de máquina y visión por computador, ya que el trabajo conjunto de estos métodos, en valor de precisión y confiabilidad, logra un mejor rendimiento.

Objetivos

Objetivo General

Desarrollar una aplicación de estimación de número de personas, mediante el uso de técnicas de aprendizaje de máquina y visión por computador, bajo una plataforma de software libre.

Objetivos Específicos

- Definir los algoritmos para llevar a cabo el conteo de personas en imágenes de multitudes.
- Desarrollar el sistema de estimación de número de personas.
- Validar el desempeño de la propuesta a través de pruebas en un escenario controlado.

Justificación

El presente trabajo se enfocará en el desarrollo de una aplicación para estimar el número de personas en imágenes de multitudes haciendo uso de algoritmos OpenSource y recopilación de toda la información necesaria que permita desarrollar un sistema confiable. El núcleo de este enfoque se basará en modelos generados a través del entrenamiento de SVM mediante las características obtenidas por HOG.

Hoy en día, el uso de sistemas de visión por computador y aprendizaje de máquina para el análisis de aglomeraciones ha adquirido una gran relevancia en múltiples ámbitos como la seguridad, administración, bioseguridad, entre otras, por la importancia que genera poder disponer de la información en cuanto a número de visitantes, cantidad estimada de personas en un recinto, estadística de asistencia, etc. Además, el implementar este tipo de sistemas atrae varias ventajas como reducción de costos, ahorro de tiempo en tareas complejas o incluso predicción de eventos [7].

La importancia de este trabajo se centra en el reconocimiento y conteo de personas en lugares con gran afluencia, lo cual en función de la pandemia servirá para identificar áreas que requieran de un mayor cuidado, y por ende establecer protocolos de bioseguridad más rigurosos, con el fin de precautelar la salud y la seguridad de la ciudadanía frente al riesgo biológico [8].

Alcance

El presente trabajo de grado desarrollará un sistema de estimación de número de personas en imágenes de multitudes, usando el modelo de aprendizaje de máquina SVM (Máquinas de Soporte de Vectores) para la clasificación automática de personas

a partir de las características obtenidas por HOG (Histograma de Gradientes Orientados), y posteriormente realizar el conteo de personas. Este sistema no incluye la implementación de los algoritmos en un prototipo físico, la validación del desempeño de la propuesta se efectuará mediante simulaciones, bajo una plataforma de software libre y código abierto.

Capítulo 1

Revisión Literaria

1.1. Generalidades de los Sistemas de Conteo

La visión por computador es un campo que pertenece a la inteligencia artificial, el cual estudia diversas técnicas para adquirir, procesar, identificar y clasificar patrones que se pueden encontrar en imágenes digitales, con el fin de producir información numérica o simbólica que pueda ser tratada por un computador [9]. Así, un sistema de conteo de objetos basado en visión artificial es un conjunto de estrategias y algoritmos que tienen la capacidad de procesar y contar objetos de cierta clase en una imagen o en un conjunto de imágenes [10].

1.1.1. Tipos de Sistemas de Conteo

Línea de Interés (Line of Interes, LOI)

Es una estrategia de conteo basada en la detección y seguimiento de personas hasta el momento en que cruzan una línea de detección definida por el usuario, en un tiempo determinado, tal como se puede observar en la Fig.1.1. En conclusión, el conteo de persona por LOI se centra en el número de personas que pasan por una escena trazada [10], [11].

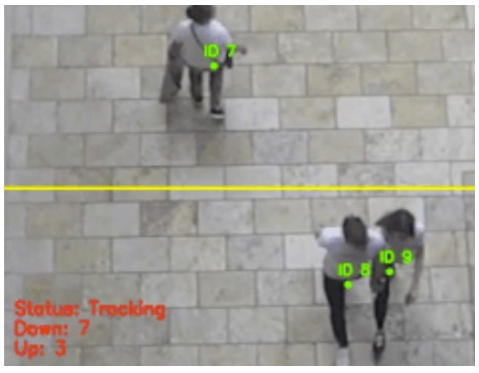


Figura 1.1: Representación de la línea de interés en una escena [12].

Región de Interés (Region of Interes, ROI)

Esta es una estrategia de estimación de número de personas estáticas o en movimiento, que se encuentran en alguna región en una determinada estancia de tiempo. El propósito de ROI es juzgar si una región es multitud o no, reduciendo así el número de cálculos y minimizando la carga sobre el sistema, este método se enfoca principalmente en la detección [10], [11], como se muestra en la Fig.1.2.

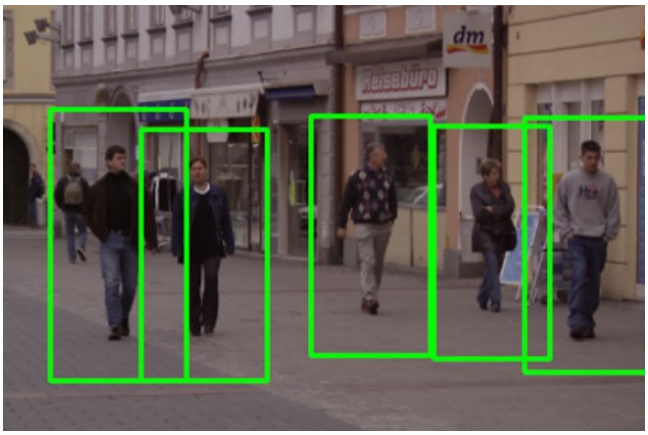


Figura 1.2: Representación de la región de interés en una escena [13].

En la Tabla 1.1, se indica las diferentes ventajas y desventajas de las estrategias mencionadas para el conteo de personas.

Tabla 1.1: Ventajas y desventajas de sistemas de conteo de personas.

Categoría	ROI	LOI
Ventajas	▪ Conteo de personas estáticas y en movimiento. ▪ Eficiente para el conteo de multitudes. ▪ Comportamiento aceptable en oclusiones de personas.	▪ Conteo de personas en movimiento. ▪ Fácil implementación. ▪ Tiempo corto de procesamiento.
Desventajas	▪ Mayor tiempo de procesamiento. ▪ Requiere del entrenamiento de clasificadores, para que reconozcan una persona en una escena.	▪ No realiza conteo de personas estáticas. ▪ Mal comportamiento en multitudes. ▪ No requiere de clasificadores.

1.2. Generación de Candidatos

La extracción de candidatos consiste en generar ventanas de tamaño fijo, que realicen desplazamientos dentro de una imagen y como resultado de este proceso se espera que se tengan N ventanas con posibles candidatos, para que puedan pasar a ser evaluados y clasificados. La ubicación y el tamaño de dichas ventanas debe ser conocidas por el usuario para localizar el candidato de manera directa dentro de la escena [10].

Esta etapa es fundamental en un sistema de conteo por visión artificial para localizar y detectar un objeto, como por ejemplo, a personas dentro de una escena [14]. Existen algunas técnicas para ejecutar esta etapa siendo las más utilizadas las siguientes:

1.2.1. Ventana Deslizante (Sliding Window)

Esta técnica parte de una ventana canónica que se va deslizando de forma que realice un barrido por toda la imagen, teniendo así varias ventanas y a partir de estas se obtiene el descriptor de cada una de ellas para enviar al clasificador, donde se define a las detecciones válidas de la imagen.

El desplazamiento de la ventana canónica requiere de definir ciertos parámetros, los cuales son, el paso en el eje X y el paso en el eje Y, tal como se observa en la Fig.1.3, estos van a servir para generar un patrón de desplazamiento [15].

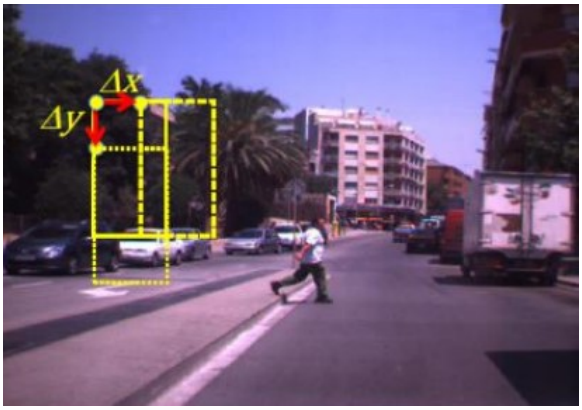


Figura 1.3: Representación del desplazamiento de una ventana deslizante [16].

1.2.2. Pirámide con Ventana Desilzante (Pyramidal Sliding Window)

Esta técnica se basa en el mismo principio de la ventana deslizante, pero lleva en su nombre lo de pirámide debido a que una vez realizado el barrido por toda la imagen completa, esta se reduce a un escala elegida, para así volver a realizar un barrido.

Este proceso se lo realiza hasta que la imagen sea inferior al tamaño de la ventana, en cada uno de los barridos realizados la ventana será capaz de predecir si en esa región de la imagen hay o no una persona [14].

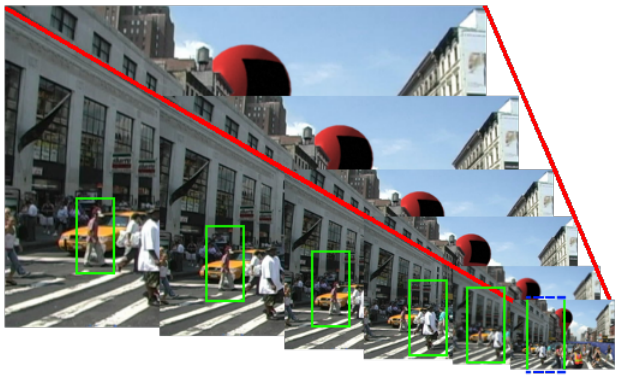


Figura 1.4: Proceso que realiza una pirámide con ventana deslizante [17].

1.3. Extracción de Características

Una característica es un atributo de un objeto de interés, que nos permite describir y reconocer de otros objetos. Por lo tanto, la extracción de características es un paso crucial para el análisis de imágenes y especialmente para detección y reconocimiento de objetos, en este caso personas.

La etapa de extracción de características en realidad se refiere al cálculo de descriptores, el descriptor consiste en transformar un conjunto de datos de entrada, en este caso una imagen en una representación simplificada que se conoce como vector de características, es decir, características agrupadas de un dato [18]. Algunos de los descriptores más utilizados se listan a continuación:

- **Transformación de Características Inavariantes a la Escala (Scale Invariant Feature Transform, SIFT):** Algoritmo presentado por Lowe [19], es un descriptor de borde basado en gradientes ampliamente utilizados para fines de reconocimiento de objetos. Las características de este algoritmo son invariantes ante cambios de rotación, escala, corrimiento y parcialmente invariantes a cambios de iluminación y perspectiva 3D.

- **Patrón Binario Local (Local Binary Pattern, LBP):** Este descriptor se propuso originalmente para el análisis de texturas [2], ha demostrado ser un enfoque simple pero poderoso para describir estructuras locales de las imágenes. La función LBP presenta grandes ventajas, como la invariancia de grises y la invariancia de rotación. En los últimos años, ha despertado un interés creciente en muchas áreas del procesamiento de imágenes y la visión por computadora, mostrando su efectividad en una serie de aplicaciones como la detección de objetos y en particular en el análisis de imágenes faciales [20].

- **Wavelet de Haar:** Estos descriptores son invariantes a cambios de color y textura, fueron propuestos por Viola y Jones [21] sobre las bases planteadas por Papageorgiou *et al.* [22], este método consiste en aplicar varios clasificadores a una ventana de detección, los cuales se usan en serie y cada uno es cada vez más complejo que el anterior, esto logra que este descriptor defina de manera robusta clases de objetos complejos.

- **Histogramas de Gradientes Orientados (Histogram of Oriented Gradients, HOG):** Este descriptor de forma fue propuesto por Dalal y Triggs [23], se basa en la orientación de gradientes de una imagen utilizados para la detección de objetos que puedan existir en aquella imagen. Este descriptor destaca en comparación a los descriptores anteriores debido a su robustez frente a diferentes condiciones de iluminación, pequeños cambios en el contorno de la imagen, diferentes fondos y escalas, y ha presentado buenas prestaciones según resultados previos de otros autores [23], [24]. Por tal motivo, este trabajo de grado se basará en este descriptor el cuál se explica a continuación:

1.3.1. Descriptor Basado en Histogramas de Gradientes Orientados (HOG)

El Histograma de Gradiente Orientado es un descriptor de características ampliamente utilizado, el cual, está basado en que la apariencia y la forma local de un objeto puede caracterizarse bastante bien mediante la distribución de gradientes locales de intensidad o direcciones de los bordes en la imagen.

Este algoritmo consiste en dividir la ventana de la imagen en pequeñas regiones espaciales denominadas *celdas*, acumulando en cada celda un histograma local de direcciones de gradiente u orientaciones de borde sobre los píxeles de cada celda. Para una mejor invariancia a la iluminación, sombras, etc., se recomienda normalizar el contraste de las respuestas locales antes de usarlas. Esto se puede lograr acumulando los histogramas locales en regiones espaciales más grandes denominados *bloques*. Con los bloques de los descriptores normalizados se refiere como el descriptor HOG [23]. Este proceso se puede visualizar en la Fig.1.5.

