ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

TEMA: IDENTIFICACIÓN EN TIEMPO REAL DE PERSONAS EN POSESIÓN DE PEQUEÑAS ARMAS DENTRO DE UN AMBIENTE VIDEO-VIGILADO

AUTOR: AGUILAR CABRERA, EDISON JACINTO

DIRECTOR: ING. SILVA TAPIA, RODRIGO.

SANGOLQUÍ

2018

CERTIFICADO DEL DIRECTOR

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que el trabajo de titulación ***"IDENTIFICACIÓN EN TIEMPO REAL DE PERSONAS EN POSESIÓN DE PEQUEÑAS ARMAS DENTRO DE UN AMBIENTE VIDEO-VIGILADO"*** realizado por el señor ***EDISON JACINTO AGUILAR CABRERA*** ha sido revisado en su totalidad y analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecido por la Universidad de las Fuerzas Armadas ESPE, por lo tanto me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 20 de Noviembre del 2018

ING. RODRIGO SILVA

DIRECTOR

AUTORÍA DE RESPONSABILIDAD

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORÍA DE RESPONSABILIDAD

Yo, EDISON JACINTO AGUILAR CABRERA, con cédula de ciudadanía Nº 070393262-4, declaro que este trabajo de titulación "IDENTIFICACIÓN EN TIEMPO REAL DE PERSONAS EN POSESIÓN DE PEQUEÑAS ARMAS DENTRO DE UN AMBIENTE VIDEO-VIGILADO" ha sido desarrollado considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas.

Consecuentemente declaro que este trabajo es de mi autoría, en virtud de ello me declaro responsable del contenido, veracidad y alcance de la investigación mencionada.

Sangolquí, 20 de Noviembre del 2018

EDISON JACINTO AGUILAR CABRERA
C.C. 070393262-4

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

Yo, EDISON JACINTO AGUILAR CABRERA, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación ***"IDENTIFICACIÓN EN TIEMPO REAL DE PERSONAS EN POSESIÓN DE PEQUEÑAS ARMAS DENTRO DE UN AMBIENTE VIDEO-VIGILADO",*** en el repositorio institucional, cuyo contenido, ideas y criterios son de mi autoría y responsabilidad.

Sangolquí, 20 de Noviembre del 2018

EDISON JACINTO AGUILAR CABRERA

C.C. 070393262-4

DEDICATORIA

A mis padres, a mi hermana y familiares en general que de una u otra manera a todos quienes como símbolo de amor y gratitud por todo lo brindado, su apoyo incondicional y su perseverancia hacia mí, permitieron el cumplimiento de una etapa más de mi vida sabiéndome dirigir por el sendero del bien. A mis maestros, amigos y a toda la Familia de este prestigioso establecimiento en reconocimiento a su valiosa guía y amistad brindada durante todo este proceso.

AGRADECIMIENTO

A Dios primeramente por permitirme día a día seguir adelante y por la maravillosa capacidad para ser una buena persona.

A mis padres por brindarme todo lo necesario para cumplir mis objetivos, a más de inculcarme las mejores enseñanzas y valores desde la infancia, Por ese apoyo constante e incondicional que me han ayudado a superarme continuamente, por sus consejos y los ánimos que me permiten siempre buscar lo mejor para mi futuro.

A mi familia entera que de alguna u otra manera me supieron guiar en este camino y siendo un apoyo durante toda mi vida han formado la persona que soy.

A mis maestros y amigos en general por toda su ayuda y conocimientos que supieron impartirme durante mi vida académica y que mediante estos pude desarrollarme tanto intelectualmente como personalmente.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

ÍNDICE DE ANEXOS

RESUMEN

La identificación de objetos tiene mucho interés debido al amplio espectro de aplicaciones tales como alertas de seguridad en entornos de videovigilancia relacionados con el Área de Inteligencia Artificial que involucrada a todas las nuevas tendencias y tecnologías. Este proyecto tiene como propósito realizar la identificación de video en tiempo real de personas en posesión de armas como pistolas y cuchillos para analizar múltiples eventos de reconocimiento, los sistemas de reconocimiento visual son capaces de clasificar con precisión las imágenes en muchas categorías para apoyar los avances tecnológicos a prevenir posibles eventos peligrosos, utilizando diferentes técnicas, como las cascadas HAAR a través de un algoritmo de sistema detector que funciona y es implementado con una cámara de video para obtener las imágenes de entrada, analizando estas imágenes con algoritmos basados en visión artificial y programados en una tarjeta embebida de bajo costo (Raspberry Pi 3 MODEL B +). El detector será diseñado utilizando estas herramientas programables con funciones de estructura basadas en un *framework*, utilizando herramientas de Aprendizaje automático para obtener características de objetos con bibliotecas OpenCV que se ejecutan sobre software Python bajo un sistema de código abierto Linux.

PALABRAS CLAVES:

- **PROCESAMIENTO DE IMÁGENES CON OPENCV**
- **CLASIFICADORES HAAR, HOG-SVM, LBP.**
- **APRENDIZAJE DE MAQUINA**
- **CENSADO DE IMAGEN EN RASPBERRY PI**
- **PROGRAMACIÓN EN PYTHON**

ABSTRACT

Object recognition has wide field of application and has a great interest in application such as security alerts in video surveillance environments concerned to Artificial Intelligence Area involved in all new tendencies and technologies. The purpose of this investigation project is to make real-time video recognition of people in possession of weapons as guns and knives for analyzing multiple events of recognition, visual recognition systems capable to accurately classify images into many categories will support technological advances preventing possible dangerous events, using different techniques like HAAR cascades through a detector system algorithm working and implemented with a video camera to obtain input images, analyzing these images with artificial vision-based algorithms programmed in a low-cost embedded card (Raspberry Pi 3 MODEL B+). Detector will be designed using programmable tools with structure functions based on a framework design using Machine Learning tools to obtain object characteristics with OpenCV libraries running on Python software under Linux open source System.

KEY WORDS:

- **OPENCV IMAGE PROCESSING**
- **HAAR, HOG-SVM, LBP, CLASSIFIERS**
- **MACHINE LEARNING**
- **RASPBERRY PI IMAGE SENSING**
- **PYTHON PROGRAMMING**

CAPÍTULO I

1. Introducción del proyecto de investigación

1.1. Resumen del Proyecto

La finalidad del presente proyecto es la de desarrollar un prototipo de bajo costo para la identificación de personas en posesión de armas (pistolas, cuchillos) con la ayuda de una tarjeta embebida y una aplicación para la detección en tiempo real de eventos emergentes que pueden incluir desde intentos de agresión hasta robos o intentos de asesinato, por esto se propone emular un entorno video-vigilado que fácilmente pueda ser instalado en vehículos de servicio de transporte público, o bien en cualquier sistema video-vigilado como puede ser los sistemas de seguridad bancarios, o de seguridad pública como los implementados por el sistema de vigilancia integrado de seguridad Ecuatoriano (ECU911).

EL uso de tecnologías de entorno libre y de fácil acceso al usuario son también una de las motivaciones principales del proyecto, pues el desarrollo en tecnologías basadas en software libre como lo es Python, con ayuda de librerías de *Machine Learning* para el análisis de imágenes en tiempo real conocidas como OpenCV

Esta investigación se centra en la detección inteligente y en tiempo real de personas en posesión de armas, ya que como conocemos en nuestro país es restringido el llevar consigo algún tipo de arma sin el respectivo permiso a más de ser prohibido en la mayoría de los países y también se enfoca el análisis sobre estos dos objetos (pistola

y cuchillo) porque forman parte de las estadísticas como una de las principales armas más utilizadas para los actos delictivos.

Para concluir con el proyecto se desarrolla finalmente un modelo de prototipo que será puesto a prueba a fin de comparar y determinar la precisión del algoritmo y su funcionamiento en entornos video-vigilados.

1.2. Trabajos relacionados

Existen algunos trabajos relacionados en especial durante los últimos tiempos en los que se ha desarrollado la inteligencia artificial , el uso de computadoras de mayor capacidad y la optimización de tarjetas embebidas dedicadas al análisis de inteligencia artificial GPUs (*Graphics Processing Units*) y NPUs (*Neural Processing Units*) , aplicadas a sistemas de video-vigilancia se tienen estudios previos para el análisis de imágenes y videos previamente grabados analizando únicamente aspectos parciales y si bien se han desarrollado clasificadores de imágenes corriendo sobre varias plataformas como Java (Joyce, 2016), Matlab (Stephen, 2016), C# (Jitendra, 2012), Python (Jhon M, 2004), bases que se pueden obtener desde directorios web (developers, 2017) y conjuntamente con redes neuronales con clasificadores de imágenes que en su mayoría están implementados para una identificación general de objetos(personas, rostros, autos, partes del cuerpo humano) con un alto costo computacional debido a su tamaño de muestra, sin centrar su diseño en la identificación única de un objeto, para obtener un clasificador detector de armas e identificación de personas armadas, se realiza la implementación de un clasificador propio; como proyectos antecesores al presente tenemos entre los más destacados: “Automated Detection of Firearms and Knives in CCTV image”, reconocimiento de armas con el uso de descriptores visuales MPEG-7,

herramientas (INACT, INSTREET) (Micha Grega, 2012) y redes neuronales(*Background&Canny Edge Detection*) con resultados favorables y esperando los avances tecnológicos para continuar el trabajo con nuevos tipos de cámaras (PTZ y cámaras nocturnas) (Greja, 2016). “Weapon Detection In Surveillance Camera Images” reconocimiento de armas en videos de calidad estandard con método de sustracción de fondo (background substraction method), HOG-CNN (*HOG+Convolutional neural Network*) (Rohit Vajhala, 2016).“An automated Hybrid Approach to Detect Concealed Weapons Using Deep Learning”, deteccion de armas bajo la vestimenta usando rayos X, aplicando wavelets pre entrenados con SVM-CNN(*AlexNet*) en Matlab sobre windows 10 (Al-Shoukry, 2017). “Developing a Real-Time Gun Detection Classifier” deteccion de pistolas con algoritmos R-CNN y CNN(GoogleNet y Tensorbox) (Justin Lai, n.d.).“AWeD (Automatic Weapons Detection)”, Detección de armas con redes neuronales implementadas con matlab y sobre una FPGA (Skyler Alsever, 2017).“Detecting Guns Using Parametric Edge Matching”, Detección de pistolas con el uso de rayos X y tecnicas de procesamiento de imagenes(Edge detection& distance mapping) (Aaron Damashek, n.d.). “A Computer Vision based Framework for Visual Gun Detection using Harris Interest Point Detector” deteccion de armas usando técnicas de segmentación de color y el detector Harris sin contemplar cambios de iluminación ni de espacios (Rohit Kumar Tiwari, n.d.). los trabajos anteriormente descritos basan su ejecución en videos pregrabados únicamente y dado que existen pocos trabajos basados en detección de armas que sean públicos, o de los cuales no se tenga acceso a su información de entrenamiento y clasificadores utilizados, se desarrolla este sistema que busca poder ser ejecutado en tiempo real y en ambientes más estrictos además de ser realizad sobre software libre, y

asi no únicamente analizar una imagen, sino un conjunto completo de fotogramas que forman a un video en tiempo real para la detección y generar una respuesta automática al evento y ser capaz de identificar o no una amenaza(armas sin presencia humana se considera evento normal) y detectar estos eventos de forma más certera mediante la creación de un clasificador utilizando imágenes de bases de datos pública que corresponden a cualquier imagen descargada desde la web y un banco de imágenes propias de los objetos para detección, adquiridas mediante una cámara para los entrenamientos de los clasificadores.

Luego de analizar y tomando como referencia todos estos trabajos previos como punto de partida para el desarrollo del presente proyecto proporcionan información importante respecto a los aspectos que debemos seguir y que procedimientos cumplir para obtener un buen desempeño con resultados más favorables.

1.3. Justificación e Importancia

Los sistemas de videovigilancia en la actualidad se emplean para la recolección y análisis de datos relacionados con la seguridad pública o privada de las personas, así también como para aumentar la productividad de las personas en sitios de trabajo, pero más importante registrar las actividades delictivas que puedan surgir, varias empresas dedicadas al hardware y software para análisis y procesamiento de imágenes como lo es Nvidia han generado propuestas para el año 2020 como “THE AI CITY” buscando crear ciudades más inteligentes y seguras (Albany, August, 2017; Corp, 2017; Piyush Modi, 2017; Shanker Trivedi, 2017).

El presente proyecto pretende estar a la vanguardia de los avances tecnológicos busca con afán el prever posibles amenazas que atenten contra la seguridad, razón principal por la que se desarrollará este trabajo de investigación que pueda ser aplicado en cualquiera de los entornos principales como son: televigilancia, centros de control, sistemas de videovigilancia en interiores de locales comerciales u oficinas, sistemas de videovigilancia en buses y taxis de la región ecuatoriana (Martin, 2012; ANT, 2017).

La integración de dispositivos electrónicos, las técnicas de procesamiento digital de señales y los algoritmos de visión artificial están facilitando la creación de sistemas inteligentes que contribuyen con el bienestar de las personas (Howse; Francisco E). Los datos de vigilancia se generan a través de la cámara, y luego estos datos se transmiten al computador para ser procesados. El procesamiento debe hacerse en tiempo real y si se detecta alguna anomalía, el sistema podría generar una alerta.

Para la implementación de los detectores se probaran al menos dos técnicas de detección tales como HOG(*Histogram of Oriented Gradients*) con algoritmo clasificador SVM(*Support Vector Machine*) y también HAAR(*feature-based Cascade Classifiers for object detection*) con algoritmo clasificador Adaboost(*Adaptive Boosting*), que correctamente entrenados de manera iterativa, permite detectar en tiempo real objetos que aparecen en escenas de video capturadas mediante cámaras fijas ubicadas en escenarios controlados (Reyjavik, 2015; Cyganek, 2013).

1.4. Alcance del Proyecto

Para la fase de investigación se utilizará distintos métodos para el procesamiento, reconocimiento y detección en sistemas que realicen el seguimiento e identificación de objetos. El diseño del framework para el sistema implica un diseño de la lógica que compone al sistema, su funcionalidad explicada de manera ordenada para simplificar y organizar su desarrollo en forma secuencial, donde se pueden incluir varias etapas como planeación, modelado, construcción y despliegue del proyecto.

Para el desarrollo del algoritmo sobre Python (versiones disponibles 2.7/3.5) y con el empleo de las librerías de OPENCV (versión 3), partiendo de una base de datos formada por varias imágenes de muestra se crea y entrena varios clasificadores HAAR cuyo función específica es la de detectar armas, para conformar el banco de imágenes se tomará un conjunto de archivos de imagen(100-2000 imágenes.bmp menores a 640x480 pixeles), las cuales tendrán un tratamiento para ajustarse a los distintos escenarios (luminosidad, contraste, nitidez y dimensión) . Es un reto en el que se emplean distintas herramientas computacionales como es OPENCV para el desarrollo de algoritmos de tratamiento y análisis de imágenes, que posteriormente ejecutan los clasificadores para realizar las tareas de detección con bajo costo computacional al ser implementados en la tarjeta embebida.