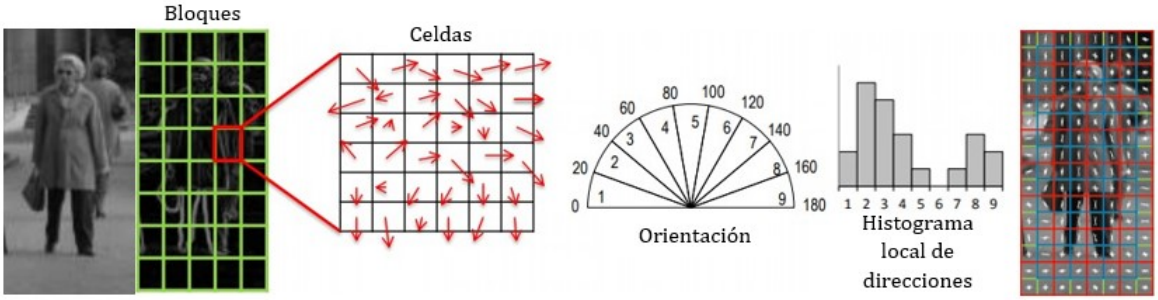


Figura 1.5: Proceso que se lleva a cabo para obtener el descriptor HOG [16].

1.4. Clasificación Supervisada

Un clasificador es el enfoque sistemático para construir modelos a partir de un conjunto de datos de entrada. Para un entrenamiento particular, cada muestra está representada por el mismo conjunto de características. Si las muestras están etiquetadas por clase, el método de aprendizaje se llama supervisado [18].

En esta etapa el sistema es capaz de decidir si los candidatos generados corresponden o no a la clase de interés, basados en el vector de características obtenido del descriptor

[10]. Existen muchas técnicas de clasificación supervisada diferentes, pero algunas de las máa utilizadas se describen a continuación:

- **Regresión Logística:** Es un algoritmo que se utiliza para problemas de clasificación binaria, de análisis predictivo y se basa en el concepto de probabilidad. Los eventos de clasificación pueden ser representados por la variable Y, la cual puede tomar dos valores 0 y 1. El valor 1 habitualmente es representado como un evento exitoso, mientras que el valor 0 como un evento fallido.
- **Adaptive Boosting:** Las técnicas del boosting obtienen clasificadores muy precisos a partir de la combinación de varios clasificadores débiles. Estos clasificadores base se distribuyen en grupos por etapas y se enlazan formando una cascada, actuando cada uno sobre las predicciones del anterior, dando lugar al clasificador final que presente resultados fuertes [25].
- **Algoritmo KNN (K-Nearest Neighbors):** Conocido también como el algoritmo del vecino cercano, sirve para clasificar información y obtener predicciones. Sirve par estimar la probabilidad de que un elemento X pertenezca a una clase C.
- **Máquinas de Vectores de Soporte (Suppor Vector Machine, SVM):** Algoritmos utilizados para la clasificación y regresión desarrollados por Vladimir Vapnik [26]. Dado un conjunto de ejemplos de entrenamiento se puede etiquetar las clases y entrenar la SVM para construir un nuevo modelo. Este es uno de los algoritmos más populares y ampliamente utilizados. Dalal y Triggs [23] evidencian en su trabajo que HOG y SVM presentan buenos resultados. En este trabajo de grado se utilizará este último clasificador supervisado.

1.4.1. Máquinas de Vectores de Soporte (SVM)

Las Máquinas de Vectores de Soporte son un algoritmo de aprendizaje supervisado que se basa en la idea de encontrar un hiperplano de margen máximo, que separe el conjunto de características en dos clases. Para tener el menor error en la clasificación es necesario que la distancia entre el hiperplano de solución y los vectores de soporte (SVC) sea lo mayor posible [14]. Este hiperplano del que se habla se puede apreciar en la Fig.1.6.

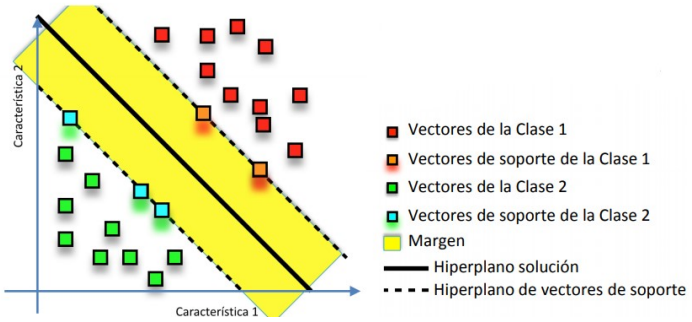


Figura 1.6: Representación gráfica del hiperplano de la máquina de vectores de soporte [16].

Descripción Matemática

Este hiperplano se puede describir mediante $w^T * x_i + b = 0$ donde w es normal al hiperplano y $\frac{b}{||w||}$ es la distancia perpendicular desde el hiperplano al origen, entonces la clasificación de dos clases se define como:

$$w^T * x_i + b > 1 \quad Para \quad y_i = 1 \tag{1.1}$$

$$w^T * x_i + b < 1 \quad Para \quad y_i = -1 \tag{1.2}$$

Estas dos ecuaciones pueden describirse mediante una sola ecuación de la siguiente forma:

$$y_i(x_i * w + b) - 1 > 0 \tag{1.3}$$

Estas ecuaciones son aplicables para conjuntos de puntos que son linealmente separables, entonces se dibuja una línea o hiperplano que pase por los puntos más cercanos al hiperplano de separación, como se ilustra en la la Fig.1.7.

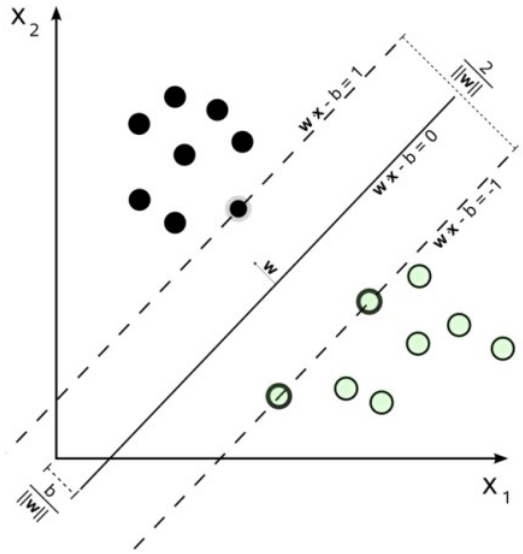
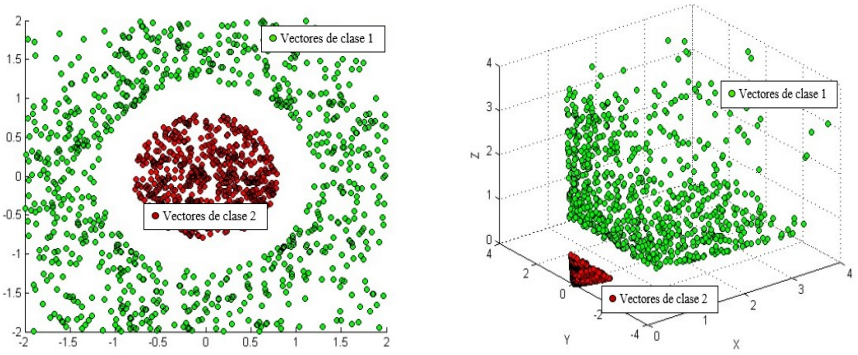


Figura 1.7: Representación de la idea matemática del hiperplano de la SVM [27].

Funciones del Kernel

La SVM cuenta con las funciones Kernel, estas se utilizan cuando las clases no son linealmente separables. Estas funciones transforman los datos a linealmente separables en un espacio de dimensión superior [16], tal como se muestra en la Fig.1.8.



(a) Conjunto de datos no lineales

(b) Mapeo 3D del conjunto de datos

Figura 1.8: Mapeo de características a un espacio de dimensión superior [27].

Adicional al kernel lineal, existe más tipos de kernel que se puede aplicar dependiendo de la necesidad del sistema, estos son:

- **Núcleo polinomial:** En este núcleo no es tan frecuente, puesto a que no es tan eficiente computacionalmente en comparación con los otros núcleos.

$$K(X_1, X_2) = (a + X_1^T X_2)^b \tag{1.4}$$

  Donde $K(X1, X2)$ representa el límite de decisión polinomial que separará sus datos, mientras que $X1$ y $X2$ representan sus datos.

- **Núcleo de función de base radial (RBF):** También conocida como Núcleo gaussiano, es uno de los más potentes y más usado en SVM.

$$K(X_1, X_2) = e^{(-\gamma||X_1 - X_2||^2)} \tag{1.5}$$

  En esta ecuación, γ especifica cuánto tiene un único punto de entrenamiento en los otros puntos de datos que lo rodean y $||X1 - X2||$ es el producto escalar entre sus funciones.

Parámetros de Ajuste

- **Regularización:** Conocido como el parámetro C, consiste en suavizar el margen que acepte un cierto nivel de clasificación errónea.
  - Si C es muy grande, el hiperplano tendrá un margen más pequeño y se puede obtener problemas de sobreentrenamiento (overfitting).
  - Si C es muy pequeño, entonces el margen será grande y se podría tener riesgo que el modelo no se ajuste al entrenamiento (underfitting).
- **Gamma:** Este parámetro define hasta dónde llega la influencia de un solo ejemplo de entrenamiento. Esto significa que una gamma alta considerará solo los puntos cercanos al hiperplano y una gamma baja considerará los puntos a mayor distancia.

1.5. Post - Procesamiento

Esta etapa consiste en la depuración de la información final de los clasificadores, es decir identifica que ventanas corresponden a un mismo candidato, para así reducirlas a una sola ventana conocida como bounding box [10], tal como se muestra en la Fig.1.9.

1.5.1. Supresión No Máxima (NMS)

Este algoritmo es el encargado de suprimir los cuadros delimitadores repetidos de una escena que detectan a una misma persona; con el fin de evitar que un individuo sea contado más de una vez.

(a) Detecciones sin post-procesamiento

(b) Aplicando NMS

Figura 1.9: Detecciones de peatones realizadas en una imagen del conjunto de datos de INRIA [28].

Proceso del algoritmo NMS:

- Crea un conjunto P con las detecciones realizadas por ventana.
- Crea un conjunto G vacío.
- Elige un primer elemento del conjunto P y se observa si se solapa mínimo un 50 % con alguno de los elementos del conjunto G. Si no existe un solapamiento suficiente con ninguno de los elementos del conjunto G, este pasa a ser elemento del conjunto G y se elimina del conjunto P. Si se solapa lo suficiente con algún elemento del conjunto G, el elemento simplemente se elimina del conjunto P. Este proceso se realiza hasta que el conjunto P este vacío.
- Finalmente, el conjunto G contendrá las detecciones finales.

1.6. Propuesta

Expuesto todo lo anterior en este capítulo, se propone desarrollar un sistema de detección y conteo de personas, el cual se dividirá en diferentes etapas como, la generación de candidatos usando pirámide con ventana deslizante, la extracción de características con HOG, la clasificación de los candidatos con SVM y el post-procesamiento aplicando NMS.

Capítulo 2

Metodología

En este apartado se presenta todos los pasos a seguir para la implementación de los algoritmos, que se han seleccionado en el proceso de la revisión literaria de este trabajo de grado.

2.1. Diseño del Sistema

Este sistema consta principalmente de 3 etapas: extracción de características con HOG, entrenamiento del clasificador SVM y el algoritmo de detección y conteo, tal como se puede apreciar en el diagrama de bloques de la Fig.2.1. Esto se desarrollará usando el lenguaje de programación de código abierto Python [29] y bibliotecas libres como la de visión por computador OpenCV [30] y la de aprendizaje de máquina Scikit-learn [31].

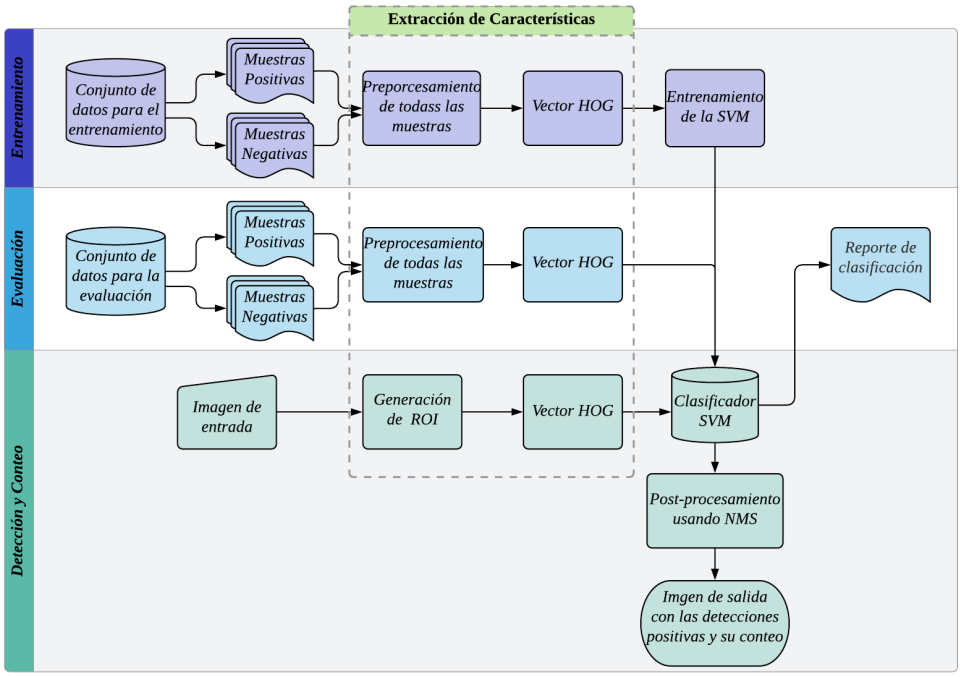


Figura 2.1: Diagrama simplificado del sistema, donde se presentan las etapas de extracción de características, entrenamiento, evaluación y detección y conteo.

La primera etapa corresponde al algoritmo que realiza la extracción de características HOG, tanto al conjunto de imágenes destinadas al entrenamiento y evaluación del clasificador, como a la imagen de entrada del sistema.

En la segunda etapa se utiliza el vector de características obtenido en la etapa anterior para entrenar el clasificador SVM, el cual es adaptado a los requerimientos del sistema. Para conocer la precisión de nuestro modelo entrenado es necesario de un conjunto de imágenes destinadas para la evaluación.

En la tercera etapa se refiere al desarrollo del algoritmo de detección, el cual se enfoca en localizar la cabeza y hombros de las personas en imágenes de multitudes. A continuación, en las siguientes secciones se describe el conjunto de datos utilizado, configuración de parámetros de los algoritmos, la justificación de las decisiones en el diseño, etc.

2.2. Conjunto de Datos

El conjunto de datos hace referencia a un grupo de imágenes de una determinada clase que pueden ser utilizados para realizar etapas de entrenamiento, en este trabajo de grado específicamente se va a emplear un conjunto de imágenes de personas y fondos.

En internet se pueden encontrar múltiples conjunto de datos que son destinados para fines educativos o de investigación. El conjunto de imágenes que se utilizará en este sistema se ha obtenido una gran parte de la base de datos INRIA [32] y otra parte de diferentes instituciones [33], [34] que han cedido sus bases de datos acerca de la detección de personas, estas están formados por dos conjuntos el de muestras positivas que son las personas y el de muestras negativas que son los fondos.

En el conjunto de imágenes positivas encontramos personas en distintos entornos que aparecen en posiciones estáticas y dinámicas sea de frente, de espaldas o se encuentre mirando hacia el lado derecho o izquierdo, también se encuentran personas de diferentes edades, géneros y etnias. Todo esto para que el entrenamiento disponga de suficiente información, en la Fig.2.2 se puede observar algunas de estas imágenes.

Figura 2.2: Muestras positivas de la base de datos INRIA[32].

En el conjunto de imágenes de fondos, es decir, las muestras negativas se trata de entornos tanto urbanos como no urbanos, por ejemplo, calles, autos, árboles, edificios, entre otros. Esto debido para que el modelo entrenado pueda detectar personas en diversos lugares. En la Fig.2.3. se observa algunas imágenes del conjunto de datos de fondos.

Figura 2.3: Muestras de fondo urbano y no urbano de la base de datos INRIA [32].

El conjunto de datos proporcionado por INRIA cuenta con imágenes de tamaño de 64x128 píxeles, ya que es ideal para trabajar con el algoritmo de detección desarrollado en [23]. En este trabajo de grado se toma como referencia este algoritmo HOG y SVM, el cual tendrá ciertas modificaciones ya que será utilizado para detectar personas en multitudes. Por lo que se requiere realizar una adecuación a los conjuntos de datos que se van a emplear.

2.2.1. Adecuación del Conjunto de Datos

Debido a que se llevará a cabo el conteo de personas en multitudes con HOG y SVM, lo óptimo será realizar la detección de cabeza-hombros, siendo necesario una adecuación de los conjuntos de datos tal como se muestra en la Fig.2.4; ya que por las oclusiones severas que se presentan en un hacinamiento sería difícil reconocer a las personas con el algoritmo planteado en [23], ya que este se enfoca en detectar personas de cuerpo entero y sin mayores oclusiones.

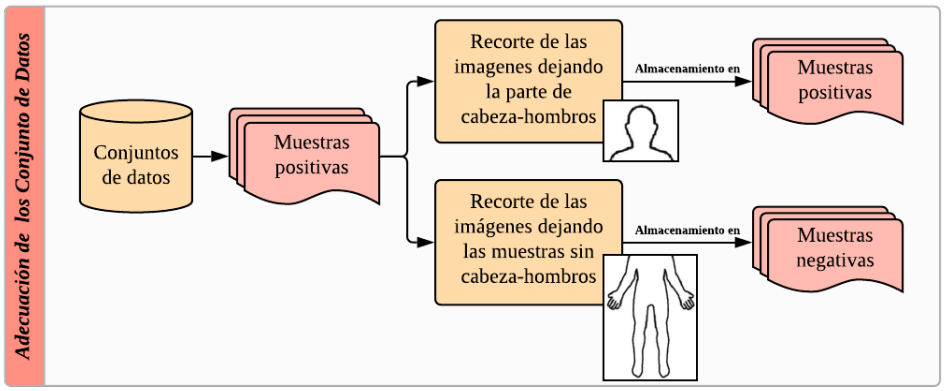


Figura 2.4: Adecuación de las imágenes de los conjuntos de datos que se emplearán en la aplicación.

Para preparar el conjunto de datos lo primero que se realiza es, recortar las imágenes dejando la parte de cabeza-hombros, siendo así estas las nuevas muestras positivas del conjunto de datos que se va a emplear, algunas de estas muestras se observa en la Fig.2.5.

Figura 2.5: Comparación entre algunas imágenes originales de la base de datos de INRIA con las imágenes que se utilizará en la aplicación.

Mientras tanto, en el conjunto de datos de muestras negativas se añadirá imágenes de personas sin la parte de la cabeza-hombros, algunas de estas imágenes se pueden observar en la Figura 2.6.

Figura 2.6: Nuevas muestras negativas que se usará en la aplicación.

Las muestras negativas deben ser mayor a las muestra positivas, por esta razón se obtiene un conjunto de datos conformado por, 3000 muestras positivas y 4260 muestras negativas. Por último, se distribuye el conjunto de datos en 80 % para entrenamiento y el 20 % para evaluación, teniendo lo siguiente:

- **Entrenamiento (Train):** 2500 muestras positivas y 3550 muestras negativas.
- **Evaluación (Test):** 500 muestras positivas y 710 muestras negativas.

2.3. Extracción de Características con HOG

Una vez que se tiene el conjunto de imágenes preparado, se procede a la extracción de características con los descriptores HOG de cada imagen, necesarios para entrenar posteriormente la SVM.

Debido a que las dimensiones de todas las muestras son diferentes, es necesario redimensionarlas, ya que uno de los requisitos de HOG es que las imágenes del conjunto

de datos deben tener una misma dimensión, es decir definir el tamaño de la ROI, los valores óptimos según Dalal y Triggs [23] es de 64x128 píxeles para detección de personas de cuerpo entero.

En este caso se busca valores diferentes, ya que la ROI se centrará en la detección de cabeza-hombros. La ROI definida es de 32x32 píxeles, valores que se basan en investigaciones previamente realizadas por otros autores [35], [20], [36].

Una vez realizado este preprocesamiento en las muestras positivas y negativas, se procede con el proceso de extracción del descriptor HOG de cada imagen de la base de datos. Estos descriptores son un vector que contiene un cierto número de componentes que definen las características de apariencia y forma de cada imagen. Estos vectores son almacenados para ser utilizados en las siguientes etapas del sistema. Todo este procedimiento se aplica tanto para las muestras positivas como negativas del conjunto de datos de entrenamiento y de evaluación, tal como se muestra en el diagrama de bloques de la Fig.2.7.

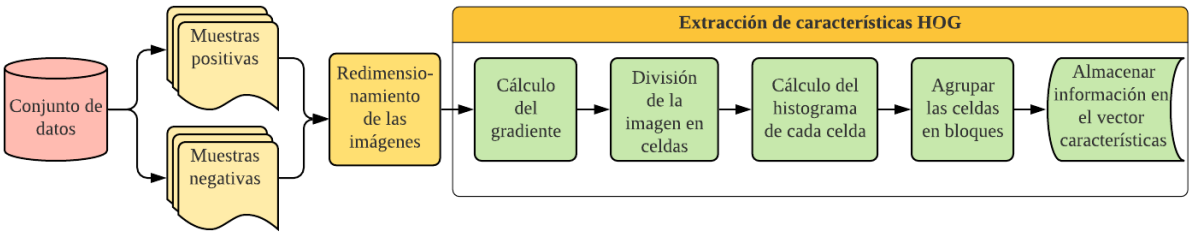


Figura 2.7: Diagrama simplifcado del proceso de la extracción de características HOG.

2.3.1. Parámetros de HOG

Como se ha mencionado, la clase HOG nace de la implementación ideado por Dalal y Triggs [23], en su artículo nos menciona algunos parámetros de diseño como: los tamaños de ventana, celdas y bloques. A continuación, se explica en detalle todos los parámetros que contiene la función *HOGDescriptor* que se encuentra en la librería OpenCV, indicando los valores que se han escogido para esta aplicación.

- **Tamaño de ventana = win_Size (32x32):** Este define el tamaño de la ventana, la cual se mide en píxeles y debe coincidir con el tamaño de la imagen de entrada para el entrenamiento, es por eso, que anteriormente se comentó que se realiza un redimensionamiento de cada imagen del conjunto de datos a 32x32 pixeles para que estas coincidan con el tamaño de la ventana del descriptor.

- **Tamaño de celda = cell_Size (8x8):** Es la división de la imagen de entrada en píxeles, a partir de estas celdas se obtiene el cálculo de los histogramas de orientación de gradientes. El tamaño que se utilizará es de 8x8 píxeles.

- **Tamaño de bloque = block_Size (16x16):** Este viene dado en píxeles e indica el tamaño del bloque utilizado en la fase de normalización de histogramas, este

debe alinearse con el tamaño de la celda, en este caso se tiene bloques de 2x2 celdas.

- **Desplazamiento del bloque = win_Stride (8x8):** Este indica el desplazamiento del bloque a lo largo de las dos dimensiones de la imagen y también define que tan largo va a ser el vector descriptor. Su tamaño debe ser múltiplo del tamaño de la celda. En este sistema se utilizará 8x8 píxeles.
- **Número de contenedores = nbins:** Indica como se va a dividir las orientaciones de los gradientes, el número por defecto de contenedores de Open CV es 9.
- **Parámetro de suavizado Gaussiano = win_sigma:** Parámetro utilizado en el procesado de la imagen que realiza el algoritmo antes de empezar el análisis y la extracción de descriptores. Su valor por defecto definido en la clase es -1.
- **Constante de normalización del bloque = threshold_L2hys:** Constante utilizada en el método de normalización, por defecto 0.2 y como única opción en OpenCV, la norma L2 con truncamiento.
- **Número máximo de ventanas de detección = nlevels:** Indica el número máximo de escalas que aplicará HOG a la ventana de detección. El valor por defecto y máximo es 64.

El descriptor HOG se implementa por medio del uso de las funciones integradas en la librería de OpenCV en Python. A continuación, en la Tabla 2.1, se muestra de forma resumida las funciones utilizadas.

Tabla 2.1: Funciones utilizadas en la implementación del descriptor HOG.

Librería	OpenCV	
Funciones para el Pre-procesamiento	*imread*	Lee las imágenes
	resize	Redimensiona las imágenes
Extracción de características HOG	*HOGDescriptor*	Crea un objeto tipo HOG
	hog.compute	Realiza la extracción de características

2.4. Entrenamiento del Clasificador SVM

Una vez obtenido el vector de características HOG de todas las imágenes del conjunto de entrenamiento de las dos clases, positivas y negativas, se procede con el entrenamiento del clasificador SVM utilizando el paquete Scikit-learn de Python.

La idea de la SVM es obtener un clasificador capaz de separar las clases con la mayor precisión posible, haciendo la diferenciación de 2 clases en un hiperplano de margen óptimo como se ha explicado en la subsección 1.4.1.

La SVM se entrenará de manera inicial con un kernel lineal, debido a que en las detecciones de personas lo utilizan por defecto, en este kernel se tiene un único parámetro que controla la regularización, C.

En la Tabla 2.2 se observa las especificaciones que se han utilizado inicialmente en el entrenamiento de la SVM.

Tabla 2.2: Especificaciones usadas en la etapa de entrenamiento.

Librería	Scikit-Learn	
Funciones para el Entrenamiento	*SVC*	Crea el clasificador tipo SVM.
	fit	Entrena el modelo tomando los datos de entrenamiento como argumentos.
Parámetros de Diseño	*kernel = linear*	Clasificación de vectores de soporte lineal.
	C = 1	Parámetro de regularización con su valor por defecto.

Para poder analizar el rendimiento del modelo entrenado se realiza una evaluación del mismo. Como se mencionó anteriormente se aparto un conjunto de imágenes para este procedimiento, en el cual se determina ciertas métricas de evaluación sobre el modelo, los resultados obtenidos se puede observar en la sección 3.1.1. Para realizar esta etapa se lleva a cabo el mismo proceso que se aprecia en la Fig.2.7

2.4.1. Exploración de Hiperparámetros para el Modelo SVM

Con el objetivo de aumentar el rendimiento del clasificador y reducir el número de falsos positivos y negativos que se obtiene en el proceso de evaluación, se realiza la búsqueda de los hiperparámetros y kernel que mejor se ajuste al sistema. Para esto se utiliza la función *Grid_search* del paquete de scikit-learn, la cual dada una serie de parámetros prueba las diferentes combinaciones de estos para evaluar cual es la mejor.