Para las pruebas del prototipo, mediante herramientas de evaluación como lo es la matriz de confusión, que contempla los eventos reales y de predicción que abarcan cuatro subgrupos de análisis $(VP \text{ verdadero positivo}, FP \text{ falso positivo}, FN \text{ falso negativo}, VN \text{ verdadero negativo})$ donde $Presicion = \frac{VP}{VP+FP}$, e incluso con la evaluación de otras

herramientas se pretende así determinar las medidas de precisión o exactitud del sistema, para lo cual se emulará un sistema de videovigilancia con la cámara fija acoplada a la tarjeta embebida en el cual se detectará en tiempo real la presencia de personas con posesión de armas emitiendo un tipo de alerta.

La implementación contempla factores como la ubicación de la cámara, el tipo de objetos con los que se construirá la base de datos, así como también los niveles de intensidad de luz o calidad de imagen, para mejorar la eficiencia del sistema de identificación propuesto.

1.5. Objetivos

1.5.1. General

Realizar la identificación en tiempo real de personas en posesión de armas (cuchillos o pistolas) dentro de un ambiente video-vigilado.

1.5.2. Específicos

- Investigar el contexto teórico-práctico del procesamiento de imágenes para la detección e identificación de objetos.
- Diseñar el modelo del framework para el sistema
- Desarrollar los algoritmos de detección y programarlos en la tarjeta.
- Evaluar el desempeño del prototipo emulando un sistema de videovigilancia.

CAPÍTULO II

2. MARCO TEÓRICO

Como parte fundamental de este capítulo se tiene la descripción de conceptos sobre el análisis de imágenes, que trabajan juntamente con *Machine Learning*, librerías de OpenCV sobre Python, clasificadores y algoritmos para la captura y análisis en tiempo real de video sobre una tarjeta embebida.

2.1. Introducción

En la actualidad en casi todos los países, los sistemas integrados de seguridad compuestos por cámaras de vigilancia instalados en distintos ambientes para mejorar la seguridad social y en algunos casos realizar un análisis de los eventos suscitados como es el caso de nuestro país con el sistema público de videovigilancia ECU 911, aunque este sistema esta monitoreado por un personal capacitado para la seguridad siguen siendo una herramienta pasiva ya que de aquí parte la necesidad de implementar ayudas tecnológicas que sean capaces de identificar un tipo de suceso especifico.

El empleo algoritmos para detección de objetos en imágenes de sistemas videos vigilados presenta un gran avance ya que hasta el momento el empleo de las cámaras de seguridad que no tienen vigilancia constante de un observador solo servía para identificar un evento luego de que este sucediera, y lo que se busca es tener una respuesta en tiempo real del suceso a fin de optimizar y dar aumento al perfil de seguridad

social. Con el análisis predictivo y empleo de varias herramientas como clasificadores de objetos mejoran los sistemas de monitoreo actuales.

Al desarrollar este sistema sobre un producto basado en software libre damos un camino a nuevas producciones de sistemas inteligentes que pueden ser analizados posteriormente, al basar el proyecto en algoritmos de reconocimiento y análisis de imagen sobre Python, usando las librerías de *Machine Learning* sobre OpenCV (Open source Computer visión) para el entrenamiento y reconocimiento.

2.2. Imagen

Una imagen es considerada una cadena, sucesión o matriz de pixeles en las que cada elemento dentro de ellas representa las características de la imagen, los pixeles en una imagen análoga no representan únicamente un punto en la imagen, sino una región rectangular mientras que en imágenes digitales los pixeles representan el brillo en un determinado punto, las distintas combinaciones y valores que tienen estos pixeles son los que conforman a una imagen, estos valores pueden ir desde 0 (negro) hasta 255 (blanco) y el valor que cada pixel puede tomar depende de la profundidad de color de la imagen.

Así mismo una imagen de 8 bits de profundidad de color posee 256 escalas de grises o tonalidades, mientras que una imagen "*true colour*" está compuesta por 24 bits = 8x8x8 que corresponden a 256x256x256 completando un aproximado de 16 millones de colores que componen a la imagen.

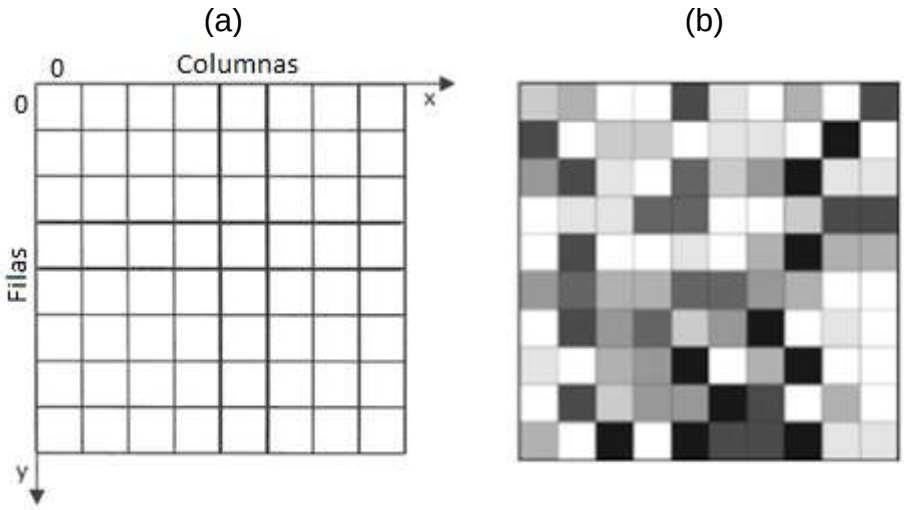

Figura 1. a) Representación de una matriz de pixeles b) representación de pixeles en tonos de color
Fuente: (Jahne, 1997)

Un archivo de imagen es una compresión de dos elementos conocidos como encabezado de archivo y conjunto de datos; el *header* que contiene la información esencial del archivo como son: tamaño de archivo, puntos de inicio y fin de archivo, tamaño de matriz dividido en filas y columnas (Figura 1a)), y niveles de escalas de grises o color RGB (*Red, Green, Blue*) como se aprecia en la figura 1 b), todos estos valores son necesarios para posteriormente realizar un correcto análisis y procesamiento de dichas imágenes.

El procesamiento de imágenes implica una relación directa con la detección de objetos ya que depende del tipo, tamaño, calidad de imagen entre otros factores influyen en la detección por el hecho de que es la muestra a ser analizada , si el sistema tiene en su entrada una muestra deficiente, la detección puede ser pobre o en algunos de los casos sin eventos existentes, por ello aquí radica la importancia de adaptar un sistema que tenga una entrada adecuada conjuntamente con el tratamiento de la imagen. (Oakley, 2003)

Como los colores de cada pixel están representados en tres diferentes que son: rojo, verde y azul (RGB) estos se pueden modificar mediante la aplicación de filtros, alterar sus valores y dando como resultado una nueva imagen final, el conjunto de estas variaciones y tratamientos de imagen nos permiten en un futuro realizar distintos tipos de aplicaciones y detecciones.

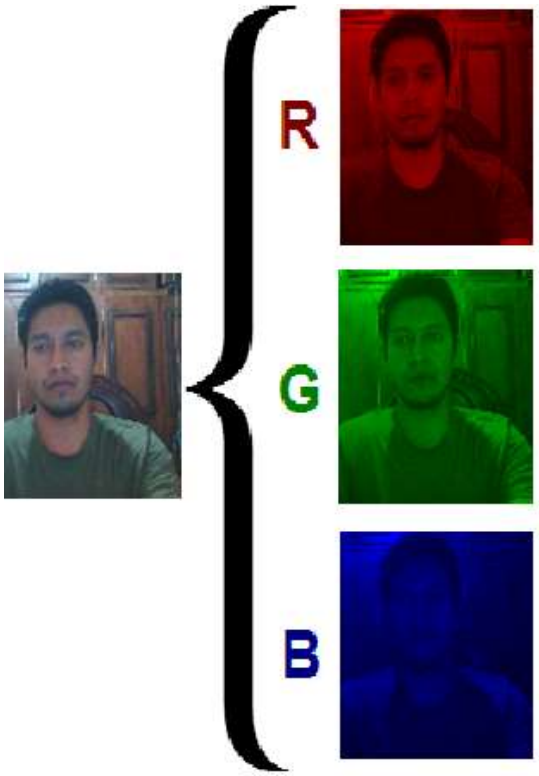

***Figura 2*.** Separación de canales RGB

La separación de canales RGB observado en la figura 2 es una gran ayuda al momento de analizar imágenes, así mismo el convertir la imagen a escala de grises y otros tipos de filtrado mejoran o empeoran la detección de un objeto oscureciéndolo o resaltándolo, por esta razón debemos analizar en entorno, el tipo de objeto y según esto adecuar el procesamiento para obtener mejores respuestas de detección y análisis de imágenes.

Algunas veces la imagen de entrada puede ser deficiente, puede influir el entorno o bien puede depender de la configuración del dispositivo de adquisición de imágenes, y dependerá del tratamiento para cada caso que se le dé a la imagen para que el sistema pueda realizar un correcto procesamiento y análisis de la imagen. (Commons, 2011)

2.3. Video

Para el procesamiento de video se debe realizar primero el procesamiento de imágenes, como ya sabemos un video es una sucesión de imágenes a una determinada frecuencia en un intervalo de tiempo generalmente tomado en segundos llamado fotogramas por segundo (FPS), dichas imágenes sucesivas conforman a un video y para su procesamiento se opera directamente sobre estas imágenes con varias técnicas y librerías propias de los lenguajes de programación, como esta operación se realiza directamente sobre las imágenes, basta con encontrar la ubicación de cada pixel que deseamos tratar o analizar, este es el principio básico del procesamiento de imágenes que posteriormente es empleado al reconocimiento de varios patrones y objetos dando resultados muy sorprendentes. Los seres humanos tienen una visión más amplia de los patrones existentes en la naturaleza y los sistemas de visión por computadora incluyen algoritmos avanzados que buscan emular al sistema humano, por esta razón se vienen desarrollando sistemas de reconocimiento facial, detección de objetos y demás, que mediante programación y distintos métodos con el uso de cámaras o videos pretenden mejorar la eficiencia de los sistemas para que tengan la una eficiencia similar a la humana.

Figura 3. Framework general de sistemas con video
Fuente: (Thomas B, 2012)

El diseño de framework de la figura 3 es válido para cualquier categoría de procesamiento de imágenes/videos puesto que todo parte de la adquisición de imágenes por parte de una cámara, para luego ser pre-procesada a términos de reconocimiento computacional más efectivo, adquirir sus características deseadas, convertir estos valores a términos que el computador entiende y luego determinar un factor de relación o coincidencia para luego elaborar una selección de clasificación. Para realizar un análisis computacional se necesita que el video resultante cumpla ciertas condiciones, por ejemplo para que se considere una correcta apreciación en tiempo real se necesita que cuente con un número de frames por segundo adecuado para que el sistema pueda percibir los movimientos y realizar el tracking de cualquier objeto que se desee, pues depende directamente de la calidad de este video, así mismo el costo computacional es muy alto al tener un video con mayor resolución y FPS, por esto buscar un equilibrio es lo más adecuado a la hora de realizar proyectos de Visión Computacional Avanzada, también se pueden incluir en esta etapa de procesamiento el uso de distintos códecs, filtros, herramientas y configuraciones para el procesamiento del video en tiempo real. (Tinku Acharya, 2005)

2.4. Detección de Objetos

En la detección de objetos se analizan las distintas imágenes para la extracción de características, y mediante la construcción de vectores tomados de una base de datos se realiza el aprendizaje automático para dar forma al clasificador, todos estos pasos llevan a crear el sistema de reconocimiento y detección de objetos indicado en la figura 4.

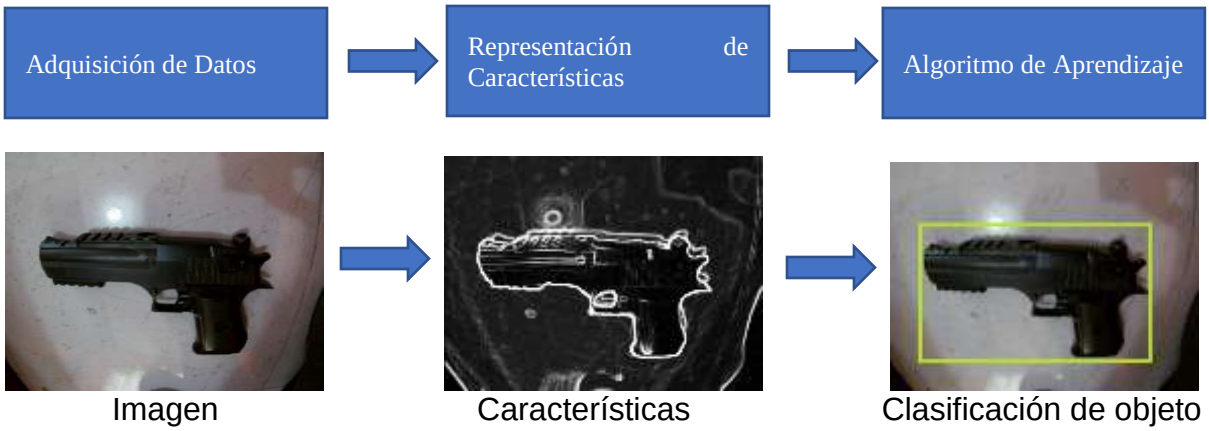

Figura 4. Proceso para la Detección de Objetos

Si bien para que exista el reconocimiento de un objeto, primeramente, se debe realizar su detección, al analizar una imagen cualquiera el sistema no sabe que objeto se va a reconocer, por lo que se debe realizar previamente el entrenamiento del clasificador, por ejemplo si deseamos identificar la presencia humana, el sistema debe ser capaz de discernir entre los objetos de la imagen e identificar o detectar la presencia humana, mediante el uso de varios algoritmos de reconocimiento y comparación el sistema finalmente será capaz de reconocer el objeto, con esta conclusión se pueden realizar así mismo procesos sucesivos de análisis que permitan ser más exhaustivos a la hora de detectar un objeto mejorando las capacidades del sistema. Así mismo el sistema de reconocimiento puede decirnos si el objeto identificado es una silla un zapato o distintamente del objeto para el cual el sistema fue entrenado y así realizar análisis

diferentes que son considerados retos en los sistemas de visión por computadora, ya que los seres humanos tienen la capacidad de reconocer distintos objetos y elegir uno en cual realizar un enfoque, los sistemas computacionales pueden realizar estas tareas con deficiencias, y con algoritmos más avanzados de programación y visión artificial para lograr extraer las características de las imágenes presentadas. (Gabriel, 2018)

2.4.1.1. Descriptores

En los algoritmos de detección se entiende como descriptor a la extracción de características de imagen desde los componentes propios que describen a la imagen, por ejemplo si se quiere analizar un rostro se extraen las características de la cara de toda región facial detectada, conjuntamente con OpenCV que es una librería de Python, proporciona una variedad de algoritmos para la detección de puntos de interés y descripción de características de imagen, para esto existen varios algoritmos como HOG, SVM, Adaboost que forman parte de los módulos de OpenCV sobre Python que se analizarán más adelante.

Como toda esta información de características de imágenes necesita un lugar de almacenamiento se crean los vectores de descriptores, el tamaño de estos vectores varía de acuerdo a las imágenes y de esta manera tenemos que los descriptores de las imágenes analizadas se agregan a un vector, que según las iteraciones que realicemos tendrá una respuesta de búsqueda, uno de los factores importantes es el gran costo computacional que este proceso conlleva, por esta razón otros ámbitos como la resolución, y cantidad de imágenes intervienen directamente ya que si se sobrecarga el tamaño de vector, este podría resultar escaso y no muy eficiente para realizar una clasificación. (Joseph, 2015)

2.4.1.2. Clasificadores

El clasificador se construye con el uso de algoritmos basados en *Machine Learning*, la generación de un clasificador reduce el costo computacional, reduce tiempos de análisis e incrementa el desempeño en la detección de objetos, con la ayuda de algoritmos como SVM en el caso de los que usan características de tipo HOG o Ada-Boost en el caso de detectores de tipo HAAR para el aprendizaje, se usan para aumentar esta detección en un algoritmo más simple, convirtiendo así a un clasificador débil en uno más fuerte. El uso de clasificadores basados en HAAR permiten una mejor extracción de las características formando algoritmos de detección más potentes y más todavía si estos están desarrollados en cascada, ya que se realiza un procesamiento continuo de la imagen a través de una cadena de clasificación como puede observarse en la figura 5, y si es aprobado por un primer clasificador, pasa a ser evaluado por un segundo y así sucesivamente mejorando la detección y eliminando así a los falsos positivos.

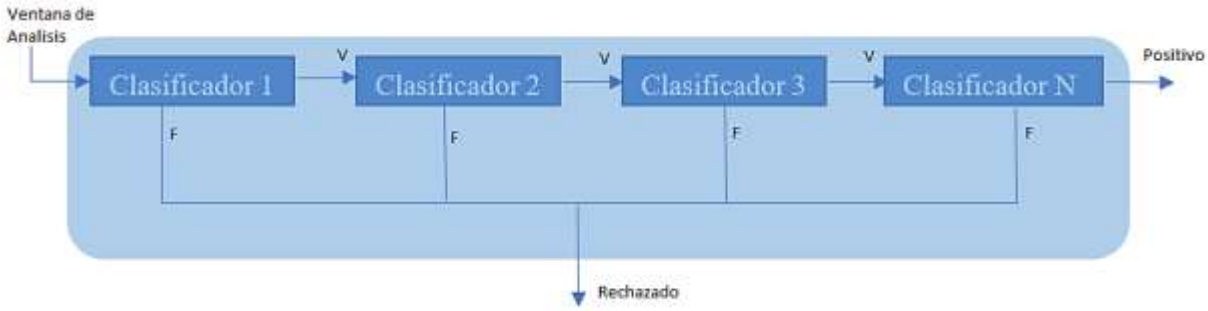

Figura 5. Funcionamiento de clasificadores en cascada
Fuente: (Juliano, 2015)

La extracción de características de imagen basadas en clasificadores HAAR están compuestas por un amplio diccionario de imágenes tanto positivas como negativas del objeto que vamos a entrenar al sistema para que reconozca sus propiedades, pudiendo reconocer sus cualidades en un video en tiempo real. (Davic, 2011)

2.5. Detectores Holísticos:

2.5.1. Detectores Tipo HAAR cascade:

Los detectores de tipo HAAR presentan características rectangulares de imagen para el reconocimiento de objetos, el análisis de dichas imágenes y las coincidencias encontradas dentro de la misma son las que generan las áreas o parámetros como se muestra en la figura 6 los patrones básicos de las características de tipo HAAR. Las ares que están representadas de color negro equivalen a las áreas negativas y las de color blanco corresponden a las positivas, como se muestra en la figura 6 , primero los valores de cada pixel dentro de las áreas negras se suman para luego sumar los valores que se encuentran dentro de las áreas blancas y con este valor total resultante de las áreas blancas o positivas se resta con el valor total de las áreas negras y con el resultado obtenido se analiza para clasificar así las subregiones de cada imagen por esta razón es la que se pueden crear clasificadores más fuertes en donde son capaces de detectar objetos con variaciones de iluminación , color o escala.

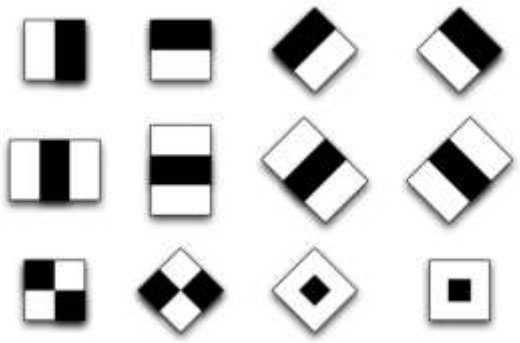

Figura 6. Patrones básicos de características HAAR
Fuente: (Bart de Decker, 2013)

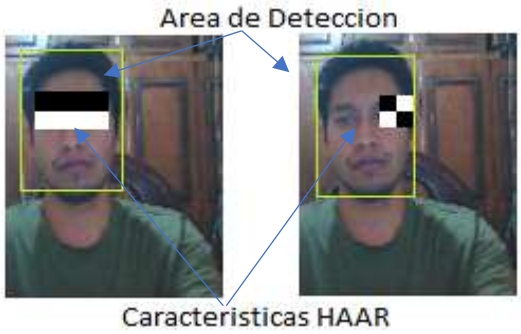

Figura 7. Ejemplo de área para Detección HAAR
Fuente: (Bart de Decker, 2013)

2.5.2. Detectores Tipo HOG-SVM:

Histogram of oriented Gradients o detectores de tipo HOG, han sido satisfactoriamente aplicados en muchos proyectos de visión por computadora, en especial en la clasificación de personas, la idea principal detrás del uso de características HOG es que las formas de los objetos y las apariencias que estos presenten dentro de una imagen se puedan describir mediante una distribución de las direcciones mediante vectores de cada borde de objeto. Estos bordes están divididos dentro de secciones en la imagen y el conjunto de todos ellos conforma a un descriptor, como se puede apreciar en la figura 8 se tiene la conversión de una imagen con su distribución de direcciones y como estos predominan en el contorno de los objetos que podrían ser identificados.

(a) (b)

Figura 8. a) imagen de entrada b)imagen con Gradientes de vector HOG
Fuente: (Beyeler, 2017)

Con el empleo de *Support vector Machines* conjuntamente con el descriptor HOG, Existen Hiperplanos infinitos que pueden estar presentes entre clases, incluso entre puntos dentro de una imagen, en la figura 9 a) se observa como tres hiperplanos $h1$, $h2$ y $h3$ logran separar los datos distintos (Blanco y Verde)correctamente en dos clases, por esta razón el poder de generalización es muy bueno y para tomarse en cuenta en procesos de separación de clases.

Al analizar la figura 9 b) se observa los márgenes $m1, m2$, que guarda la separación en la clasificación de clases de datos, y así poder determinar cuál de estos planos es más válido, ya que entre mayor sea el margen mayor será la capacidad de separar las clases de una manera lineal y sencilla.

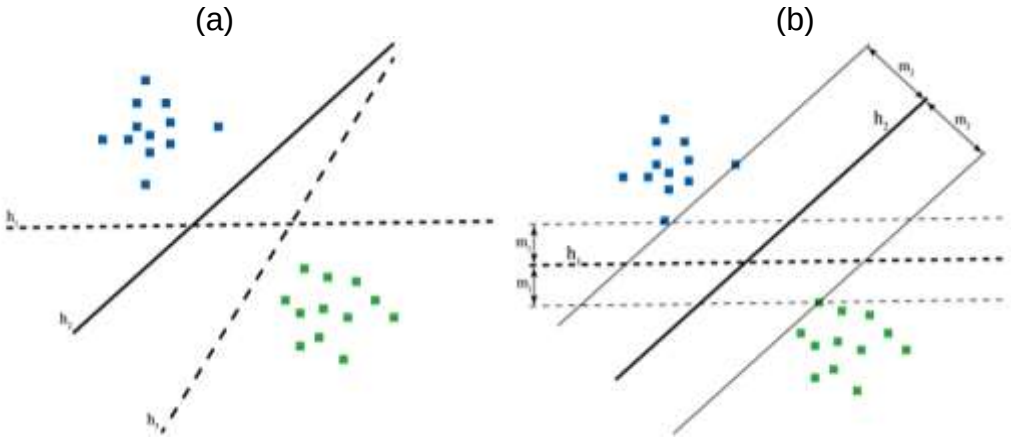

Figura 9. a) Clasificaciones posibles b) Hiperplanos con su margen de separación
Fuente: (Juliano, 2015)

2.5.3. Detectores tipo LBP-Cascade:

Las características basadas en LBP o *Local Binary Patterns* se han optimizado muy bien en varias aplicaciones, entre las cuales están la clasificación de objetos por segmentación y texturas, procesos de regeneración e inspección de imágenes, entre otros. Los detectores con características LBP realizan una comparativa o etiqueta de los

pixeles dentro de una imagen detectando un umbral de vecindad, este umbral se da cada 3x3 pixeles en los que el pixel central tiene un valor resultante y el cual es considerado como un numero binario. Como se puede observar en la figura 10 se muestra un ejemplo de cálculo de LBP.

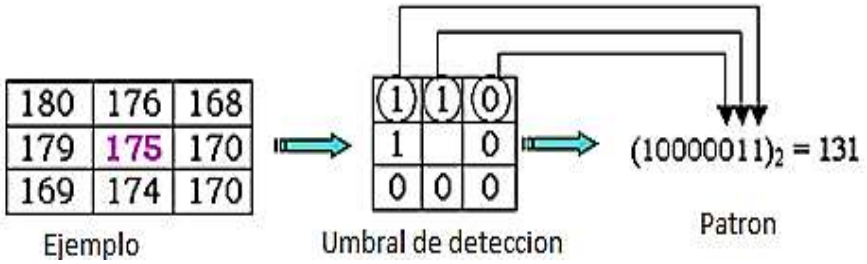

Figura 10. Ejemplo del cálculo de LBP
Fuente: (Jo, 2008)

Según Jo Chang-yeon, con el empleo de representaciones faciales basadas en LBP con más propuestas de análisis en las que se consideró a la cara como un conjunto de micro patrones reconocibles en LBP y así analizar mejor la información, al tomar en cuenta la forma que componen las caras, se realizó una división de las imágenes que la componen en regiones más pequeñas o superpuestas $R_0, R_1, \ldots\ldots R_M$ generando el histograma como se puede apreciar en la figura 11, así cada histograma obtenido de cada región que compone a la imagen se concatena en uno solo con características mejoradas que está definido de la siguiente manera:

$$H_{i,j} = \sum_{x,y} I(f_l(x,y) = i) I\left((x,y) \in R_j\right)$$

Ecuación 1. Fórmula para Histogramas LBP
Fuente: (Jo, 2008)

Donde $i = 0, \ldots, L-1, j = 0, \ldots, M-1$ y la característica del histograma extraído describe la textura loca y global de la forma del rostro de la imagen.

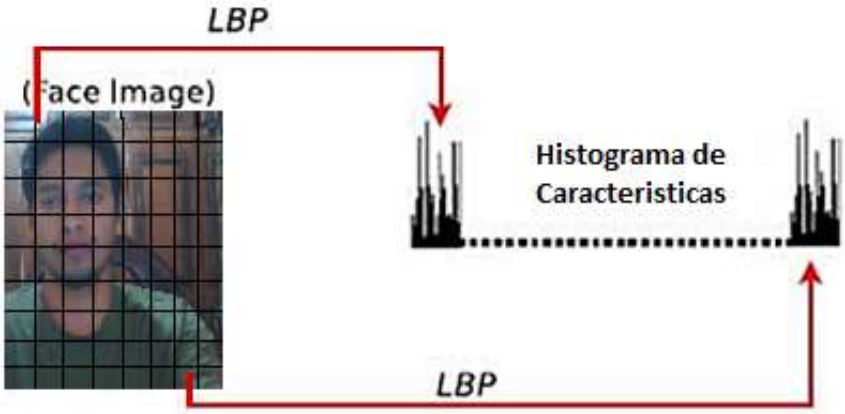

Figura 11. Representación facial basada en LBP
Fuente: (Jo, 2008)

2.6. Detectores no Holísticos

Los detectores no Holísticos están basados en dos modelos principales conocidos que son modelo basado en partes con deformación y sin deformación, el modelo basado en partes sin deformación , presenta en el análisis de la imagen una ventana externa que contiene a todo el objeto, dentro de ella se encuentran varias ventanas internas productos de la detección que pueden constituir partes de análisis para determinar un reconocimiento, es decir que la suma de las partes internas de un reconocimiento constituyen al tamaño final que tendrá el objeto reconocido.

Para el modelo basado en partes deformables, al momento de realizar la detección, las partes que lo conforman pueden estar en posiciones distintas, igual para cada parte se entrena y conforma a la estructura de análisis del objeto final y constituye a la región de análisis, solo que únicamente pueden tener posiciones variables y no seguir un patrón único de reconocimiento, aquí los objetos q nos interesan tomamos en cuenta cualquier parte del match pues no están atados a una posición especifica.

Para el análisis y procesamiento se pueden juntar varios descriptores en uno solo para entrenar a los modelos, de esto depende la eficiencia que tiene el sistema, o también combinando los clasificadores para crear un algoritmo más potente y así reducimos los falsos positivos. En relación con los detectores Holísticos, aquí se presentan problemas en la relación de aspecto, ya que cambia bastante el ancho y alto del objeto analizado en relación con otro igual encontrado en la misma imagen. Para ambos casos se obtiene la frontera de la imagen obteniendo buenos resultados en ambos casos e igualmente trabajando con los descriptores de apariencia HOG y descripción de las deformaciones de las partes, incluyendo aprendizaje del modelo SVM con variables latentes que son las características más determinantes de una detección pues son las respuestas obvias de lo que se espera en la detección, por ejemplo todos sabemos que los pies se encuentran casi siempre junto al piso, esto sería una variable latente. (Lopez, 2017)

2.6.1. Redes Neuronales

Una red Neuronal artificial es un modelo computacional que busca principalmente desarrollar un modelo matemático en representación y semejanza a la estructura del cerebro humano, simulando poseer su sistema nervioso y así lograr extraer características y conocimientos dentro de ella. Una red neuronal tiene un elemento básico que la conforma y son las neuronas, que se encuentran agrupadas en millones de redes o capas de procesadores conocidos como neuronas artificiales, estas pueden o no estar relacionadas entre sí, lo que conlleva a que su información viaje por múltiples caminos y así poder generar procesos predictivos de búsqueda y deducción de información.