A continuación, se puede apreciar los parámetros que se probarán, en donde se implementa dos kernels adicionales al kernel lineal, que son el kernel radial (rbf) y el kernel polinomial (poly) con diferentes valores de C y gamma.

- C: [0.01, 0.1, 1, 10]
- Gamma: [0.01, 0.1, 1, 10]
- Kernel: [linear, rbf, poly]

Los resultados obtenidos de este proceso se pueden analizar en la sección 3.1.4.

2.5. Algoritmo de Detección

Para desarrollar este algoritmo de detección de cabeza-hombros de personas usando HOG y SVM, lo primero que hay que realizar es cargar el modelo entrenado anteriormente.

La estructura de este algoritmo sigue los siguientes pasos:

1. Se crea el objeto de la clase HOGDescriptor con los mismos parámetros de diseño que se utilizaron en el entrenamiento .

2. Se diseña la pirámide con una ventana deslizante, la cual ya se ha explicado en la sección 1.2.2, esta función ayuda a predecir las clases positivas sobre cualquier imagen, el tamaño de esta ventana debe coincidir con el tamaño de ventana de HOG, generando así las regiones de interés (ROIs) en la imagen analizada.

3. Definida ya el tamaño de la ventana deslizante, se escoge un paso de ventana, es decir el número de pixeles se va a desplazar, en este casa se escoge un paso X e Y de 8 píxeles, ya que es múltiplo del tamaño de ventana.

4. En cada desplazamiento se realiza la extracción de características y se predice si en la ventana existe o no una ROI, usando el clasificador entrenado.

5. Para tomar una ventana de detección como positiva, se toma en cuenta si supera el umbral de decisión en este caso de un 0.96, y posteriormente se guardan las coordenadas de la ventana en un vector.

6. Una vez realizado el desplazamiento de la ventana en toda la imagen, tanto en el eje X como en el eje Y, se realiza una reducción de la imagen a una escala elegida, y así volver a realizar el barrido.

7. Todos los pasos anteriores se repiten hasta que la imagen sea inferior al tamaño de la ventana, por motivo de que en una imagen puede aparecer una ROI con un tamaño muy superior al tamaño de la ventana deslizante y de esta manera poder detectarla.

8. Al finalizar el proceso de detección se obtendrá que varias ventanas están superpuestas en una ROI, para este problema se deberá realizar una etapa posterior que es la NMS explicada en la sección 1.5.1

2.6. Post-Procesamiento

En esta etapa se lleva a cabo la implementación del algoritmo de supresión no máxima y el conteo de personas detectadas.

2.6.1. Supresión No Máxima (NMS)

Este algoritmo se implementa usando la función non_max_supression del paquete imutils, que consiste en suprimir las ventanas repetidas en una ROI, eligiendo la ventana con mayor predicción como la principal, en la imagen las detecciones se representarán mediante cuadrados delimitadores (bounding box), de los cuales se guardarán sus coordenadas en arreglos, tomando cada arreglo como una ROI detectada.

A pesar de realizar este proceso de NMS, hay que tomar en cuenta que pueden existir falsos positivos o que dos cuadros estén detectando la misma persona.

2.6.2. Conteo de Personas

En esta segunda etapa del post procesamiento es simplemente realizar el conteo de las cabezas-hombros detectadas en la imagen, para esto se lleva a cabo el conteo de los arreglos obtenidos después de la NMS. Adicional a este paso, hay que realizar una comparación de los candidatos detectados con los candidatos reales de la imagen, con la finalidad de analizar el desempeño del sistema.

Capítulo 3

Pruebas y Resultados

En este capítulo se da a conocer los resultados de la evaluación del rendimiento del modelo entrenado inicialmente, así como los resultados con el ajuste de hiperparámetros. Además, se probó la eficiencia del sistema desarrollado mediante distintas imágenes de multitudes, realizando una comparación del número de personas detectadas por el sistema con el número real que se encuentran en una imagen.

3.1. Evaluación del Modelo Entrenado

3.1.1. Métricas de Evaluación

Para evaluar el rendimiento del modelo entrenado para la clasificación se basó en las siguientes métricas:

- **Precisión:** Esta es la calidad de la respuesta del clasificador. Es el número de verdaderos positivos sobre todas las detecciones obtenidas.

$$Precisión = \frac{VP}{VP + FP} \tag{3.1}$$

- **Sensibilidad:** Es la eficiencia del modelo entrenado en la clasificación de todos los elementos de la clase positiva. Cuantos verdaderos positivos realmente son detectados correctamente.

$$Sensibilidad = \frac{VP}{VP + FN} \tag{3.2}$$

- **Especificidad:** Es la eficiencia en la clasificación de los elementos de la clase negativa. Es la proporción de verdaderos negativos.

$$Especificidad = \frac{VN}{VN + FP} \tag{3.3}$$

- **Exactitud:** Es la proximidad entre el resultado obtenido con la clasificación exacta, es decir, el número de detecciones correctas de las dos clases, tanto verdaderos positivos como verdaderos negativos.

$$Exactitud = \frac{VP + VN}{VP + FN + FP + VN} \tag{3.4}$$

- **Puntaje F1:** Es cálculo de la media armónica de la precisión y la sensibilidad proporcionando así un valor más general del sistema, lo deseable de esta métrica es 1.

$$F1 = 2 * \frac{Precisión * Sensibilidad}{(Precisión + Sensibilidad)} \tag{3.5}$$

Siendo:

- **VP:** Verdadero Positivo, fue predicho como verdadero y es en realidad verdadero.
- **FP:** Falso Positivo, fue predicho como verdadero y es negativo.
- **VN:** Verdadero Negativos, predicho como negativo y es negativo en realidad
- **FN:** Falso Negativo, predicción de negativo y en realidad es un positivo.

3.1.2. Resultados Obtenidos

Para efectuar la evaluación del modelo se utilizó el conjunto de imágenes de prueba conformadas por 710 muestras negativas y 500 muestras positivas. Para obtener las métricas de evaluación sobre el modelo se utilizó la función *classification_report* del paquete scikit-learn. Los resultados obtenidos se pueden visualizar en la Tabla 3.1.

Tabla 3.1: Resultados obtenidos sobre el conjunto del test para el descriptor HOG utilizando el modelo entrenado SVM con kernel lineal.

Métricas de Evaluación	**Resultados**
Precisión	0.89
Sensibilidad	0.83
Especificidad	0.93
Exactitud	0.89
Puntaje F1	0.89

Además, para una visión más clara de cómo se encuentra el modelo, se utiliza la matriz de confusión usando la función *plot_confusion_matrix* del paquete de scikit-learn, la cual nos ayuda a observar de forma detallada los aciertos y errores que tiene nuestro modelo sobre cada clase.

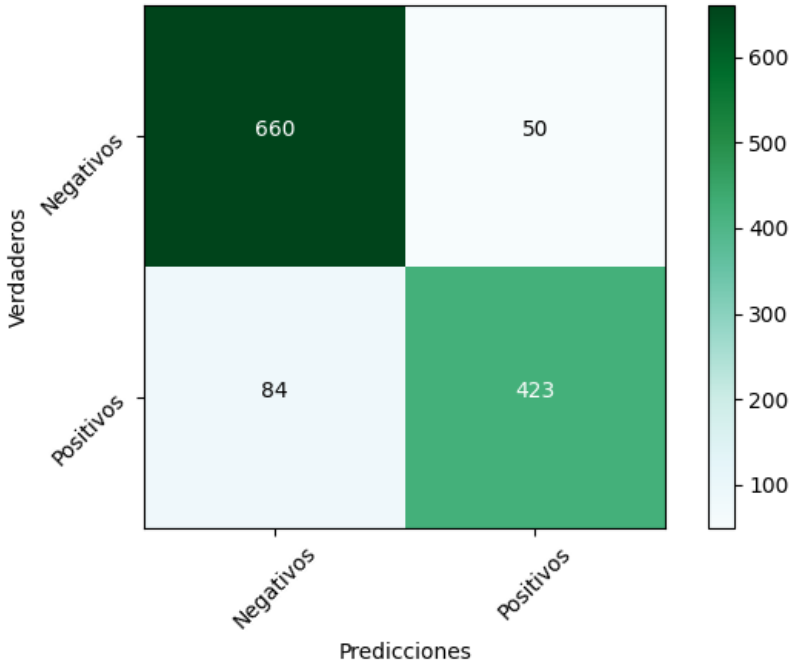


Figura 3.1: Matriz de confusión del modelo de clasificación SVM lineal.

En la Fig.3.1 se observa que 84 personas fueron clasificadas de manera incorrecta como fondo y 50 muestras negativas fueron clasificadas de forma errónea como personas, Esto quiere decir que se esta cometiendo más errores en la clasificación de la clase positiva, hay que tomar en cuenta que siempre habrá errores de clasificación debido al parámetro de regularización de la SVM.

3.1.3. Validación Cruzada (k-Fold Cross Validation)

Se realizó una mejora en el modelo implementando la validación cruzada con el objetivo de conseguir un mejor balance entre sesgo y varianza, ya que si se tiene un sesgo muy alto se obtiene un modelo que no se ajusta a los datos de entrenamiento y si se tiene una varianza muy alta provocará un sobreajuste en el modelo.

La validación cruzada consiste en unir los conjuntos de imágenes del entrenamiento y de la evaluación para después dividirlo en k subconjuntos, no existe una regla formal para elegir este valor k pero no debe ser demasiado pequeño o demasiado grande, es común elegir $k = 5$ o $k = 10$, basado en un estudio de Kohavi [37] donde se presenta evidencia empírica que estas elecciones producen equilibrio razonable entre sesgo y varianza.

En este caso fue dividido en 10 grupos, entonces el modelo se entrena sobre 9 grupos y habrá 1 de evaluación, luego se repite el proceso de forma iterativa hasta que

los 10 grupos hayan sido utilizadas como conjunto de evaluación. En la Fig.3.2 se puede observar un esquema de este proceso.

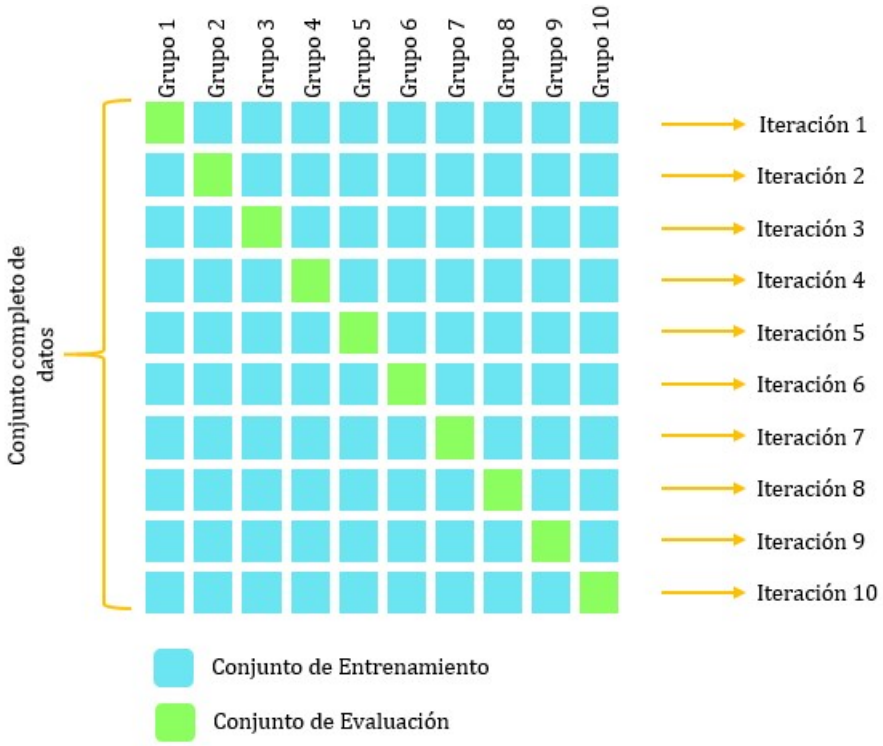


Figura 3.2: Proceso de la validación cruzada para un conjunto de imágenes divido en 10 grupos.

Finalmente, el resultado de la validación cruzada sobre este modelo es una media de los 10 resultados obtenidos, los cuales se pueden visualizar a continuación:

- **Exactitud Promedio/(Varianza)=** 0.8963 (+/- 0.0633)
- **Exactitud de cada grupo =** [0.8516 0.853 0.9162 0.9272 0.9258 0.8874 0.8613 0.9437 0.8845 0.912]

3.1.4. Resultados de la Exploración de Hiperparámetros para el Modelo SVM

Hasta el momento solo se ha analizado los resultados del modelo entrenado con el kernel lineal y los demás parámetros por defecto. Pero la SVM tiene diferentes hiperparámetros que pueden ser ajustados con el fin de mejorar el modelo.

Como se mencionó en la sección 2.4.1, se usó la función *Grid_search*, que realiza diferentes combinaciones de los parámetros a probar en el modelo y evaluar cual es la mejor combinación. Los parámetros que se probaron fueron los que se establecieron en esta misma sección. De la Tabla 3.2 se puede observar que la mejor combinación de hiperparámetros es la siguiente:

- **Mejores hiperparámetros:** $[C = 10, gamma = 1, kernel = rbf]$

Tabla 3.2: Resultados de la exploración de hiperparámetros para la SVM.