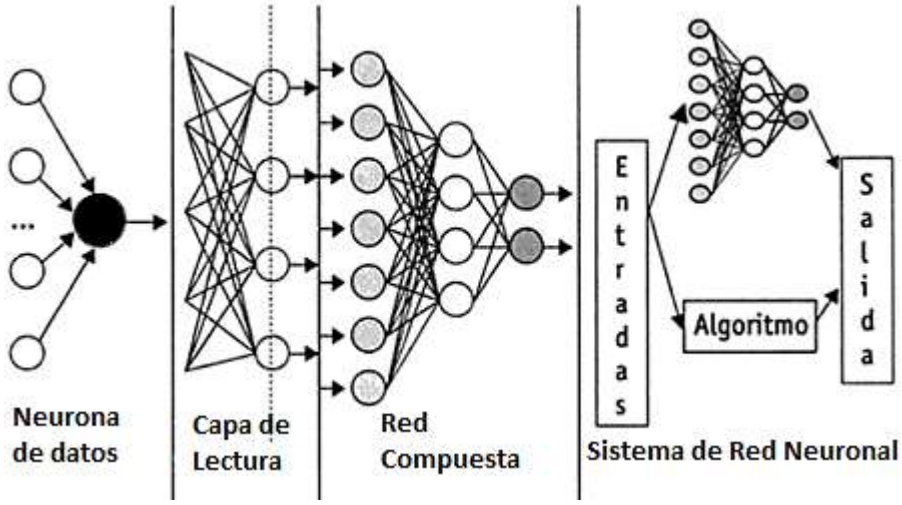

Figura 12. Estructura jerárquica de Sistemas de Red Neuronal
(Florez, 2008)

Siguiendo esta estructura es por qué las redes neuronales son consideradas modelos de cálculo compuestos por algoritmos operando eficientemente en paralelo para poder desarrollar tareas cognitivas, cálculos avanzados, reconocimiento de objetos, aprendizaje de patrones y varias operaciones para análisis de otras funciones. (Florez, 2008)

2.7. Evolución de la detección de Objetos

La detección de objetos es y será una de las áreas de la visión por computadora que más rápido se ha desarrollado, desde sus inicios entre los años 50’s y 60’s buscando coincidencias y relaciones para plantillas se desarrollaron para formar los primeros sistemas de reconocimiento de patrones de objetos principalmente en dos dimensiones, este estudio fue aplicado con más enfoque al reconocimiento de caracteres, huellas dactilares y también en imágenes microscópicas. Los primeros sistemas de detección estaban basados en reconocimientos principalmente de líneas de dibujo, u objetos curvos

y rectos planos, para posteriormente analizar cilindros y cubos, llegando al análisis de objetos en 3D.

Con la llegada de la era geométrica, se estudió distintas formas y objetos que presentaban mayores dificultades, llegando hasta la esfera alrededor de los años 70 donde se enlistó a la descripción de objeto por clases, en las que se estudió piezas ya manufacturadas y con más detalle, para lo que utilizaron capas de superficie poligonal conocidas como SAR (*Synthetic Aperture Radar Imagery*).

Luego con el uso de más técnicas de procesamiento se desarrolló los Histogramas Orientados a Gradientes (HOG) y el algoritmo de clasificación de Máquinas de Soporte Vectorial (SVM) y cuya estructura principal se basa en la subdivisión de la imagen en zonas más pequeñas para su reconocimiento obteniendo mejores resultados, así mismo con esto se pudo crear ya clasificadores en cascada más potentes como HAAR que son de los más utilizados en la actualidad, ya que sus aplicaciones son muy diversas y se puede realizar fácilmente su entrenamiento.

Así mismo con la evolución de la inteligencia artificial, el uso de tarjetas embebidas, la futura implementación de tarjetas NPU en varios dispositivos integrados lleva al análisis y desarrollo de nuevas tecnologías, a más de utilizar clasificadores en cascada, también se utilizan redes neuronales convolucionales (CNN) que son más optimizadas para el uso de la inteligencia artificial y proyecto de desarrollo que van de la mano con el internet de las cosas (IoT). Todos estos sistemas aplicados conjuntamente logran mejores resultados a la hora de realizar proyectos de visión por computadora. (Ponce, 2006)

Para evaluar la calidad de la predicción del modelo entrenado previamente se utilizará la matriz de confusión, este modelo de análisis clasifica los resultados por Reales o de Predicción, en los que las filas representan a los valores de predicción y las columnas a los valores reales de cada evento, sean estos considerados como positivos o negativos.

Tabla 1.
Matriz de confusión

MATRIZ DE CONFUSION OBJETO		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	VP	FN
	Negativos	FP	VN

Fuente: (Torres, 2018)

2.8. Evolución de los detectores:

$\boldsymbol{VP}$: Es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.

$\boldsymbol{VN}$: Es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.

$\boldsymbol{FN}$: Es la cantidad de positivos que fueron clasificados incorrectamente como negativos.

$\boldsymbol{FP}$: Es la cantidad de negativos que fueron clasificados incorrectamente como positivos.

Precisión: Indica un valor cuando se predicen eventos positivos, al multiplicarlo por 100 indica que porcentaje se clasifico correctamente.

$$\boldsymbol{Presicion = \frac{VP}{VP + FP}}$$

Ecuación 2. Fórmula de cálculo de Precisión
Fuente: (Torres, 2018)

TVP: tasa de verdaderos positivos, es también conocida como sensibilidad o exhaustividad, e indica cuando el evento es positivo en que porcentaje logra clasificar.

$$TVP = \frac{VP}{Total\ Positivos}$$

Ecuación 3. Fórmula de cálculo de TVP
Fuente: (Torres, 2018)

2.9. Herramientas de desarrollo

2.9.1. OpenCV sobre Python

Al ser una biblioteca de un muy buen desempeño y al tener como base su desarrollo previo en lenguajes de programación como C y C++, corriendo sobre Python logra grandes prestaciones, al trabajar sobre software libre, hace que su empleo se expanda por todo el mundo, siendo utilizado en muchos proyectos no solo para la seguridad electrónica, sino también en áreas como la medicina, la industria, y para mejorar el control y eficiencia de personal en otros sectores. OpenCV mediante el entrenamiento y uso del *Machine Learning* con algoritmos de detección ejecuta distintos tipos de tareas llamados clasificadores, estos clasificadores son creados partiendo de una muestra o base de datos para así determinar las coincidencias que pueden ir desde objetos, personas, o en este caso particular aplicado a la detección de armas.

2.9.2. Tarjetas embebidas

En las tarjetas embebidas, todos los componentes electrónicos que la componen deben estar unidos en una sola placa impresa o PCB (*Printed Circuit Board*), una de las primeras tarjetas embebidas populares fueron las FPGA, que ya contaban con la estructura básicas de una tarjeta embebida como era un procesador central (CPU),

sistemas de conexiones internas por buses de datos, memoria interna para el sistema y varios canales de entrada y salida. La evolución de estas tarjetas de Hardware embebido han logrado que la competencia en cuanto a desarrollo se haya incrementado, y es aquí donde empresas dedicadas al desarrollo tecnológico de *Hardware* y *Software* como Raspberry Pi Foundation, Nvidia, ASUS, INTEL y otras más marcaron nuevos retos en la producción e implementación de tarjetas embebidas más pequeñas y económicas conocidas como SBCs (*single board computers*), que constituyeron estructuras no modulares como las computadoras, con la desventaja de que sus componentes no pueden ser actualizados o fácilmente reemplazados ya que son construidos con tecnología SoC (*system on Chip*) implementando los circuitos integrados en un chip simple con capacidades de procesamiento a la medida del usuario , así como también el logro de una tarjeta embebida de bajo costo como es la Raspberry PI, que cuesta alrededor de 35 dólares y que tiene grandes funcionalidades pues trabaja sobre software libre. (Pajankar, 2017)

Figura 13. Raspberry Pi model B+
Fuente: (Raspberry ORG, 2018)

Tabla 2

Características de Raspberry pi Model B+

Año de lanzamiento	2017
Arquitectura	ARMv8
Chipset	BCM2837B0
CPU	1.4Ghz 64-bit ARM Cortex-A53 SoC
GPU	VideoCore IV @ 400Mhz
Memoria	1GB LPDDR SDRAM
USB	4 USB 2.0
Salida de Video	DSI para pantalla touchscreen o HDMI
Almacenamiento	Puerto micro SD
Network	2.4Ghz y 5Ghz WLAN, Bluetooth 4.2
Fuente de Alimentación	5V/2.5 A con soporte PoE
Consumo	5.66 W

Fuente: (Raspberry ORG, 2018)

CAPÍTULO III

DISEÑO E IMPLEMENTACION

3. ETAPA DE DISEÑO E IMPLEMENTACION

3.1. IMPLEMENTACION DEL CLASIFICADOR

3.1.1 Adquisición de Imágenes para entrenamiento

En esta etapa se recopilan todas las imágenes positivas y negativas que van a formar parte del entrenamiento del clasificador, se entiende por imágenes positivas a las cuales contienen específicamente al objeto que se desea detectar, y como negativas a cualquier imagen en la que no esté contenido el objeto que se quiere detectar, la captura e realiza mediante el empleo de una minicámara de marca Powerextra de alta resolución en formato JPEG, a una resolución de 4608x2592 pixeles, la cámara es capaz de conseguir imágenes y video en 4K sobre una tarjeta de memoria microSD de 32GB clase 10 y para cada caso de muestreo se tomó por separado la cantidad de imágenes necesaria, dependiendo de la cantidad de imágenes de muestra tanto positivas como negativas influye directamente sobre la eficacia que tendrá el clasificador, para este proyecto se tomaron un aproximado total de 1000 imágenes positivas y 9340 negativas que van a conformar el banco de imágenes para el entrenamiento del clasificador que corresponde a la pistola, y de igual manera se tomó una muestra total de 942 imágenes positivas y se utilizaron las mismas imágenes negativas empleadas en el entrenamiento del cuchillo, pues estas imágenes únicamente deben cumplir la condición de que no tienen que incluir en ninguna porción de imagen a las muestras positivas, esta imágenes

pueden ser de cualquier tipo u origen. A estas muestras positivas se les realiza un tratamiento previo modificando sus características, principalmente su tamaño logrando una reducción de cada imagen hasta los 640 x 480 pixeles, esta operación es necesaria puesto que si se trabaja con imágenes muy grandes el preprocesamiento de cada imagen tomara mucho tiempo, y retrasando así el entrenamiento a varios días.

Posteriormente dichas imágenes se enumeran correctamente y se almacenan en la carpeta POS que contiene todas las imágenes positivas de cada objeto a entrenar, para luego así crear un archivo de texto, este archivo contiene información importante para el entrenamiento como son: la ruta de carga de imágenes positivas, el nombre de archivo de imagen junto con su extensión, la cantidad de objetos presentes en cada imagen y su posición en coordenadas donde se encuentra en objeto dentro de la imagen. Esta información es utilizada por la librería de OpenCV para así crear el vector de entrenamiento.

Figura 14. Imágenes de entrenamiento Pistola

Figura 15. Imágenes de entrenamiento Cuchillo

La imagen 14 detalla cómo está compuesto el conjunto de imágenes positivas que corresponden a la pistola, de aquí obtendremos la información donde está ubicada la imagen, todas almacenadas dentro de la carpeta POS, en la imagen 15 se tiene el conjunto de imágenes positivas que se utilizaran para el entrenamiento del Cuchillo, de la misma forma de aquí se obtiene la información de los pixeles de la imagen y se almacena en la carpeta POS que corresponde al cuchillo.

Para la captura de imágenes que forman parte de las imágenes “negativas” se realiza una descarga de imágenes de la web de uso libre y en cualquier formato para ser almacenadas dentro de la carpeta NEG, posteriormente estas imágenes serán tratadas, redimensionadas y agrupadas junto al banco de imágenes para el entrenamiento. El banco de imágenes está compuesto tanto por imágenes del internet e imágenes propias, estas imágenes contienen cualquier tipo de información, las imágenes negativas pueden ser cualquier tipo de imagen, fondo o color, la condición principal para una imagen

negativa es que dentro de ella no se encuentre ningún objeto considerado como positivo (pistolas, cuchillos), en si toda imagen que nos permita diferenciar al objeto del universo de análisis como se puede observar en la figura 16, y es por esta razón que las imágenes utilizadas en los entrenamientos toman el nombre de muestras positivas y negativas.

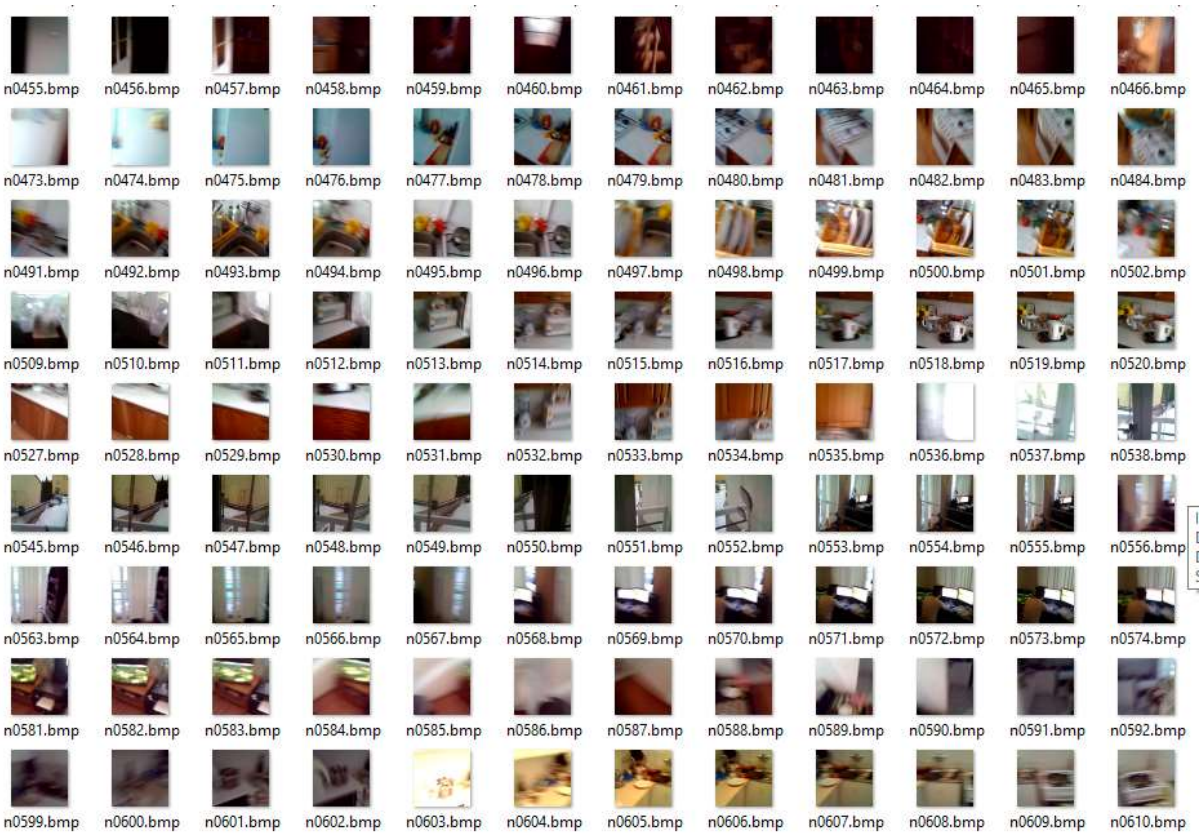

Figura 16. Imágenes negativas para entrenamiento

Es importante realizar la muestra con imágenes reales que contengan varios entornos de reflexión, iluminación y variación en los fondos de imagen, dependiendo de la calidad de imagen, tamaño e incluso de variaciones de tonos de color se espera tener distintos tipos de detección que se adapten fácilmente a varios entornos, ya que posteriormente todos estos datos son los que se utilizaran para el entrenamiento del clasificador y son los responsables de que el entrenamiento este correcto.