Parámetro C	**Parámetro Gamma**	**Kernel**	**Puntaje promedio**
10	1	rbf	0.934187
1	1	rbf	0.930890
10	0.1	rbf	0.928966
0.1	1	poly	0.928965
10	0.1	poly	0.928279
0.001	1	poly	0.928279
0.1	10	poly	0.928141
10	1	poly	0.928141
1	10	poly	0.928141
1	1	poly	0.928141
0.01	10	poly	0.928141
10	10	poly	0.928141
1	0.1	poly	0.909866
1	0.1	rbf	0.905607
10	0.01	rbf	0.892279
1	10	linear	0.888156
1	1	linear	0.888156
1	0.1	linear	0.888156
1	0.01	linear	0.888156
10	10	linear	0.886370
10	1	linear	0.886370
10	0.1	linear	0.886370
10	0.01	linear	0.886370
0.1	0.01	linear	0.885821
0.1	10	linear	0.885821
0.1	0.1	linear	0.885821
0.1	1	linear	0.885821
0.1	0.1	poly	0.876204
0.1	0.1	rbf	0.872082
1	0.01	rbf	0.869749
0.1	1	rbf	0.854495
0.001	0.01	linear	0.854495
0.001	10	linear	0.854495
0.001	0.1	linear	0.854495
0.001	1	linear	0.810664
0.1	0.01	rbf	0.803107
10	0.01	poly	0.803107
0.01	0.1	poly	0.795276
0.01	0.1	rbf	0.628064
10	10	rbf	0.627651
1	10	rbf	0.587112
0.01	1	rbf	0.586700
0.01	10	rbf	0.586700
0.1	0.01	poly	0.586700
0.01	0.01	poly	0.586700
0.01	0.01	rbf	0.586700
1	0.01	poly	0.586700
0.1	10	rbf	0.586700

Ya obtenido los mejores hiperparámetros se realizó un reentrenamiento del modelo y se realizó su respectiva evaluación, y al igual que en el primer modelo se realizó la validación cruzada en 10 grupos, consiguiendo los siguientes resultados:

Tabla 3.3: Resultados obtenidos sobre el conjunto de test para el descriptor HOG utilizando el modelo reentrenado SVM con kernel radial.

Métricas de Evaluación	**Resultados**
Precisión	0.94
Sensibilidad	0.90
Especificidad	0.97
Exactitud	0.94
Puntaje F1	0.94

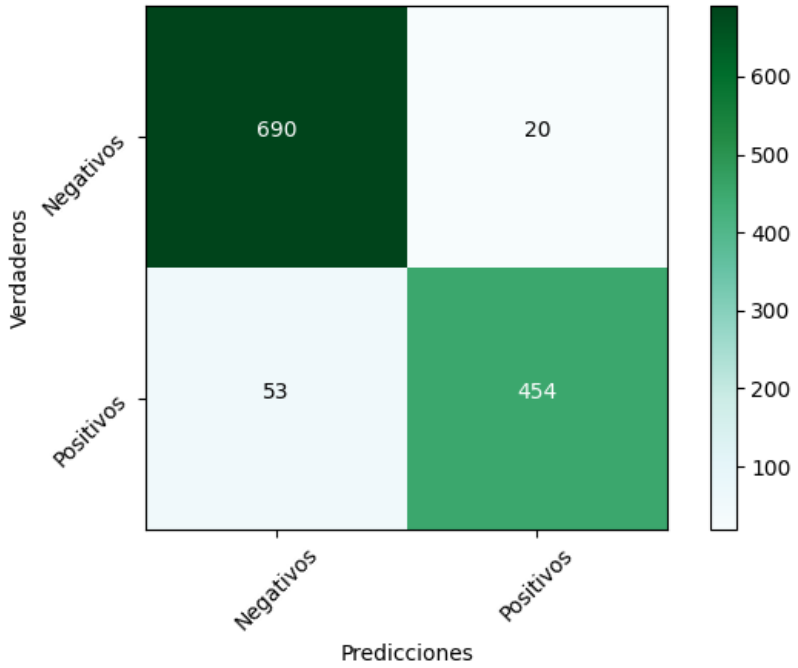


Figura 3.3: Matriz de confusión del modelo de clasificación SVM lineal.

- **Exactitud Promedio/(Varianza)**= 0.9426 (+/- 0.0414)

- **Exactitud de cada grupo** = [0.90934066 0.90247253 0.94230769 0.95604396 0.97252747 0.947802 0.96016484 0.93818681 0.9546079 0.94222834]

Observando los resultados obtenidos de la evaluación sobre este modelo SVM de kernel radial, se puede ver hay una notable mejora con respecto al modelo inicial lineal y sus parámetros por defecto. Por lo tanto, este modelo de clasificación SVM radial es el que se utilizó en el algoritmo de detección.

3.2. Pruebas Experimentales en el Algoritmo de Detección y Conteo

Como se ha mencionado anteriormente, la implementación del modelo propuesto se basa en el marco del aprendizaje de máquina SVM y extracción de características HOG. La experimentación se centra en dos puntos: la parte de la detección de cabeza-hombros y el conteo de estas en imágenes de multitudes.

La evaluación del desempeño del sistema se realizó con imágenes del conjunto de datos JHU-Crowd++ [38], este contiene imágenes de diferentes dimensiones con una gran variación de multitudes, diferentes perspectivas de cámara, variaciones de ilumina-ción, regiones de fondo complejas y desordenadas que puede confundirse con multitud. Además los autores, dividen las imágenes en las siguientes categorías:

- **Multitudes de Alta Densidad:** Contiene imágenes con un recuento de más de 500 personas.
- **Multitudes de Media Densidad:** Imágenes que contienen un recuento entre 51 y 500 personas.
- **Multitudes de Baja Densidad:** Imágenes que contienen un recuento entre 0 y 50 personas.

3.2.1. Multitudes de Alta Densidad

La Fig.3.4 muestra el conteo real de personas y las detecciones realizadas en 24 imágenes de prueba de multitudes de alta densidad. Además, en la Fig.3.5 se muestran algunas de las imágenes a las que se le aplicó el detector.

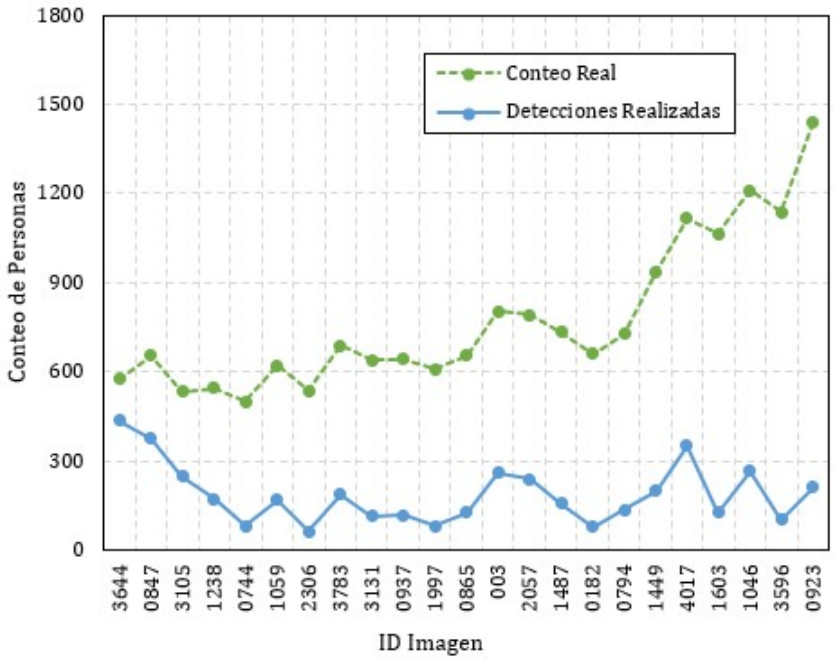


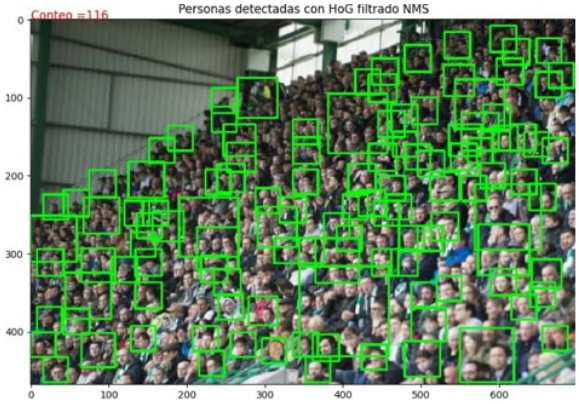
Figura 3.4: Representación del conteo real de personas y las detecciones realizadas por el algoritmo en imágenes de multitudes de alta densidad.

Conteo Real = 575
Detecciones = 437

(a)

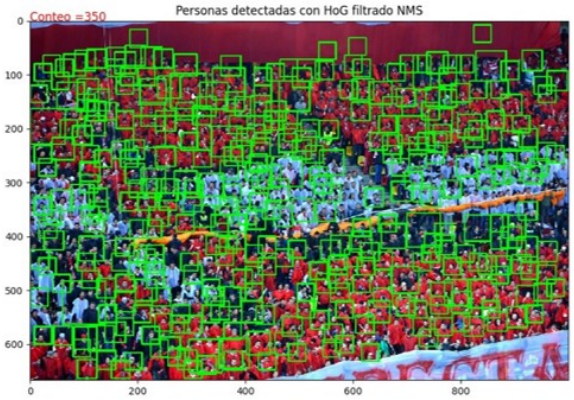
Conteo Real = 545
Detecciones = 171

(b)



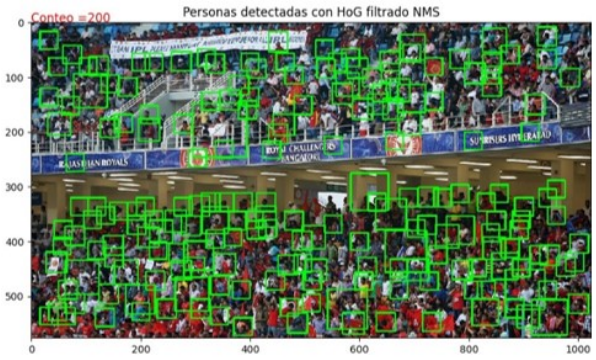
Conteo Real = 642
Detecciones = 116

(c)



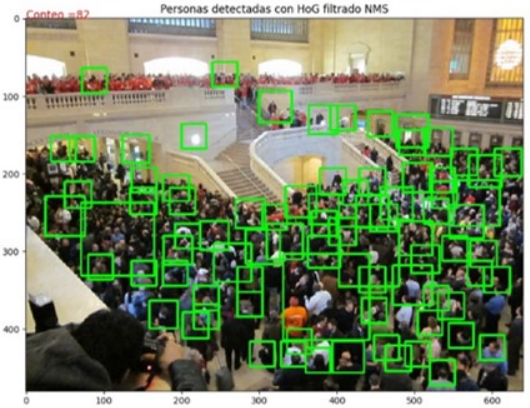
Conteo Real = 1116
Detecciones = 350

(d)



Conteo Real = 936
Detecciones = 200

(e)



Conteo Real = 501
Detecciones = 82

(f)

Figura 3.5: Detecciones realizadas en imágenes de multitudes de alta de densidad en diferentes escenarios (a), (c) Coliseo, b) Calles, (d),(e) Estadio, (f)Congreso.

Analizando los resultados obtenidos en la Fig.3.4, se ve que el número de detecciones realizadas por el algoritmo, están muy lejos de acercarse al conteo real de personas de cada imagen. Como se puede observar en la Fig.3.5, en las imágenes no se obtienen buenos resultados en la detección, ya que existen muchos falsos positivos y negativos.

Esto se debe a que en la multitudes de densidad alta, las cabezas son a menudo muy pequeñas y están ocluidas, también a más factores como, las variaciones en la escala y perspectiva, densidad no uniforme, entre otros. Por lo tanto, se deduce que el algoritmo es ineficaz para ser aplicado a imágenes de este tipo de multitudes.

Como observación especial, en la imagen 3.5a se puede apreciar que el algoritmo hizo mejores detecciones a comparación de las otras imágenes, la diferencia es que en esta imagen presenta personas sin mayores oclusiones, densidad uniforme y un ángulo de cámara normal.

3.2.2. Multitudes de Media Densidad

La Fig.3.6 muestra el conteo real de personas y conteo estimado en 28 imágenes de prueba de multitudes de densidad media y en la Fig.3.7 se aprecia algunas de las imágenes a las que se le aplicó el detector y sus resultados.