3.2. Preprocesamiento:

3.2.1. Acondicionamiento de imágenes para entrenamiento

Para realizar el entrenamiento previamente se debe modificar las características de las imágenes para que sean más fáciles de leer y procesar por la computadora, cada imagen primeramente es renombrada y organizada dentro de cada carpeta, buscando que cada nombre de archivo no contenga espacios en blanco ni caracteres especiales para facilitar y acortar las rutas que debemos manejar para los entrenamientos, para realizar el renombrado de imagen más fácil se utilizó una aplicación llamada *Advanced Renamer* en su versión liviana portable que permite renombrar de manera rápida una gran cantidad de archivos y dependiendo de la versión si es libre o pagada aumenta la cantidad de archivos que podemos renombrar a la vez; así mismo con la ayuda de otra aplicación llamada *Light Image Resizer* en su versión 5.1 se procede con la conversión de formato desde JPEG a BMP y se realiza la redimensión de las imágenes a una de 640x480 o sus equivalentes inferiores tratando de conservar características como relación de aspecto y centrado de imagen para que la muestra no se vea afectada y el entrenamiento sea satisfactorio.

Para realizar el renombrado de archivos y el redimensionamiento de imagen se emplea programas de uso gratuito que permiten realizar este tipo de operaciones de edición y cambio de formatos en cuanto a calidad de imagen, resoluciones y más características que se encentran disponibles en la web.

a) b)

Figura 17. a) Imágenes capturadas b) Imágenes tratadas

Como se puede observar en la imagen 17 a) las imágenes están como la cámara las captura, dichas muestras positivas deben renombrarse para que en el programa de entrenamiento no se generen errores en la lectura de datos, al lado derecho en la figura 17 b) se tiene las imágenes enumeradas y redimensionadas listas para generar el archivo de localización del objeto dentro de la imagen, este proceso facilita la extracción de información y la velocidad con la que se lo realiza.

3.3. Localización del objeto dentro de la imagen positiva

Existen varias maneras de crear un entrenamiento en base a imágenes positivas y negativas de clasificadores en cascada, entre las principales y más utilizadas están las de tomar fotografías de ambientes, descargarlas del internet o extraerlas directamente desde un archivo de video, el objetivo es conseguir la mayor cantidad de imágenes positivas y negativas para que OpenCV pueda trabajar, alternando distintos tipos de

iluminación y entornos de imagen que permitan desarrollar ambientes controlados más idóneos para la detección.

Una vez que se obtienen todas estas imágenes positivas y negativas, extraemos la información que tenemos de las imágenes positivas , aquí también existen dos formas mediante las cuales podemos hacerlo y es bien tomando fotos únicamente del objeto que deseamos entrenar y que dicho objeto ocupe la totalidad de la imagen, en otras palabras que en el tamaño total de la foto se debe contener al objeto deseado y no más información pues esta se considera directamente una muestra “positiva”, este proceso permite un rápido entrenamiento puesto que no debemos realizar otras tareas, otra manera es la de realizar la localización del objeto dentro de la positiva de mayor tamaño , en si lo que se busca es que dentro de una imagen cualquiera se indique donde está ubicado el objeto a ser identificado, siempre manteniendo en cuenta que debe existir proporciones de imagen a la hora de ser recortadas y que el tamaño de la muestra debe tener una relación entre el tamaño de imagen y el del objeto identificado. En el presente proyecto se realizó mediante el recorte de varias imágenes que contienen a los objetos a analizar (pistola, cuchillo) tomados en distintos ambientes de posicionamiento e iluminación y una vez realizando las muestras esta información es almacenada dentro de un archivo de texto que será posteriormente leído en el proceso de entrenamiento realizado por la librería de OpenCV.

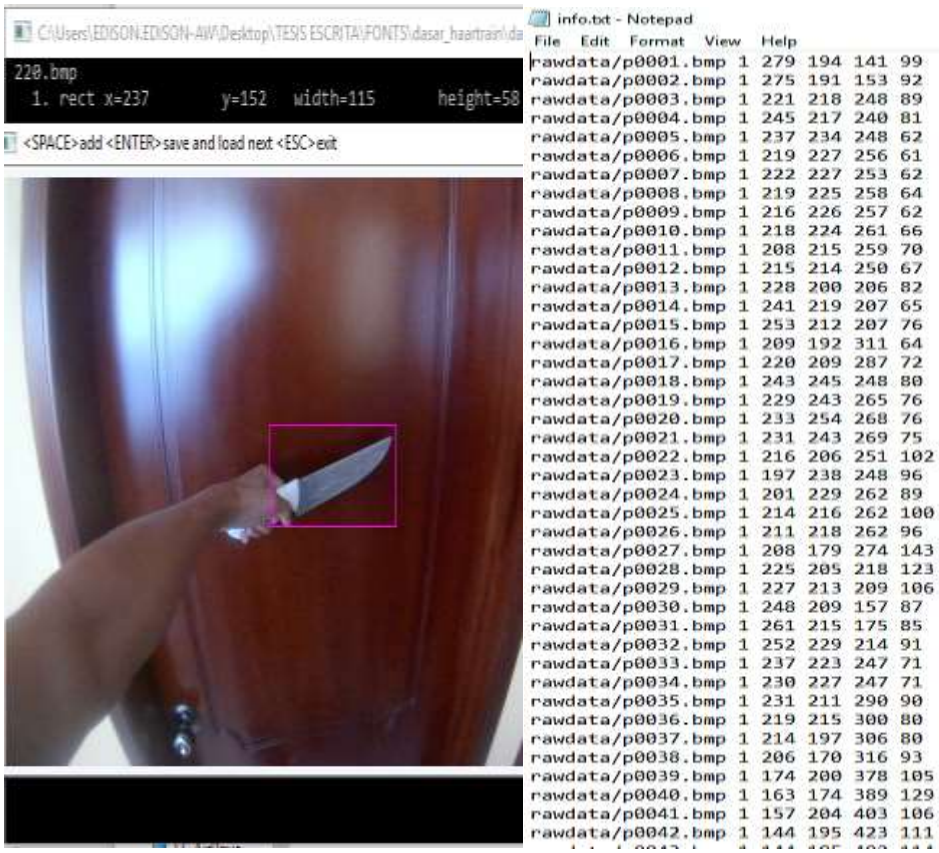

Figura 18. Archivo de información

En la imagen 18 se detalla en donde se encuentra la posición del objeto y que posición este ocupa dentro de la misma imagen, esta información se describe mejor posteriormente en el proceso de entrenamiento del clasificador que es el objetivo de esta adquisición y toma de datos.

Para la toma de las muestras se realiza el marcado del objeto de cada imagen indicando la localización dentro de la imagen, para esto se requiere de una aplicación utilizada en proyectos de investigación previos , la cual se llama dasar_haartraining que contiene object_maker propuesta por Mahdi Rezaei (Mahdi), que es una aplicación desarrollada para ir delimitando la región dentro de la imagen positiva en la que se encuentra ubicado el objeto de interés, como se ilustra en la figura 18 se realiza la

adquisición de la posición de la imagen en coordenadas y la dimensión en largo y ancho que ocupa el objeto dentro de la imagen y tomando en cuenta que la señalización corresponde a los pixeles de la imagen este marcado debe hacerse de izquierda a derecha para que no existan problemas de errores en los datos del objeto para el posterior entrenamiento, así tenemos que para la imagen positiva numero 1 (p0001.bmp) el objeto está ubicado en la posición 279,194 pixeles (en x,y) y que este tiene un largo y ancho de 141x99 (largo, alto o (w,h)), y así sucesivamente para cada imagen positiva como se detalla en la figura 18.

Si bien se desarrollarán varios tipos de clasificadores en los que se contemplan varios entrenamientos, se considera no adecuado implementar los que tengan bajos niveles de entrenamiento o poca cantidad de imágenes positivas ya que con las pruebas de trabajos anteriores que partieron desde 10 niveles de aprendizaje no fueron para nada eficientes, para estas pruebas se tomaron una base de datos de imágenes que comprendía 200 positivas y 900 negativas obteniendo resultados muy limitados y deficientes, así mismo se pudo establecer un precedente y determinar así varios errores que limitaron su calidad como son dimensiones y formatos de imagen que pueden manejar las librerías de entrenamiento de OpenCV.

Otro factor importante tomar en cuenta para un entrenamiento son los parámetros de captura de muestras de imágenes, al realizar la adquisición de muestras no se tomó en cuenta un valor dimensional para la muestra y se utilizó los parámetros ya establecidos, esto se da porque el sistema tiene un entrenamiento de muestra por defecto de 24x24 pixeles que es en su mayoría el que se utiliza para los entrenamientos, este parámetro fue implementado en sus orígenes para la detección de rostros, al realizar la

prueba rápida de los clasificadores estos carecían de detección en la mayoría de los casos pues dichos parámetros que estas deben estar en relación con la imagen de entrada o más específicamente deben guardar relación con el objeto a ser identificado, en el caso del objeto pistola no se aprecia tanto esta referencia ya que el tamaño de muestra se asemeja a un cuadrado aunque su valor estable asemeja más a un rectángulo de relaciones de ancho y altura de 2 a 1, en cambio esto no sucede con el objeto cuchillo cuyo tamaño de muestra o presencia dentro de imagen correspondiente a una equivalencia del triple de ancho por unidad de alto (3 a 1), este análisis es importantísimo ya que a la hora de entrenar si se toma un valor no adecuado el producto final puede ser deficiente y no presentar resultados a la altura de los niveles de entrenamiento.

a) b)

Figura 19. Localización del objeto dentro de la imagen

Como se observa en la figura 19 se aprecia cómo se cumple la relación entre largo y ancho dicho anteriormente en la que la pistola (2 a 1) y el cuchillo (3 a 1), y por ser objetos de distinta forma necesitan así mismo distintos parámetros de entrenamiento. Con este simple pero vital análisis se obtuvo un criterio para elegir un factor de muestreo de entrenamientos a futuro que brinden grandes resultados y que también de esta manera necesiten un menor nivel de entrenamiento para lograr satisfacer las necesidades de detección y clasificación.

3.4. Creación del vector de entrenamiento:

Uno de los pasos iniciales para el entrenamiento de un clasificador es la creación del vector que contiene las características (nombre de archivo, posición y tamaño) de las imágenes , aquí se crea un archivo tipo vector (.vec), el cual contiene toda la información anteriormente detallada, esta información es leída por la librería opecv_createsamples para su previo entrenamiento , la creación del vector de características es muy importante, este vector contiene el número de muestras que se van a crear para el análisis del entrenamiento, este número de muestras sigue una formula aproximada para su desarrollo en programación , dicha formula está dada por varios factores como son el número de imágenes positivas , número de imágenes negativas, numero de etapas de entrenamiento de la cascada, y los valores de aproximación deseados como son los niveles de error y falsos positivos.

Parámetros de opencv_createsamples:

- **-vec <vec_file_name>:** Nombre del archivo tipo vector que contendrá las muestras positivas para el entrenamiento.
- **-img <image_file_name>:** ruta donde se encuentro el objeto de entrada.
- **-bg <background_file_name>:** contiene la ruta e información de las imágenes de fondo
- **-num <number_of_samples>:** Numero de muestras positivas para generar..
- **-bgcolor <background_color>:** controla el color de fondo y su transparencia para discernir entre el objeto que se desea separar.
- **-inv:** si esta especificada indica que se realizara una inversión de color

- **-randinv:** Si esta especificada los colores serán invertidos de forma randomica.
- **-maxidev:** desviación máxima de los pixeles de muestras en primer plano.
- **-maxxangle:** Angulo de máxima rotación con respecto al eje X, está indicado en radianes.
- **-maxyangle:** Angulo de máxima rotación con respecto al eje Y, está indicado en radianes.
- **-maxzangle:** Angulo de máxima rotación con respecto al eje Z, está indicado en radianes.
- **-show:** Usado como ayuda para presentar las muestras adquiridas, este puede ser utilizado para observar si el entrenamiento se llevara a cabo correctamente con las muestras que se están obteniendo. Se puede omitir este proceso presionando la tecla ESC.
- **-w <sample_width>:** Tamaño en pixeles del ancho de las muestras de entrenamiento.
- **-h <sample_height>:** Tamaño en pixeles del alto de las muestras de entrenamiento.

En caso de existir problemas a la hora de crear el vector , o al utilizarlo en algún entrenamiento se puede verificar que se cumpla con ciertas condiciones , entre las principales están las del número de muestras positivas que estarán contenidas dentro del archivo tipo vector, el número de imágenes positivas y negativas que se utilizaran y en ciertos casos el número de niveles de entrenamiento que se deben utilizar, ya que de esto depende también la cantidad de muestras que se deberán crear para dicho vector, para realizar una aproximación de los valores que debemos utilizar se lo puede realizar

mediante la siguiente fórmula propuesta por Maria Dimashova en Opecv.org, que contiene una buena explicación de que valores a emplear para un correcto entrenamiento:

$$vecfile\ has\ to\ contain >= (numPos + (numStages - 1) * (1 - minHitRate) * numPos)S$$

Ecuación 4. Fórmula de cálculo de Parámetros de entrenamiento*.*
Fuente: (Dimashova, 2012)

Donde el parámetro S viene a ser un recuento de las muestras que se utilizaran como fondo, generalmente al modificar los valores de entrenamiento este valor es el que genera problemas por esto con la utilización de la formula anterior se puede obtener parámetros válidos para el entrenamiento, por ejemplo, si quisiéramos 10 niveles de entrenamiento con un conjunto de 200 imágenes positivas y 10000 negativas con un parámetro de minHitRate 0.995 y siguiendo la ecuación 4 tendríamos:

$$200 >= (numPos\ +\ (10 - 1)\ *\ (1\ -\ 0.995)\ *\ numPos)\ +\ S$$

La fórmula no es una guía de cómo crear los parámetros para el clasificador en sí, no está a disposición en ningún paper o trabajo anterior, más bien es una explicación clara de cómo se utilizan estos valores dentro del entrenamiento en base a experiencias previas de otros usuarios, pero que es una gran ayuda para tener presente a la hora de elaborar proyectos con entrenamientos de clasificadores en cascada, razón por la cual fue incluida en este proyecto. (Dimashova, 2012)

3.2 Entrenamiento del clasificador

A continuación se detalla el entrenamiento del clasificador HAAR y LBP a partir de los datos anteriores, luego de generar el archivo tipo vector con las características de entrenamiento e información de muestreo hacemos uso de las librerías de entrenamiento de OpenCV, aquí hacemos una elección de que librería utilizaremos puesto que se dispone de dos aplicaciones diferentes de entrenamiento como son opencv_haartraining y opencv_traincascade, cuya diferenciación está en que haartraining genera directamente un único archivo de cascada en base a los parámetros adquiridos y en cambio traincascade genera a su salida cada nivel de entrenamiento de la cascada y además utiliza procesamiento multinúcleo, reduciendo en una gran mayoría el tiempo de entrenamiento en algunos casos hasta en la mitad del tiempo necesario y es por esta razón que se utilizará esta segunda opción. Esta librería propia de OpenCV necesita ser instalada previamente en el computador que realizara el entrenamiento sobre sistema operativo Linux, para realizar su instalación únicamente necesita la línea “sudo apt-get install libopencv-dev” , para posteriormente solo ejecutar su librería, esta librería necesita varios parámetros de entrada que indiquen las características que va a llevar el clasificador, estas van desde el tipo de cascada a utilizar, el formato que llevara, el número de imágenes positivas, numero de imágenes negativas, e incluso la cantidad de memoria que deseamos asignar para el entrenamiento, todas estas propiedades son importantes, pues de ellas depende la calidad y eficiencia del clasificador.