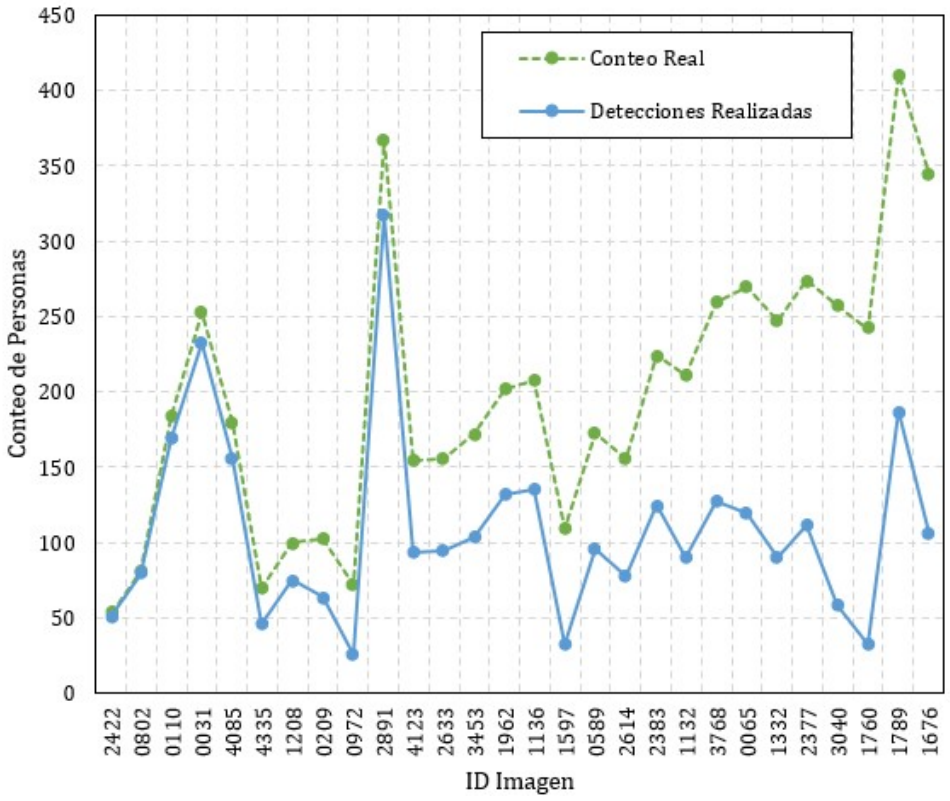


Figura 3.6: Representación del conteo real de personas y las detecciones realizadas por el algoritmo en imágenes de multitudes de densidad media.

Conteo Real = 184
Detecciones = 169

(a)

Conteo Real = 367
Detecciones = 317

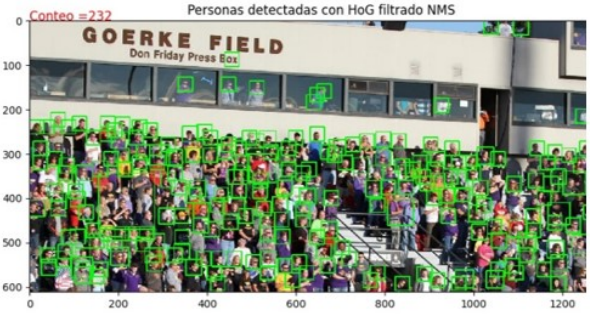
(b)

Conteo Real = 344
Detecciones = 106

(c)

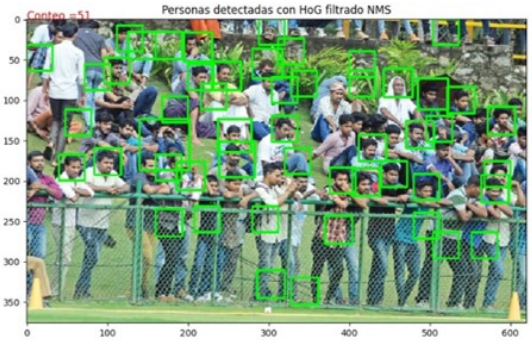
Conteo Real = 224
Detecciones = 125

(d)



Conteo Real = 253
Detecciones = 232

(e)



Conteo Real = 53
Detecciones = 51

(f)

Figura 3.7: Detecciones realizadas en imágenes de multitudes de media densidad en diferentes escenarios (a), (b), (f) Estadio, (c), (d) Coliseo.

Observando los resultados obtenidos en la Fig.3.6, se aprecia que el rendimiento del algoritmo en esta categoría presenta unos resultados más favorables a comparación de los obtenidos en las multitudes de alta densidad, aunque sigue existiendo algunos resultados donde la estimación esta muy lejana al número real de personas.

En la Fig.3.7 se encuentran las imágenes con las detecciones obtenidas del algoritmo, se puede ver que los resultados de la estimación están cercanos al conteo real, pero hay que considerar que en las detecciones realizadas siguen existiendo varios falsos positivos y negativos en cada imagen.

3.2.3. Multitudes de Baja Densidad

Los resultados obtenidos se muestran a continuación en la Fig.3.8, donde se puede apreciar el conteo real de personas y conteo estimado en 14 imágenes de prueba de multitudes de baja densidad y en la Fig.3.9 se muestran algunas de las imágenes a las que se le aplicó el detector y sus resultados.

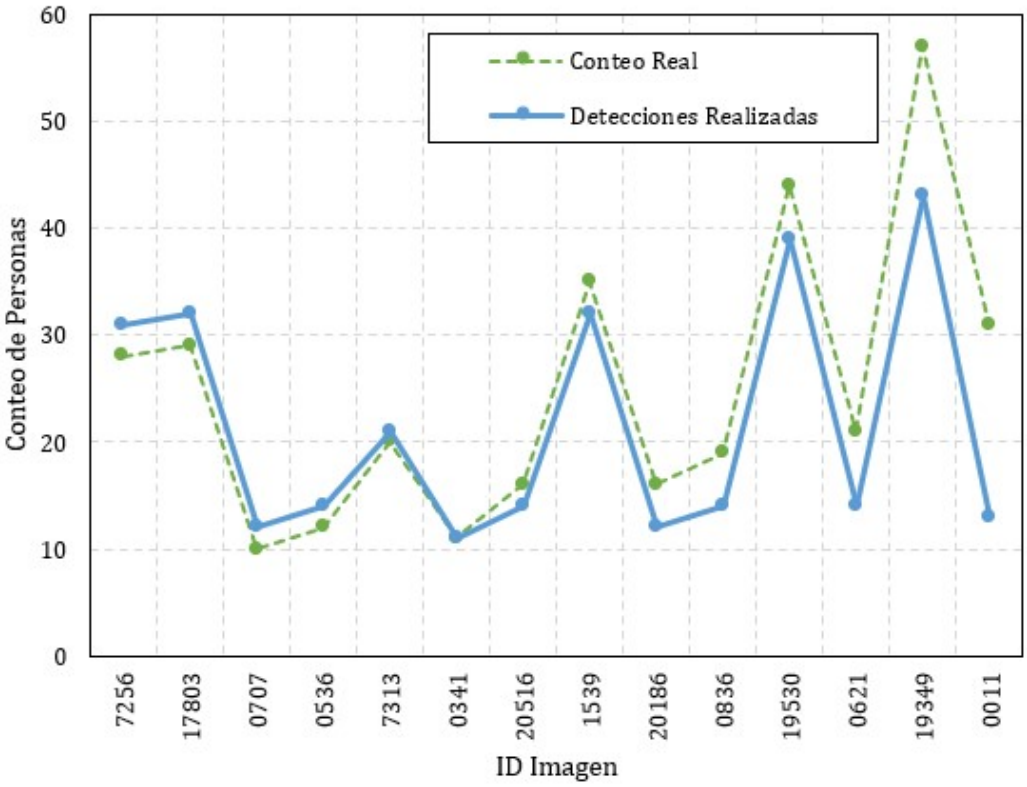
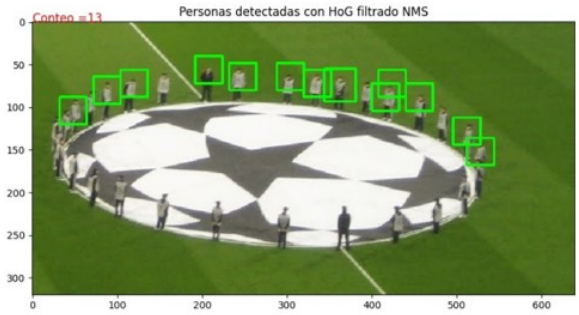


Figura 3.8: Representación del conteo real de personas y las detecciones realizadas por el algoritmo en imágenes de multitudes de baja densidad.

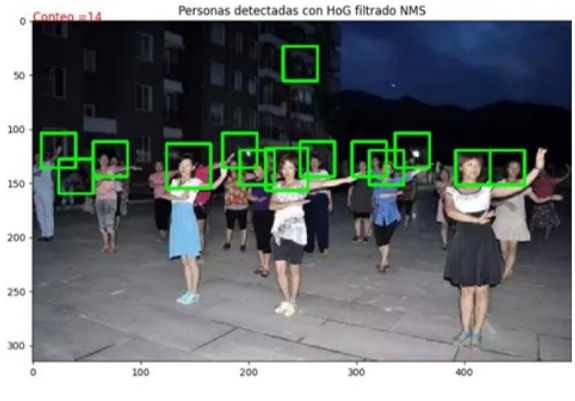


Conteo Real = 31
Detecciones = 13

(a)

Conteo Real = 20
Detecciones = 21

(b)

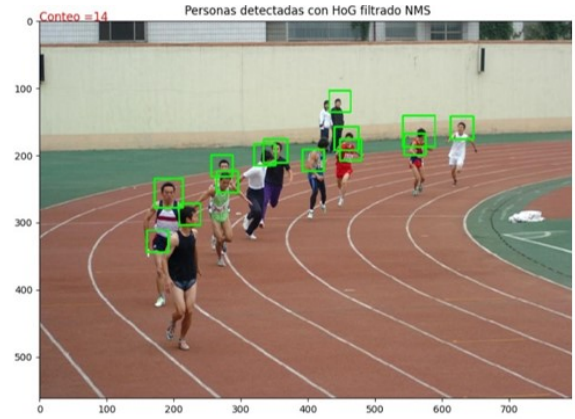


Conteo Real = 21
Detecciones = 14

(c)

Conteo Real = 10
Detecciones = 12

(d)



Conteo Real = 12
Detecciones = 14

(e)

Conteo Real = 57
Detecciones = 43

(f)

Figura 3.9: Detecciones realizadas en imágenes de multitudes de baja densidad en diferentes escenarios (a) Estadio, (b), (c), (f) Calle, (d) Campo, (e) Pista atlética

En esta categoría se puede observar en la Fig.3.8 que los valores predichos son muy cercanos al conteo real. A pesar de esto, en la Fig.3.9 se encuentran algunas de las imágenes a las que se le aplicó el detector, en las que se aprecia con más claridad los falsos positivos y negativos que se presentan en cada una de ellas. .

Por lo tanto, se deduce que este es uno de los principales inconvenientes que presenta el sistema, haciendo que este no sea óptimo para una implementación. Una posible solución a este problema podría ser aumentar la cantidad de muestras a los conjuntos de datos de entrenamiento y de evaluación.

Capítulo 4

Conclusiones y Trabajo Futuro

Una vez culminado el presente trabajo de grado, en este capítulo se realizará un análisis de como se han cumplido los objetivos planteados en el capítulo . Además se mencionan posibles líneas de trabajo a futuro que pueden conseguir mejores resultados.

4.1. Conclusiones

- Se realizó un estudio exhaustivo de los diferentes algoritmos existentes para la detección de personas, con el fin de definir los algoritmos a utilizar en el desarrollo del sistema de detección y conteo de personas en imágenes de multitudes, seleccionando como los principales, el extractor de características Histogramas de Gradientes Orientados (HOG) y el clasificador de aprendizaje supervisado Máquinas de Vectores de Soporte (SVM).

- En este trabajo de grado se desarrolló y evaluó un algoritmo de detección y conteo de personas en imágenes de multitudes, usando como región de interés (ROI) la parte de la cabeza-hombros, con el fin de poder detectar a personas en escenas abarrotadas y con oclusiones. Sin embargo, observando los resultados obtenidos, se deduce que el método basado en la detección es muy ineficaz para cuando la densidad de la multitud es alta y existen oclusiones severas.

- El llevar a cabo una exploración de hiperparámetros para la SVM y realizar un reentrenamiento, fue un paso muy importante, ya que de esa manera se logró mejores resultados en las métricas de evaluación aplicadas sobre el modelo.

- Un punto importante, es el papel de los parámetros en la etapa de generación de candidatos, ya que el ajuste de la escala y el desplazamiento de la ventana, influyen en la precisión y velocidad de procesamiento del detector, por ende esto implica también el coste computacional.

- Se realizó las respectivas pruebas usando imágenes de un conjunto de datos de conteo de multitudes sin restricciones, observando que el sistema tiende a dar mejores resultados en imágenes donde presentan una densidad uniforme con ángulo de cámara normal y que las personas se encuentren sin oclusiones severas, probando así el desempeño del sistema.

- Como conclusión final se puede decir que, los resultados que se obtuvieron en este trabajo de grado no son realmente a los que se esperaba llegar cuando se planteó el mismo. A pesar de mucho estudio, análisis, trabajo y revisiones, el desempeño del sistema solo evidenció que no es confiable. Además, el tiempo que demora el sistema en procesar una imagen y mostrar los resultados es demasiado largo, por lo que sería imposible de ajustar a las necesidades de un sistema en tiempo real.

4.2. Trabajo Futuro

- **Descriptor:** En este trabajo de grado se basó en el descriptor HOG por ser uno de los más robustos y su gran utilidad en la detección de personas. Sin embargo, existe un gran problema con las oclusiones y abarrotamiento, es por eso que combinarlo con otro descriptor o simplemente cambiar por otro tipo de descriptor puede ofrecer mejores resultados.