Para establecer los parámetros del entrenamiento del clasificador se necesitan de varias líneas de argumentos de entrenamiento, estos parámetros comunes son los que establecen el método utilizado, los niveles requeridos entre otros propósitos.

Parámetros de entrenamiento del clasificador:

- **-data:** Es la ruta donde se generará el o los archivos de la cascada
- **-vec:** Es la dirección y nombre del vector creado anteriormente con opencv_createsamples.
- **-bg:** Directorio de información de imágenes negativas.
- **-numPos:** Es el número de muestras o imágenes positivas que empleamos en el entrenamiento.
- **-numNeg:** Es el número de muestras o imágenes negativas que se utilizaron en el entrenamiento.
- **-numStages:** Aquí se indica cuántas etapas de entrenamiento se van a realizar.
- **-precalcValBufSize:** Es el valor del buffer de almacenamiento en MB, dependiendo del valor asignado el entrenamiento será más rápido o lento.
- **-precalcIdxBufSize:** De igual manera tamaño en MB para los índices de características, se debe tener en cuenta que la suma de estos valores combinados no debe exceder el tamaño total de uso de memoria.
- **-numThreads:** Indica el número máximo de hilos que se utilizaran durante el entrenamiento.
- **-acceptanceRatioBreakValue:** Aquí determinamos cuan preciso queremos que sea el entrenamiento y dependiendo de este valor realizara varios ciclos de entrenamientos hasta satisfacer el requerimiento.

Así mismo también se deben incluir más parámetros que corresponden específicamente a los que contendrá la cascada, que indiquen el tipo de cascada resultante, las características que se utilizaran

- **-stageType <BOOST(default)>:** Se asigna el tipo de cada nivel, actualmente solo se soportan entrenamientos tipo BOOST.
- **-featureType<{HAAR(default), LBP}>:** Type of features: HAAR – HAAR-like features, LBP - local binary patterns.
- **-w:** especifica en ancho de las muestras de entrenamiento dado en pixeles, es el mismo valor que se utilizó para crear las muestras de entrenamiento (opencv_createsamples utility).
- **-h:** Indica el tamaño en Altura de las muestras de entrenamiento, este también está dado en pixeles y de igual manera es el mismo utilizado en la creación de muestras del vector de entrenamiento.

Como el entrenamiento está basado en parámetros de clasificador de tipo BOOST que es el tipo de nivel de característica, y se deben establecer sus términos de entrenamiento que incluyen el tipo de algoritmo, si es discrete AdaBoost, real AdaBoost, LogitBoost o gentle Adaboost, juntamente con los parámetros que se están buscando para el clasificador como tasa de aceptación, falsa alarma, entre otros.

- **-bt <{DAB, RAB, LB, GAB(default)}> :** indica el tipo de clasificador tipo Boost, puede ser : DAB - Discrete AdaBoost, RAB - Real AdaBoost, LB - LogitBoost, GAB - Gentle AdaBoost, por defecto se utiliza el GAB en los entrenamientos.

- **-minHitRate <min_hit_rate>:** Es la mínima tasa de aciertos requeridos por el usuario para cada nivel de entrenamiento del clasificador.
- **-maxFalseAlarmRate <max_false_alarm_rate>:** es el valor máximo requerido para el nivel de falsa alarma de cada nivel.

En manera particular para los clasificadores con características de tipo HAAR (*Haar-like feature*) se requieren de un parámetro que indique el tipo de características que pueden ir desde Basic con muestras simples o modo ALL con variaciones y rotaciones.

- **-mode <BASIC (default) | CORE | ALL>:** Selecciona el tipo de característica en modo BASIC usa únicamente imágenes frontales de una dirección (modo vertical), mientras que en ALL se usa variaciones y rotaciones de 45 grados.
- De igual forma los parámetros de LBP o Local Binary Patterns no son necesarios.
- Los parámetros en los que no se establece valores por el usuario toman todos los valores por defecto que están indicados anteriormente, los entrenamientos al depender de características como tamaños de objeto, iluminación y calidad de imagen deben ser muy tomados en cuenta en cada entrenamiento ya que no son los mismos parámetros para cada objeto, el resultado de una configuración ideal también dependerá de la cantidad de pruebas de entrenamiento para que se determine valores satisfactorios.

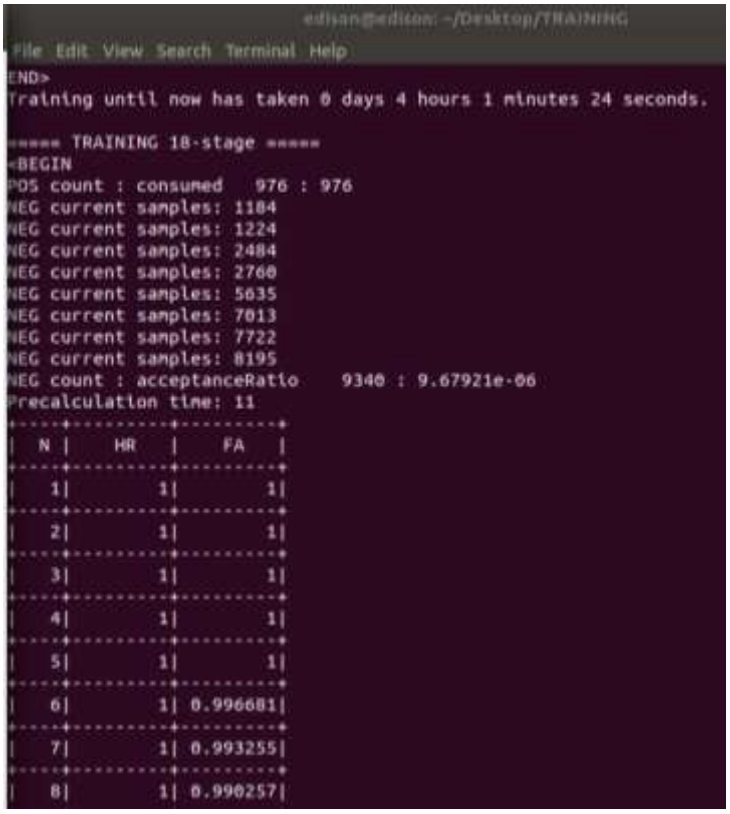

Figura 20. Imagen de entrenamiento de clasificador

En las columnas de la figura 20 están representadas las características empleadas para el entrenamiento, los parámetros utilizados y descritos anteriormente, y si el programa logra calcular una proyección del entrenamiento continua con el proceso o previamente detalla un error si detecta inconsistencias en los datos incluidos con la información de entrenamiento. La primera columna N detalla el número de iteraciones que se necesitaron para lograr los parámetros de entrenamiento de esa etapa de la cascada, la segunda columna HR o Hit Rate indica el valor comprendido entre 0 a 1 de Hit Rate, la tercera y última columna FA o False Alarm, detalla el porcentaje de False Alarm generado y busca que su valor este dentro del requerido por el usuario previamente, una vez que este valor es cumplido continua con el siguiente nivel de

entrenamiento. Así mismo al culminar cada nivel de entrenamiento el sistema indica cuanto tiempo ha transcurrido desde que se inició el entrenamiento y cuando termina el proceso simplemente este cierra su ventana de ejecución generando los archivos de entrenamiento. Una vez terminada la operación de la librería opencv_traincascade y de realizar los distintos niveles de entrenamiento al cumplir los parámetros ingresados, el archivo final es guardado con el nombre de cascade.xml, el archivo tiene una extensión .xml como se observa en la figura 21 y se almacena dentro de la carpeta con la ruta asignada anteriormente en -data. De igual manera por cada nivel de entrenamiento se van generando continuamente en caso de una interrupción o falla del programa, estos niveles de entrenamientos ya superados pueden ser cargados nuevamente o bien si se desea incrementar los niveles de entrenamiento, estos se pueden eliminar luego de que el programa haya terminado su ejecución y se genere el archivo final.

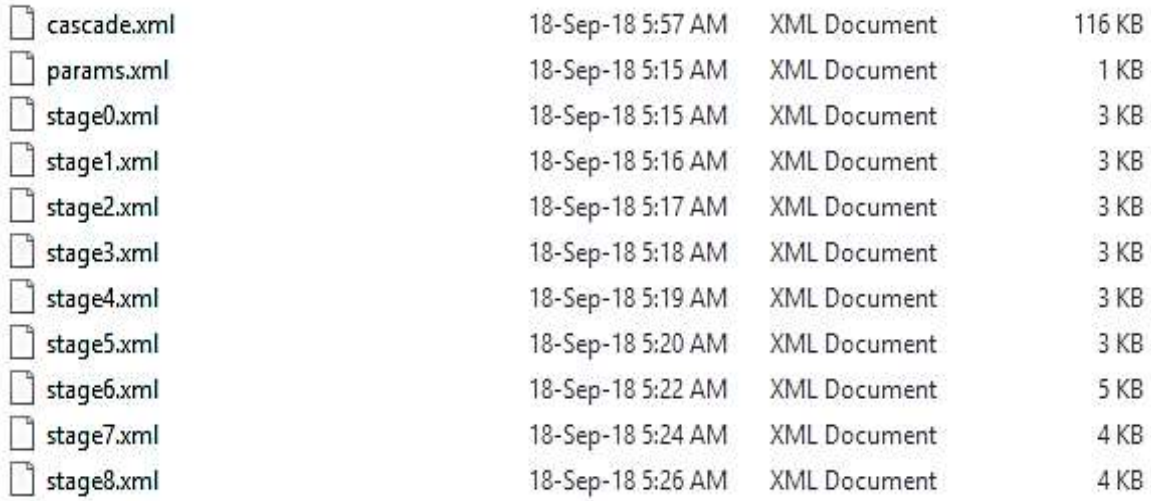

Figura 21. Cascada final generada

3.3 IMPLEMENTACION DEL DETECTOR

El algoritmo de ejecución para las cascadas entrenadas previamente se realizó sobre el IDLE de Python 3.5.3 corriendo en un sistema operativo Linux Ubuntu 16.04 con la ayuda de un computador Alienware M17x que cuenta con un procesador Intel Core i74700MQ (8 CPUs), con una memoria de 24 GB de RAM a 2133Mhz, Disco duro de estado sólido y tarjeta de video Nvidia GTX 765M para procesamiento de video.

La versión de OpenCV utilizada es la 3.3 y posteriormente se la actualizo a la 3.5 sin encontrarse variaciones en los procesos de ejecución, la versión de Python fue desarrollada sobre un entorno virtual creado dentro del almacenamiento interno del computador, pues es la manera en la que trabaja la librería de OpenCV. A continuación se indica como fue el proceso de programación del sistema de detección, en un principio se trabajó únicamente sobre la computadora, pues el trabajar en un entorno notablemente veloz y con más prestaciones como es un computador portátil, nos brinda mayores beneficios a la hora de desarrollar un programa, al disponer de mayor cantidad de procesamiento y memoria RAM que es los principal a la hora de realizar proyectos de aprendizaje de máquinas, y si bien es cierto que se busca como objetivo el implementar el sistema sobre una tarjeta embebida, se opta por elaborar el programa y luego implementarlo y adecuarlo para su uso en la Raspberry Pi 3 Model B+ 2018. Entonces dicho los requerimientos anteriores, con esto se procede a crear los bloques de programa buscando elaborar de manera ordenada y sistemática un programa que sea fácilmente adaptable a cualquier entorno de programación posterior y de fácil entendimiento, para esto se desarrolla mediante la creación de un programa principal (main) y funciones en las que se divide al programa que se llaman cuando son requeridas.

3.4 Framework del sistema

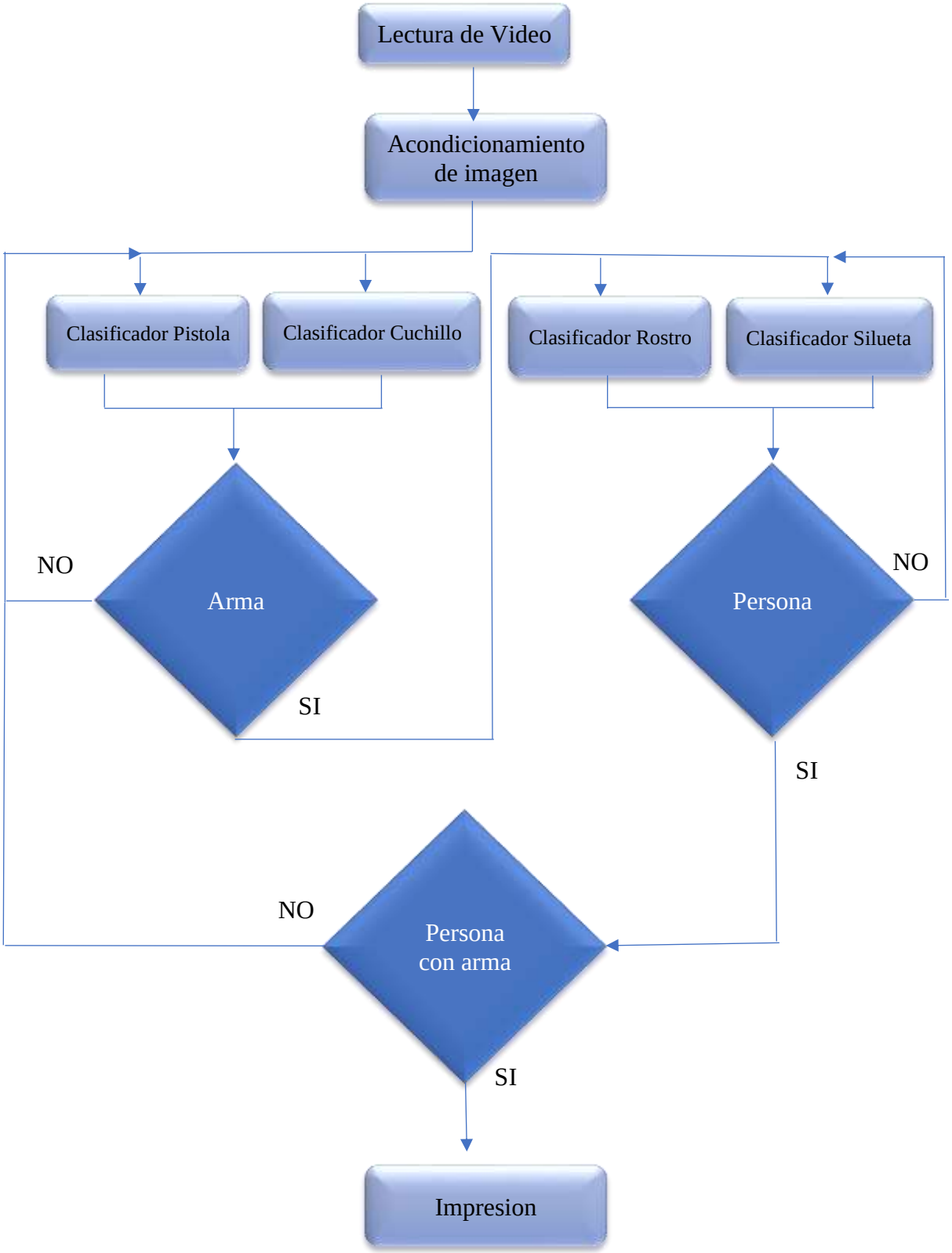

Figura 22. Framework del sistema

El sistema al evaluar la posesión de un arma inicia por determinar si esta es considerada una amenaza o no, se debe evaluar si existe la presencia del arma y al mismo tiempo una persona que pueda acceder a ella o se encuentra portándola, al realizarse una detección positiva por parte de estos eventos se genera o no una alerta visual dentro del sistema. Al determinar que un arma no es una amenaza al no encontrarse presencia humana, el sistema realiza la detección, omitiendo la alerta y continuando con el proceso de análisis, para esto se evalúan tres tipos de eventos de los que se incluye el orden para el análisis y son:

1. Existencia de arma en ausencia de personas: En el evento está presente un arma que puede ser un cuchillo o una pistola, el sistema es capaz de detectar la presencia del arma y esta no debe ser identificada como peligrosa, pues no existe la presencia humana para que el arma sea considerada como peligrosa, al indicar “presencia humana” el sistema continúa y escanea en busca de un rostro o una parte de su cuerpo que pueda generar como resultado la detección positiva de un ser humano, también sabemos que vamos a tener objetos que en algunos ambientes pueden semejarse a la silueta o rostro humano y posteriormente se tienen que validar y analizar, por esta razón se continua con más eventos de detección.

2. Presencia de persona con el rostro visible en posesión de arma: Mientras la detección de la persona pueda ser posible se considera el escenario en el que mientras un individuo tiene el rostro descubierto y está colocado frente a la cámara, luego de detectar la presencia de un arma, el sistema realiza un escaneo del entorno en busca

de un rostro, si el sistema no genera ninguna detección considerada como peligrosa, entonces procede a finalizar su búsqueda y seguir con la siguiente cadena de detección.

3. Presencia de personas con rostro no visible en posesión de armas: Un escenario en el que no se pueda determinar con certeza la detección una persona mediante la identificación de su rostro. El sistema realiza una detección de la silueta del ser humano, aquí la persona pude tener el rostro cubierto o de difícil detección, para resolver este problema el sistema es capaz de identificar las extremidades superiores de los seres humanos si estos están en posiciones neutrales como los brazos extendidos hacia abajo y si la persona se encuentra de pie, al realizar esta detección el sistema envía al sistema una alerta visual que indica que el arma es considerada peligrosa, pues un arma sin la presencia humana no debería representar una amenaza.

3.10. Diagrama de flujo del detector:

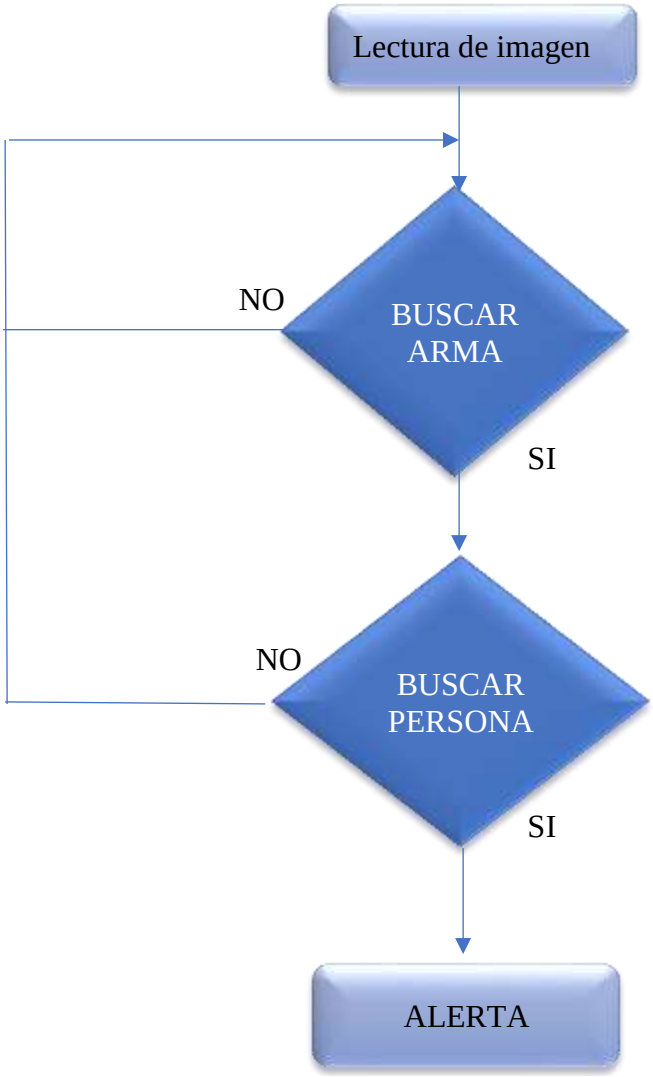

Figura 23. Diagrama de Flujo del Detector

3.11. Algoritmo detector:

1. Se inicializan las librerías requeridas en python para OpenCV y las de proyección de video
2. Si inicializan todas las variables globales
3. Se crean los accesos de los dispositivos de entrada o adquisición de imágenes
4. Se cargan los clasificadores en cascada
5. Ser realiza la lectura de cuadros de imagen desde el dispositivo de entrada
6. Se realiza el acondicionamiento de la imagen: ajuste del tamaño, conversión a escala de grises, y limitación de la tasa de cuadros por segundo.
7. Se realiza la detección de rostros devolviendo valores de análisis (1/0 o datos de cascada) y se continua con la detección de arma en caso de existir un evento positivo de detección.
8. Si la detección de rostro no genera eventos positivos, se realiza una ejecución de otro clasificador llamado human_detec que permite identificar la silueta del cuerpo humano y de igual manera devuelve datos llamados detector y la imagen analizada, si existe detección procede con la ejecución de la función gun-detec para verificar la presencia de armas retornando valores de impresión como la imagen y la etiqueta.
9. Luego de identificar el objeto, se realiza la impresión en pantalla.

Luego de indicar brevemente como está desarrollado el algoritmo detector visto en la figura 23, se explica a continuación los pasos necesarios para que el programa pueda cumplir su función se necesitan cargar e inicializar varias librerías de Python juntamente con Opencv, entre ellas están time, que permite generar tiempos de espera y lapsos de

pausa de programa, cv2 es la librería perteneciente a OpenCV, aquí también se inicializan las variables globales utilizadas como “detector” que almacena e indica que tipo de evento se está realizando de acuerdo a validaciones futuras y la variable “img”, que almacena el frame obtenido desde la cámara Web o raspicam.

```
import time
import cv2
detector= None
img=None
```

Así mismo se requiere una lectura previa de las rutas donde están almacenados los archivos clasificadores en cascada, y si el programa se encuentra en el mismo directorio basta con añadir únicamente el nombre de archivo y su extensión xml que es la cascada generada anteriormente. Dicho esto, se hace la lectura de las rutas de cada clasificador Rostro, humano, pistola y cuchillo. Basta con asignar un nombre para el llamado completo de la función conjunta con OpenCV y para esto se puso a cada caso una identificación por ejemplo face_cascade, que realiza el llamado a cv2 cargar el archivo clasificador faces.xml.

```
face_cascade = cv2.CascadeClassifier('faces.xml')
human_cascade = cv2.CascadeClassifier('human.xml')
gun_cascade = cv2.CascadeClassifier('gun.xml')
knife_cascade = cv2.CascadeClassifier('knife.xml')
```

Se requiere también inicializar y asignar un tiempo de espera al dispositivo de adquisición de imágenes que en este caso puede ser una cámara web o una raspicam,

dependiendo de cuantas cámaras tengamos conectadas a nuestra PC el ID o número de dispositivo varia y puede ser (0) (1) y así hasta llegar al número de dispositivos conectados. Todas estas acciones son necesarias antes de hacer el llamado a la función principal, que es la encargada de realizar las tareas de llamada a función de todos los procesos principales del programa, desde el llamado a carga y adquisición de imágenes desde la cámara, redimensionado de imagen para lograr mayor velocidad al procesamiento, realizar tratamientos de la información adquirida como es el caso de conversión de color a escala de grises para así también reducir el costo computacional y realizar una valoración de estos datos para posteriormente identificar sus características mediante los clasificadores que serán llamados de igual manera por funciones.

```
        vs = cv2.VideoCapture(0)
        time.sleep(2.0)
```

```
while True:

        #frame = vs.read()
        #frame = imutils.resize(frame, width=400)
        #gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    ret , image = vs.read()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    aux=gray

    detector,data= face_detect(aux)
    if detector==0:
        detector,data=human_detect(aux)

    while detector==1 or detector==2:
        label="HUMANO"
```

La primera función creada es face_detect() cuyos datos de entrada son necesariamente una imagen y una variable de análisis que llamaremos “detector”, dentro

de esta función se desarrolla la detección de rostros como parte de la identificación del ser humano se puede realizar la conversión en escala de grises de la imagen como ya se dijo anteriormente dentro de la función principal o bien subdividiendo este proceso en cada una de las funciones de llamada de datos según la validación otorgada se puede comparar que al existir o no datos creados por la detección por parte del clasificador, esta devuelva sus características o bien retorne 1 o 0 para validarlo como evento positivo o negativo, dado esto al encontrar un evento positivo, se retorna los puntos en los que se encuentra dicho evento para proceder con el sistema, estos datos de retorno son los que indican o no si el programa debe continuar con la búsqueda de características o bien seguir la estructura del programa para analizar un nuevo clasificador.

```
def face_detect(img):
 print("ESCANENADO")
 aux = img
 faces = face_cascade.detectMultiScale(gray, 1.3,5)
 if (len(faces)>0):
    return 1,faces
 else:
    return 0,0
```

La segunda función manejada es la human_detect es una parte complementaria de la función anterior, pues su objetivo es luego de verificar la detección de rostros, ejecutar otro clasificador para reconocer a una persona dentro de un cuadro de video (frame) que se obtuvo al inicializar el dispositivo y proceder con la lectura del mismo, este clasificador que tiene su entrenamiento en base a la silueta del cuerpo humano, analiza este frame y determina de igual forma la existencia de un evento para así posteriormente continuar con el proceso de análisis.

```
def human_detect(image):
 print("Buscando       HUMANO")
 aux = image

 #gray = cv2.cvtColor(aux, cv2.COLOR_BGR2GRAY)
 human = human_cascade.detectMultiScale(aux, 1.3,18)

 #print("deteccion rostro  :  ", len(faces))
 if (len(human)>0):
    #print(" Rostro detectado")
    detector=human
    return 1,human
 else:
    #print(" SIN ROSTRO ")
    return 0,0
```

Una vez realizada la identificación o presencia de un ser humano dentro del video, se procede con la función para la detección de arma gun_detect y knife_detect, esta función realiza la lectura del clasificador entrenado previamente como Cuchillo o Pistola, dependiendo del clasificador que se cargue en el programa, este será capaz de reconocer una vez aprobadas las funciones anteriores puesto que un arma no es una amenaza si no existe una presencia humana, para esto se retorna los datos de validación o bien la imagen original adquirida en escala de grises para llevarla a la impresión, al validar la presencia del arma y también la del ser humano el sistema emite una alerta visual en pantalla indicando peligro, función que forma parte de la impresión de resultados que se mostraran posteriormente.

```
def gun_detect(img):
 print("Buscando                ARMA")
 aux = img
 faces = gun_cascade.detectMultiScale(gray, 1.1,5)
 if (len(faces)>0):
```

```
    detector=faces
    return 1,faces
 else:
    return 0,False
def knife_detect(img):
 print("Buscando                    CUCHILLO")
 aux = img
 faces = knife_cascade.detectMultiScale(gray, 1.1,5)
 if (len(faces)>0):
    detector=faces
    return faces,gray
 else:
    return 1,gray
```

Finalmente, una vez que se realiza la detección e identificación en cada caso, y luego de validar esta información para indicar de qué tipo de evento se trata, se procede con el envío de toda esta información al periférico de salida deseado (uno o varios monitores por ventana) mediante una función de impresión llamada “draw”, como se puede apreciar en las siguientes líneas de código, aquí existen varias validaciones para que según los retornos de cada funciona anterior se imprima el objeto detectado indicando si es humano, pistola o cuchillo, al analizar estas variables se muestra en pantalla los datos obtenidos.

```
def draw(image, detector,label):

 font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(image,'ESCANEANDO',(10,50), font,
        0.5,(255,255,255),2,cv2.LINE_AA)
        cv2.imshow("DETECCION PERSONA", image)
break

for (x,y,w,h) in detector:
      if label=='arma':
            cv2.putText(image, "ARMA DETECTADA", (250, 20),
                  cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        a=0
```

```
            b=0
            c=255
        else:
            a=255
            b=0
            c=0
        if(label=='HUMANO'):
            font = cv2.FONT_HERSHEY_SIMPLEX
                    cv2.putText(image,'DETECTADO HUMANO',(10,50),
font,0.5,(255,255,255),2,cv2.LINE_AA)
            a=0
            b=255
            c=0
```

3.5 Implementación en RaspberryPi

Luego de tener instaladas todas las librerías requeridas, desde la consola accedemos al entorno virtual de Python para OpenCV, luego para iniciar la interfaz se accede al IDLE del programa con las sintaxis sudo Python -m idlelib.idle, en la cual se nos despliega una ventana para la edición y generación de código de Python (Python IDLE), aquí ya podremos editar nuestro código, realizar pruebas, y adaptaciones del programa con los valores del ambiente, primeramente el acceso al ingreso de datos varía dependiendo de las condiciones del entorno, el obtener imágenes desde una cámara web externa no es lo mismo que obtenerlos desde la raspicam que se utiliza para la implementación, entonces dicho esto se tiene la siguiente sintaxis de programa:

```
vs = VideoStream(usePiCamera=1 > 0).start()

time.sleep(2.0)


# loop over the frames from the video stream
while True:


        #frame = vs.read()
        #frame = imutils.resize(frame, width=400)
        #gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    image = vs.read()
```

Figura 24. Código de inicialización de Raspicam

En la figura 24 se tiene la inicialización de la cámara de la raspberry, aquí también podemos incluir en número de frames por segundo (FPS) que queremos utilizar, tratando de reducir el costo computacional que conlleva el analizar una mayor cantidad de imágenes por segundo. De igual manera se puede controlar la resolución o tamaño de imagen de video tanto para la entrada como para la salida de video y así también optimar aún más el programa.