- **Clasificador:** Utilizar otro tipo de clasificador puede ser una opción o aplicar la técnica del Boosting, que consiste en utilizar un conjunto de clasificadores débiles en cascada y así obtener un clasificador final robusto. Además, se puede considerar la opción que en lugar de un clasificador utilizar un modelo de regresión, que puede ofrecer resultados más precisos.

- **Aprendizaje profundo:** El concepto de aprendizaje profundo surge del estudio de las redes neuronales artificiales. Utilizar los métodos de vanguardia es la mejor opción, como por ejemplo, las redes neuronales convolucionales (CNN) que han demostrado su gran capacidad en varias tareas de visión por computador, hoy en día estas son unas de las más utilizada en la estimación de personas en imágenes de multitudes.

Lista de Códigos

En este apartado se muestran los diferentes códigos que se han realizado para llevar a cabo el desarrollo del sistema.

.1. Entrenamiento

```
# Cargo las librerías necesarias
import numpy as np
import cv2 as cv2
from os import scandir, getcwd
from os.path import abspath
from sklearn.svm import SVC
from tqdm import tqdm
from joblib import dump

# Seteo de paths y archivos de prueba
PATH_POSITIVE_TRAIN="Base de Datos/Train/Positivos/"
PATH_NEGATIVE_TRAIN="Base de Datos/Train/Negativos/"

#Definición de Parámetros HOG
winSize=(32, 32)
blockSize=(16, 16)
blockStride=(8, 8)
cellSize=(8, 8)
nbins=9
derivAperture=1
winSigma=-1
histogramNormType=0
L2HysThreshold=0.2
nlevels=64

#Funciòn para la lectra de datos del entrenamiento
def loadTrainingData():
    """
    Retorna:
    trainingData: matriz con las instancias
    classes: vector con las clases de cada instancia
    """
    # Matriz de descriptores de las clases
    trainingData=np.array([])
    trainingDataNeg=np.array([])

    #----------------------------------------Casos positivos
    # Obtengo la lista de casos positivos
```

```
39      listFiles=[abspath(arch.path) for arch in scandir(PATH_POSITIVE_TRAIN
        ) if arch.is_file()]
40      # Itero los archivos
41      for i in tqdm(range(len(listFiles))):
42          file=listFiles[i]
43          # Leo y redimensiono la imagen
44          imag=cv2.imread(file, cv2.IMREAD_COLOR)
45          img=cv2.resize(imag, (32, 32))
46          #Crea el objeto HOG y realiza su cálculo
47          hog=cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize,
        nbins, derivAperture, winSigma, histogramNormType, L2HysThreshold,
        nlevels)
48          h=hog.compute(img)
49          # Lo paso a 1 dimension
50          h2=h.ravel()
51          # Agrego a trainingData
52          trainingData=np.hstack((trainingData, h2))
53
54      print("Leidas " + str(len(listFiles)) + " imágenes de entrenamiento
        -> positivas")
55      # Hago un reshape
56      trainingData=trainingData.reshape((len(listFiles), len(h2)))
57      # Genero el vector de clases
58      classes=np.ones(len(listFiles))
59
60      #-----------------------------------------------Casos negativos
61      # Obtengo la lista de casos positivos
62      listFilesNeg=[abspath(arch.path) for arch in scandir(
        PATH_NEGATIVE_TRAIN) if arch.is_file()]
63      # Itero los archivos
64      for i in tqdm(range(len(listFilesNeg))):
65          file=listFilesNeg[i]
66          # Leo y redimensiono la imagen
67          imag=cv2.imread(file, cv2.IMREAD_COLOR)
68          img=cv2.resize(imag, (32, 32))
69          # Creo el objeto HOG y realiza su cálculo
70          hog=cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize,
        nbins, derivAperture, winSigma, histogramNormType, L2HysThreshold,
        nlevels)
71          h=hog.compute(img)
72          # Lo paso a 1 dimension
73          h2=h.ravel()
74          # Agrego a trainingData
75          trainingDataNeg=np.hstack((trainingDataNeg, h2))
76
77      print("Leidas " + str(len(listFilesNeg)) + " imágenes de
        entrenamiento -> negativas")
78      # Hago un reshape
79      trainingDataNeg=trainingDataNeg.reshape((len(listFilesNeg), len(h2)))
80      # Genero el vector de clases
81      classesNeg=np.zeros(len(listFilesNeg))
82
83      # Matriz de características
84      trainingData=np.concatenate((trainingData, trainingDataNeg), axis=0)
85      # Vector características de las clases
86      classes=np.concatenate((classes, classesNeg), axis=0)
87      return trainingData, classes
```

```
88
89 # Obtengo los datos de trainning
90 X_Train, y_train = loadTrainingData()
91
92 # Guardo los vectores obtenidos
93 np.save("X_Train_HOG", X_Train)
94 np.save("y_train_HOG", y_train)
95
96 # Creo una SVM con kernel linear
97 clf = SVC(kernel="linear", C=1, probability = True)
98
99 # Entreno la SVM
100 clf.fit(X_Train, y_train)
101
102 # Guardo el modelo entreando
103 dump(clf, 'HOG_clf.joblib')
```

Listing 1: Código de entrenamiendo de la SVM

.2. Evaluación

```
1  #Cargo las librerías necesarias
2  import itertools
3  from sklearn.metrics import confusion_matrix, classification_report
4  import numpy as np
5  import cv2
6  from matplotlib import pyplot as plt
7  from os import scandir
8  from os.path import abspath
9  from sklearn.model_selection import cross_val_score
10 from sklearn.svm import SVC
11 from tqdm import tqdm
12 from joblib import load
13 from sklearn.metrics import roc_curve
14 from sklearn.metrics import precision_recall_curve
15
16 # Seteo de paths y archivos de evaluación
17 PATH_POSITIVE_TEST="Base_de_Datos/Test/Positivos/"
18 PATH_NEGATIVE_TEST="Base_de_Datos/Test/Negativos/"
19
20 #cargo el modelo entrenado
21 svml = load('HOG_clf.joblib')
22
23 #Defino los parámetros HOG
24 winSize=(32, 32)
25 blockSize=(16, 16)
26 blockStride=(8, 8)
27 cellSize=(8, 8)
28 nbins=9
29 derivAperture=1
30 winSigma=-1
31 histogramNormType=0
32 L2HysThreshold=0.2
33 #gammaCorrection=True
34 nlevels=64
```

```
35
36 #Funciòn para la lectura de datos del test
37 def loadTestingData():
38
39     # Matriz de descriptores
40     testingData=np.array([])
41     testingDataNeg=np.array([])
42
43     #-----------------------------------------------Casos positivos
44     # Obtengo la lista de casos positivos
45     listFiles=[abspath(arch.path) for arch in scandir(PATH_POSITIVE_TEST)
       if arch.is_file()]
46     # Itero los archivos
47     for i in tqdm(range(len(listFiles))):
48         file=listFiles[i]
49         # Leo y redimensiono la imagen
50         imag=cv2.imread(file, cv2.IMREAD_COLOR)
51         img=cv2.resize(imag, (32, 32))
52         # Crea el objeto HOG y realiza su cálculo
53         hog=cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize,
    nbins, derivAperture, winSigma,
54                               histogramNormType, L2HysThreshold, nlevels)
55         h=hog.compute(img)
56         # Lo paso a 1 dimension
57         h2=h.ravel()
58         # Agrego a testingData
59         testingData=np.hstack((testingData, h2))
60     print("Leidas " + str(len(listFiles)) + " imágenes de entrenamiento
    -> positivas")
61     # Hago un reshape
62     testingData=testingData.reshape((len(listFiles), len(h2)))
63     # Genero el vector de clases
64     classes=np.ones(len(listFiles))
65
66     #-----------------------------------------------Casos negativos
67     # Obtengo la lista de casos positivos
68     listFilesNeg=[abspath(arch.path) for arch in scandir(
    PATH_NEGATIVE_TEST) if arch.is_file()]
69     # Itero los archivos
70     for i in tqdm(range(len(listFilesNeg))):
71         file=listFilesNeg[i]
72         # Leo la imagen
73         imag=cv2.imread(file, cv2.IMREAD_COLOR)
74         img=cv2.resize(imag, (32, 32))
75         # Crea el objeto HOG y realiza su cálculo
76         hog=cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize,
    nbins, derivAperture, winSigma,
77                               histogramNormType, L2HysThreshold, nlevels)
78         h=hog.compute(img)
79         # Lo paso a 1 dimension
80         h2=h.ravel()
81         # Agrego a trainingData
82         testingDataNeg=np.hstack((testingDataNeg, h2))
83
84     print("Leidas " + str(len(listFilesNeg)) + " imágenes de
    entrenamiento -> negativas")
85     # Hago un reshape
```

```
86      testingDataNeg=testingDataNeg.reshape((len(listFilesNeg), len(h2)))
87      # Genero el vector de clases
88      classesNeg=np.zeros(len(listFilesNeg))
89
90      # Merge de los datos
91      testingData=np.concatenate((testingData, testingDataNeg), axis=0)
92      classes=np.concatenate((classes, classesNeg), axis=0)
93      return testingData, classes
94
95  # Obtengo los datos de test
96  X_Test, y_test=loadTestingData()
97  target_names=['Negativos', 'Positivos']
98
99  # Guardo los datos
100 np.save("X_Test_HOG", X_Test)
101 np.save("y_test_HOG", y_test)
102
103 # Realizo predicciones sobre el dataset de test
104 predicciones=svml.predict(X_Test)
105
106 #Función para graficar una matriz de confusion
107 def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt
        .cm.BuGn):
108     """
109     Parámetros:
110     cm: matriz de confusion
111     classes: etiquetas de las clases
112     cmap: colores a emplear para graficar
113     """
114     plt.imshow(cm, interpolation='nearest', cmap=cmap)
115     plt.title(title)
116     plt.colorbar()
117     tick_marks=np.arange(len(classes))
118     plt.xticks(tick_marks, classes, rotation=45)
119     plt.yticks(tick_marks, classes, rotation = 45)
120
121     fmt='d'
122     thresh=cm.max() / 2.
123     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))
        :
124         plt.text(j, i, format(cm[i, j], fmt),
125                  horizontalalignment="center",
126                  color="white" if cm[i, j] > thresh else "black")
127
128     plt.ylabel('Verdaderos')
129     plt.xlabel('Predicciones')
130     plt.tight_layout()
131
132 # Calculo la matriz de confusion
133 cnf_matrix = confusion_matrix(y_test, predicciones)
134
135 # Grafico la matriz de confusion
136 np.set_printoptions(precision=2)
137 plt.figure()
138 plot_confusion_matrix(cnf_matrix, classes=target_names,title='Matriz de
        confusión')
139 plt.savefig("img/Matriz_de_Confusion.png")
```

```
plt.show()

#-------------------------------Mejoras. K-Fold Cross Validation

# Hago un merge de los datos de train y test
allData = np.concatenate((np.load('X_Train_HOG.npy'), X_Test), axis=0)
allClasses = np.concatenate((np.load('y_train_HOG.npy'), y_test))

# Guardo los datos mergeados
np.save("allData_HOG", allData)
np.save("allClasses_HOG", allClasses)

# Ahora implemento 10 Fold Cross Validation
scores = cross_val_score(SVC(kernel="linear", C=1, probability=True),
    allData, allClasses, cv=10, n_jobs = 4)

# Con esto puede calcular la exactitud promedio y la varianza
print("Exactitud Promedio (Varianza): %0.4f (+/- %0.4f)" %(scores.mean()
    , scores.std() * 2))

# Muestro la exactitud obtenida en cada fold de 10 Fold CV
np.set_printoptions(precision=4)
print("Exactitud de cada fold= {}".format(scores))
```

Listing 2: Código de evaluación para el modelo SVM entrenado

.3. Exploración de Hiperparámetros

```
import datetime
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from joblib import dump, load
import pandas as pd
from sklearn.model_selection import GridSearchCV
import os

 #-----------------------------Exploración de hiperparámetros SVM
print('Inicia proceso...')
start=datetime.datetime.now()

#Defino la función Grid-search
def grid_search(classifier, X_train, y_train, gparams,score='accuracy',
    cv=10, jobs=-1):
    """
    Realizo una búsqueda de hiperparametros utilizando la función
    GridSearchCV de sklearn.model_selection
    Parámetros:
    score: valor de evaluación
    cv: número de grupos de la validación cruzada
    """
    # Inicializo la clase de GridSearch
    gd_sr=GridSearchCV(estimator=classifier,param_grid=gparams,scoring=
    score,cv=cv,n_jobs=jobs)

```

```
    # Hago un fit de los datos de Entrenamiento
    gd_sr.fit(X_train, y_train)

    # Resultados
    best_result=gd_sr.best_score_
    print("Mejor Resultado = {}".format(best_result))

    # Obtengo los mejores paràmetros
    best_parameters=gd_sr.best_params_
    print("Mejores parámetros = {}".format(best_parameters))

    return gd_sr

    # Defino los parámetros de búsqueda
params={'C': [0.01, 0.1, 1, 10],'gamma': [0.01, 0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly']
            }
results=grid_search(SVC(), np.load("allData_HOG.npy"), np.load("allClasses_HOG.npy"), params, score='accuracy', cv=5, jobs=6)

cv_results = results.cv_results_
scores_df = pd.DataFrame(cv_results).sort_values(by='rank_test_score')
scores_grid_search = scores_df[['param_C','param_gamma','param_kernel','mean_test_score', 'std_test_score']]
print(scores_grid_search)
scores_df.to_csv("HOG_hiperparametros.csv", index=False)

    # Guardo los resultados de gridsearch
dump(results, 'results_gridsearch_HOG.pkl')
```

Listing 3: Código de exploración de hiperámetros de la SVM

.4. Detección y conteo

```

import os
import sqlite3
import time
from datetime import datetime
import cv2
import matplotlib.pyplot as plt
import numpy as np
from imutils.object_detection import non_max_suppression
from joblib import load

print('Inicia proceso...')
start=datetime.now()

#Cargo el modelo entrenado
#hog_clf= load('HOG_clf.joblib') #modelo lineal
hog_clf = load('HOGbest_clf.joblib') #paràmetros SVM- modelo RBF
#Dirección de las imágenes de prueba
ruta = ("images/4017.jpg")
base=os.path.basename(ruta)

```

```
22 #Funciòn piramide
23 def get_pyramid(img):
24
25     pyramid = [img] # Asigno la imagen sin escala como primer elemento
26     new_level = img # Imagen sobre la que ire realizando las
       disminuciones de tamaño
27     while np.shape(new_level)[0] >= 32 and np.shape(new_level)[1] >= 32:
28         new_level = cv2.GaussianBlur(src=new_level, ksize=(7, 7), sigmaX
       =1)
29         # 0.9524 is 1 / 1.05
30         new_level = cv2.resize(new_level, dsize=(0, 0), fx=0.9524, fy
       =0.9524)
31         pyramid.append(new_level)
32     return pyramid
33 #Leo la imagen
34 imagen = cv2.imread(ruta)
35 piramide = get_pyramid(imagen)
36
37 # Función para la detección de peatones utilizando el descriptor HoG y
       una SVM
38 def piramide_HOG(piramide_img, pred_thr=0.965):
39     """
40     Parámetros:
41      piramide_img: piramide imágenes
42      pred_thr: thresold de la probabilidad para guardar una deteccion
       como peaton
43     """
44     # Definición de Parámetros HOG
45     winSize=(32, 32)
46     blockSize=(16, 16)
47     blockStride=(8, 8)
48     cellSize=(8, 8)
49     nbins=9
50     derivAperture=1
51     winSigma=-1
52     histogramNormType=0
53     L2HysThreshold=0.2
54     #gammaCorrection=True
55     nlevels=64
56     hog=cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize,
       nbins, derivAperture, winSigma,
57                           histogramNormType, L2HysThreshold, nlevels )
58     # Imagen sin escalar
59     img_sin_escala=piramide_img[0]
60
61     # Variables auxiliares
62     cantidad=0
63     peatones=[]  # Aquí guardo todas las detecciones
64
65     # Itero las imagenes de la piramide
66     for index in range(len(piramide_img)):
67
68         # Obtengo la escala y la imagen
69         imgp=piramide_img[index]
70         # Calculo la escala de la imagen, a partir del tamaño de la
       original
71         escala=imgp.shape[0] / img_sin_escala.shape[0]
```

```
72
73          # Voy a ir desplazandome por la imagen
74          alto_maximo=imgp.shape[0]
75          ancho_maximo=imgp.shape[1]
76
77          # Itero el alto
78          for wy in np.arange(0, alto_maximo - 32, 8):
79              # Itero el ancho
80              for wx in np.arange(0, ancho_maximo - 32, 8):
81
82                  # Obtengo la porcion de imagen 32X32
83                  cropped=imgp[wy:wy + 32, wx:wx + 32]
84                  # Calculo el hog
85                  h=hog.compute(cropped)
86                  # Lo paso a 1 dimension
87                  h2=h.reshape(1,-1)
88                  # Pruebo si hay un peaton
89                  pred=hog_clf.predict(h2)
90                  # Calculo las probabilidades de las dos clases
91                  predprob=hog_clf.predict_proba(h2)
92
93                  # Si la prediccion de peaton es positiva y supera el
    threshold
94                  if pred == 1 and predprob[0][1] > pred_thr:
95                      # Guardo la posicion de la deteccion para graficar
96                      detectado=[]  # Genero lista vacia
97                      if escala < 1:
98                          detectado.append(wx / escala)
99                          detectado.append(wy / escala)
100                         detectado.append((wx / escala) + (32 / escala))
101                         detectado.append((wy / escala) + (32 / escala))
102                     else:
103                         detectado.append(wx)
104                         detectado.append(wy)
105                         detectado.append(wx + 32)
106                         detectado.append(wy + 32)
107                     peatones.append(detectado)  # Lo agrego a la lista
    general
108                     cantidad+=1  # Cuento un peaton más
109     return peatones
110
111 peatones_encontrados=piramide_HOG(piramide)
112 copia_hog=imagen.copy()
113 for peatonf in peatones_encontrados:
114     v1=int(peatonf[0])
115     v2=int(peatonf[1])
116     v3=int(peatonf[2])
117     v4=int(peatonf[3])
118     cv2.rectangle(copia_hog, (v1, v2), (v3, v4), (255, 0, 0), 2)
119
120     # Grafico las detecciones del HoG comun
121 img_peatonesHoG=copia_hog[:, :, ::-1]
122 fig = plt.figure(figsize = (10, 10))
123 ax = fig.add_subplot(111)
124 ax.imshow(img_peatonesHoG, interpolation='none')
125 plt.title('Personas detectadas con HoG')
126 plt.savefig("Pruebas/HoG_peaton" + base )
```

```
127 plt.show()
128
129 copia_hog_NMS = imagen.copy()
130 # Obtengo los cuadrados para calcular el NMS
131 rects = np.array([[x, y, w, h] for (x, y, w, h) in peatones_encontrados])
132 pick = non_max_suppression(rects, probs=None, overlapThresh=0.45)
133 print(pick)
134
135 #Conteo de las detecciones realizadas
136 print('Personas detectadas: ', len(peatones_encontrados))
137 print('Personas detectadas con filtro NMS: ', len(pick))
138
139 # Grafico los cuadrados finales
140 for (xA, yA, xB, yB) in pick:
141     cv2.rectangle(copia_hog_NMS, (xA, yA), (xB, yB), (0, 255, 0), 2)
142 texto = 'Conteo =' + str(len(pick))
143
144 # Grafico las detecciones después del NMS
145 img_peatonesHoGNMS = copia_hog_NMS[:,:,::-1]
146 fig1 = plt.figure(figsize = (10, 10))
147 ax = fig1.add_subplot(111)
148 ax.imshow(img_peatonesHoGNMS, interpolation='none')
149 plt.title('Personas detectadas con HoG filtrado NMS')
150 plt.text(0, 0, texto ,fontsize=12, color="r")
151 plt.savefig("Pruebas/HOG_NMS_peaton"+ base )
152 plt.show()
```

Listing 4: Código de la detección y conteo de personas en imágenes estáticas

Bibliografía

[1] N. Hernández y G. Zapata, *Guía Técnica para la reglamentación local de eventos con aglomeraciones de público*, 2018. dirección: `https://repositorio.gestiondelriesgo.gov.co/bitstream/handle/20.500.11762/27735/Guia_%20aglomeraciones_publico.pdf?sequence=6&isAllowed=y.`.

[2] *Nuevo coronavirus 2019*, Organización Mundial de la Salud, 2019. dirección: `https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019`.

[3] "Protocolo de Bioseguridad Centros Comerciales Post Cuarentena," Departamento de Seguridad y Salud Ocupacional, Quito, inf. téc., 2020, págs. 27-33.

[4] R. García, "Detección y conteo de personas, a partir de mapas de profundidad cenitales capturados con cámaras TOF," Universidad de Alcalá. Escuela Politécnica Superior, Madrid, 2015.

[5] J. Silveira, S. Raupp y C. R. Jung, "Crowd analysis using computer vision techniques," *IEEE Signal Processing Magazine*, págs. 66-77, 2010.

[6] *La revolución de la visión artificial en 4 ejemplos*, 2019. dirección: `https://planetachatbot.com/la-revoluci%C3%B3n-de-la-visi%C3%B3nartificial-en-4-ejemplos-e7d438bfc8a8`.

[7] A. García y W. Moreno, "Sistema de Conteo Automático de Flujo de Personas por medio de Visón Artificial," Bogotá, 2016, págs. 12-12.

[8] B. Zhan, D. N. Monekosso, P. Remagnino, S. A. Velastin y L.-Q. Xu, "Análisis de multitudes: una encuesta," *Machine Vision and Applications*, n.º 5-6, págs. 345-357, 2008.

[9] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[10] L. Camacho, A. Felipe, R. Menjura y N. Esteban, "Detección y conteo de personas en espacios cerrados utilizando estrategias basadas en visión artificial," Pontificia Universidad Javeriana.

[11] A. Kowcika y S. Sridhar, ""A Literature Study on Crowd (People) Counting with the Help of Surveillance Videos," *International Journal of Innovative Technology and Research*, págs. 2353-2361, 2016.

[12] A. Rosebrock, *OpenCV People Counter - PyImageSearch*, 2018. dirección: `https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/`.

[13] A. Rosebrock, *Pedestrian Detection OpenCV - PyImageSearch*, 2015. dirección: `https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/`.

[14] Martínez y M. Sánchez, "Detección de Personas mediante Técnicas de Aprendizaje Automátivo: SVM Y CNN," Universidad Polítécnica de Madrid, Madrid, 2018.

[15] D. Hoiem, *Object Category Detection: Sliding Windows*. dirección: https://courses.engr.illinois.edu/cs543/sp2011/lectures/Lecture%2019%20-%20Sliding%20Window%20Detection%20-%20Vision_Spring2011.pdf.

[16] *Detección de objetos*. dirección: https://www.coursera.org/learn/deteccion-objetos/lecture/A34LS/l3-5-a-evaluacion-dela-clasificacion-por-ventana-i.

[17] C. Troya Sherdek, *Generación de ventanas candidato*, 2016. dirección: https://cesartroyasherdek.wordpress.com/2016/03/10/generacion-de-ventanas-candidato/.

[18] C. R. Sancho, "Pedestrian Detection using a boosted cascade of Histogram of Oriented Gradients," Tesis doct., Citeseer, 2014.

[19] D. G. Lowe, "Object recognition from local scale-invariant features," en *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, págs. 1150-1157.

[20] S. Gong, "Real-time implementation of counting people in a crowd on the embedded reconfigurable architecture on the unmanned aerial vehicle," Tesis doct., Université Bourgogne Franche-Comté, 2021.

[21] P. Viola y M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, n.º 2, págs. 137-154, 2004.

[22] C. P. Papageorgiou, M. Oren y T. Poggio, "A general framework for object detection," en *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, IEEE, 1998, págs. 555-562.

[23] N. Dalal y B. Triggs, "Histograms of oriented gradients for human detection," en *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, vol. 1, 2005, págs. 886-893.

[24] F. Suard, V. Guigue, A. Rakotomamonjy y A. Benshrair, "Pedestrian detection using stereo-vision and graph kernels," en *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, IEEE, 2005, págs. 267-272.

[25] P. Viola y M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade," *Advances in neural information processing systems*, vol. 14, págs. 1311-1318, 2001.

[26] C. Cortes y V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, n.º 3, págs. 273-297, 1995.

[27] J. M. Rodríguez Fernández, "Computer vision for Pedestrian detection using Histograms of Oriented Gradients," 2014. dirección: http://hdl.handle.net/2099.1/21343.

[28] D. Lee, G. Cha, M.-H. Yang y S. Oh, "Individualness and Determinantal Point Processes for Pedestrian Detection," *Computer Vision – ECCV 2016*, págs. 330-346, 2016. DOI: 10.1007/978-3-319-46466-4_20.

[29] *What is Python? Executive Summary*, 2021. dirección: https://www.python.org/doc/essays/blurb/.

[30] *About OpenCV*, 2021. dirección: https://opencv.org/about/.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, págs. 2825-2830, 2011.

[32] N. Dalal, *INRIA Person dataset*, 2005. dirección: http://pascal.inrialpes.fr/data/human/.

[33] L. Wang, J. Shi, G. Song e I.-f. Shen, "Object Detection Combining Recognition and Segmentation," *Computer Vision – ACCV 2007*, págs. 189-199, 2007. DOI: 10.1007/978-3-540-76386-4_17.

[34] D. Peng, Z. Sun, Z. Chen, Z. Cai, L. Xie y L. Jin, "Detecting Heads using Feature Refine Net and Cascaded Multi-scale Architecture," *arXiv preprint arXiv:1803.09256*, 2018.

[35] M. Li, Z. Zhang, K. Huang y T. Tan, "Estimating the number of people in crowded scenes by MID based foreground segmentation and head-shoulder detection," *2008 19th International Conference on Pattern Recognition*, 2008. DOI: 10.1109/icpr.2008.4761705.

[36] M. A. Hassan, I. Pardiansyah, A. S. Malik, I. Faye y W. Rasheed, "Enhanced people counting system based head-shoulder detection in dense crowd scenario," *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*, 2016. DOI: 10.1109/icias.2016.7824053.

[37] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection.," . *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, págs. 1137-1143, 1995. dirección: https://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.48.529.

[38] V. A. Sindagi, R. Yasarla y V. M. Patel, "JHU-CROWD++: Large-Scale Crowd Counting Dataset and A Benchmark Method," *Technical Report*, 2020.