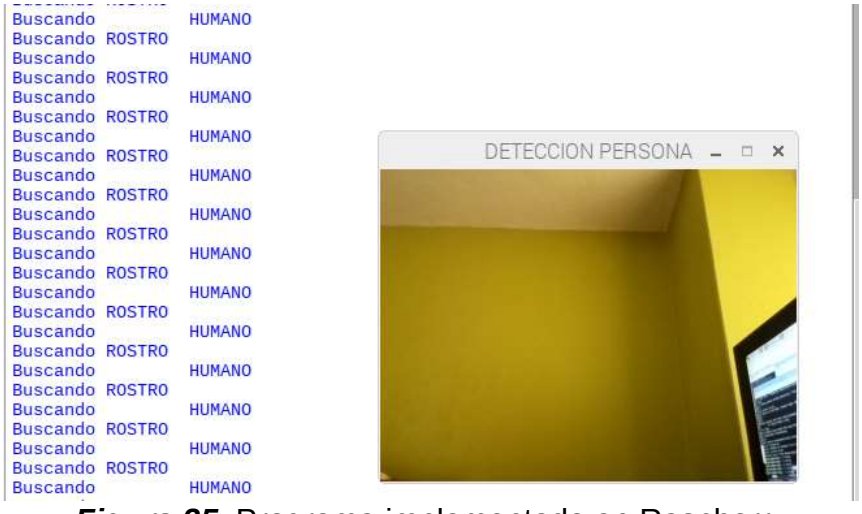

Figura 25. Programa implementado en Raspberry

Así mismo como una ampliación del programa final se realizó una edición final en la salida del video para comprimir todavía más los datos introducidos y tener una respuesta de detección más alta a una resolución de video de 176x144 como se observa en la figura 25, en el que se tiene un tamaño muy pequeño de video, pero que tiene una alta respuesta en tiempo real que es lo que se tiene como finalidad para este trabajo de investigación y como se observa en la figura 26 se puede expandir esta resolución a tamaño completo de pantalla para poder apreciar mejor el video.

Figura 26. Video a pantalla completa

3.6 Creación de un archivo ejecutable para el usuario

Con el propósito de que este programa sea implementado sobre cualquier plataforma como un aporte extra se realizó la migración de nuestro algoritmo desarrollado en python sobre plataforma Linux a la de Microsoft Windows, mediante el empleo de pyinstaller, que se puede obtener desde sudo apt-get install pyinstaller se ejecuta la línea de comando pynstaller –onefile program.py , se generan un conjunto de archivos entre los que se encuentra el .exe, basta con abrir el ejecutable y nuestro programa podrá correr sin necesidad de instalar los prerrequisitos como son librerías ni aplicaciones de programa.

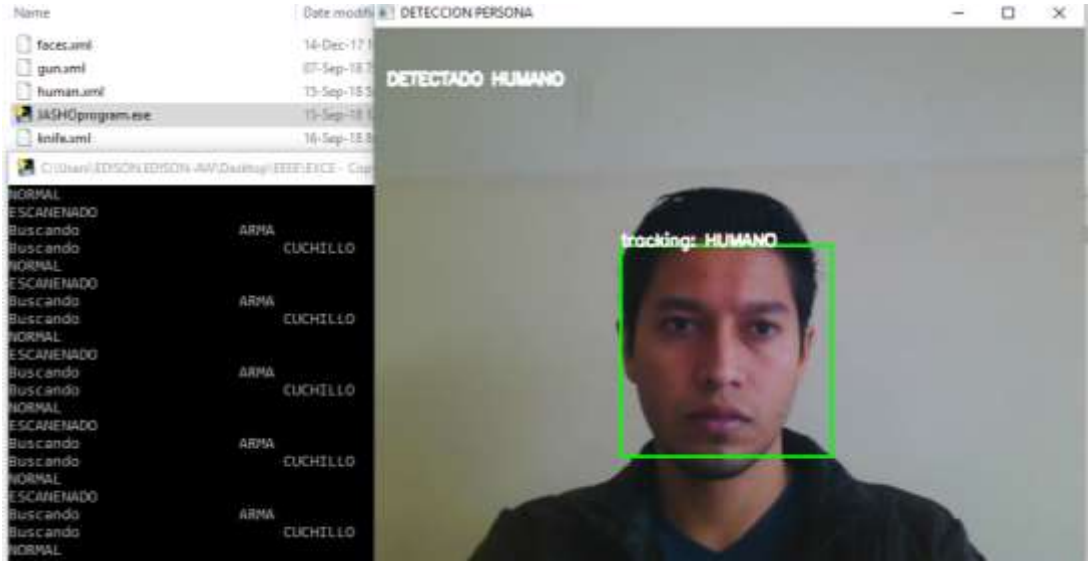

Figura 27. Programa de ejecución libre en Windows 10

En la figura 27 se observa los archivos generados migrados desde Python, entre los que están el ejecutable de Windows junto a los clasificadores necesarios, y así mismo al ejecutarlo no se necesitan tener instalado las librerías de OpenCV ni demás archivos necesarios para Python, así se logra migrar desde una plataforma instalada para investigación y programación a una libre accesible para cualquier usuario.

CAPÍTULO IV

ANALISIS DE RESULTADOS

4. ANALISIS DE RESULTADOS

En este capítulo se muestra los resultados de las pruebas del detector operando video de entrada en tiempo real por lo que se procesan en forma secuencial la detección de eventos en todos los frames de video (30 fps aproximadamente) generando un efecto de seguimiento de objetos (tracking) en el video de salida del detector.

Para efectos de evaluación del detector se utiliza un archivo de video pregrabado del cual se capturan determinadas frames de prueba cuyo resultado se cuantifican a través de la matriz de confusión, extrayendo así la precisión y sensibilidad del detector.

4.1. Evaluación de los Detectores:

Para la evaluación se utiliza un video de prueba pregrabado que incluya los distintos eventos a ser detectados, de este video se capturan aleatoriamente *frames* tal como se muestra en la figura 28 y también otras imágenes de prueba. para complementar la muestra de imágenes requeridas para el análisis de desempeño del detector.

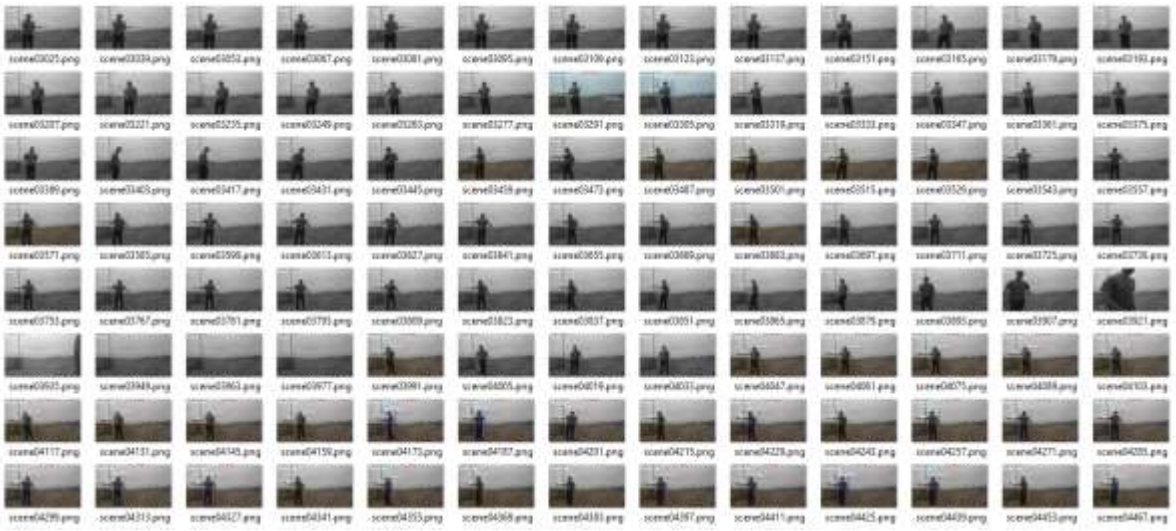

Figura 28. Imágenes para análisis

El video pregrabado con la cámara web del computador es sometido a los tres tipos de detectores implementados en este trabajo y se realizó una nueva grabación en la que se incluyen en análisis de nuestro algoritmo de detección.

Se tomó un frame de muestra por cada 30 frames de video, esto significa que en un video de 2 minutos con 30 fps (3600 frames) se procesaran únicamente 120 frames en el detector ya que es conveniente analizar una muestra representativa siendo suficiente para determinar los valores de análisis.

Como se enuncio anteriormente, el proyecto fue evaluado en varios entornos que generen una respuesta para el clasificador y el algoritmo de detección, a fin de determinar las características que debe tener el programa, analizar los casos en los que se deben utilizar uno u otro clasificador.

4.1.1. HOG + SVM

Para el primer evento de análisis se utiliza al menos un clasificador que posea características HOG+SVM para el análisis en la detección, aquí se analizan las muestras obtenidas con cada entrenamiento , se realiza una clasificación y comparación de la imagen de entrada por parte del descriptor (HOG) con la base de entrenamiento conjuntamente con la frontera (SVM) dentro del espacio que comprende al descriptor el algoritmo se desplaza por la pantalla buscando la coincidencia y posteriormente se realiza una toma de decisión por parte del algoritmo clasificador realizando más actividades como el escalado para así determinar si existe o no un evento positivo.

Figura 29. Análisis de evento para Pistola

En la imagen 29 se puede apreciar como el sistema al no lograr reconocer claramente al ser humano analiza de forma erronea la categorizacion, pero en cambio se indica que el objeto esta siendo reconocido , por esto tiene su representacion en color blanco, forma

parte de la clasificacion que uso como base el clasificador HOG y uno de los primeros que se probo con niveles bajos de entrenamiento con banco de imágenes de 200 a 300 positivas y con un aproximado de 1000 imágenes negativas, los niveles de entrenamiento iniciales fueron de 10 stages o niveles de entrenamiento cuyos resultados no se incluyeron al ser muy deficientes, posterioremente se mejoro el algoritmo con mas niveles y ampliando en numero de muestras positivas.

Figura 30. Análisis para pistola con presencia humana

En la figura 30 se proyecta la detección satisfactoria del objeto en este caso pistola, con la detección positiva de un ser humano, en este caso su rostro que es el de más fácil detección. Así mismo este entrenamiento mostro varios falsos positivos como se puede apreciar en la esquina lateral derecha donde el sistema indica la presencia de un ser humano cuando no lo existe en esa área, ya que el sistema detecta primordialmente un rostro humano, y al encontrarlo omite la búsqueda de la silueta de persona, por esta razón las personas ubicadas en la parte trasera no son detectadas.

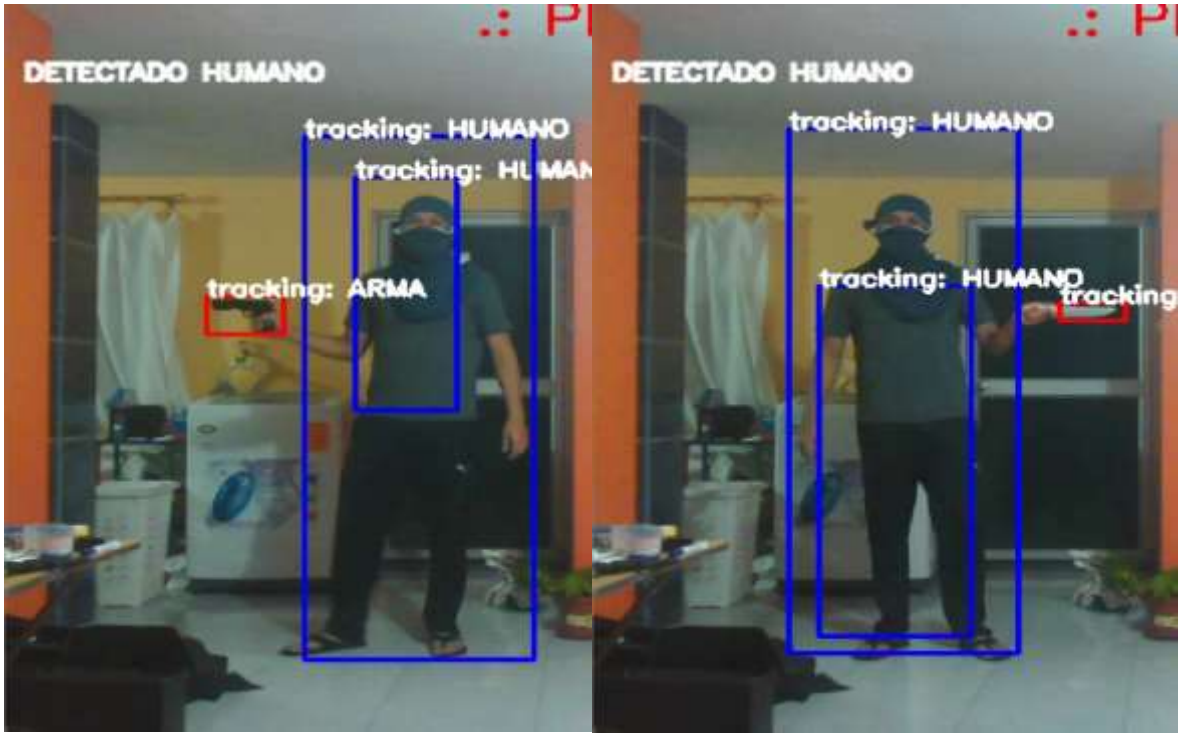

Figura 31. Análisis de Pistola y cuchillo para persona con rostro cubierto

La imagen 31 nos muestra dos recortes de pantalla de análisis que se le hicieron a una persona en la que la identificación de su rostro es de difícil detección o bien está cubierto y no puede ser identificado, para esto el sistema logro identificar correctamente la silueta del ser humano, luego de intentar buscar el rostro, el sistema automáticamente pasa a la siguiente fase pudiendo detectar el arma adecuadamente tanto para la pistola como para el cuchillo. El entrenamiento de este clasificador final utilizado y que entra en parte del análisis de resultados fue de 20 *stages* con cerca de 1000 imágenes positivas y casi 10000 imágenes negativas.

Figura 32. Análisis de Armas cuando es difícil reconocer el rostro

En la imagen 32 se puede apreciar un entorno exterior con degradación propia de la cámara en el que se quiere analizar la presencia de un arma dentro de la imagen, al ser exterior este ámbito puede ser muy variable, al inicio al no encontrar una presencia humana o una silueta contundente que indique su presencia el arma no es considerada como amenaza y su representación dentro de la imagen resultante es considerada como neutral por esto se muestra el resultado de color Blanco, a pesar de que es posible la detección del objeto requerido y que con la evaluación posterior del algoritmo se confirma si el arma detectada corresponde una amenaza o no para el entorno, por esto la imagen contigua indica que ya existe una persona y que además al tener el rostro tapado , al encontrar coincidencia de la persona en base a las partes que logra identificar como cuerpo humano como es el caso de la figura 33 donde en la parte izquierda se puede apreciar cómo está reconociéndose el rostro de la persona mientras que en el lado derecho al no poder encontrar un rostro se detecta la forma del cuerpo humano.

Figura 33. Detección de arma con rostro y silueta de la persona

Tabla 3

Matriz de confusión HOG-SVM Cuchillo

MATRIZ DE CONFUSION CUCHILLO		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	28	76
	Negativos	46	4

Fuente: Adaptado por el autor

Tabla 4

Matriz de confusión HOG-SVM Pistola

MATRIZ DE CONFUSION PISTOLA		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	54	46
	Negativos	30	5

Fuente: Adaptado por el autor

4.1.2. LBP

Mediante las extensiones de *Local Binary Patterns* se realizó pruebas de un gran alcance pues aquí se logran reducir los tamaños de los vectores que se obtienen a partir de las características, este es uno de los procesos más rápidos de entrenamiento, con estos clasificadores se observó que algunos patrones se repiten más que otros dentro de una imagen y así se realiza las transiciones entre cada análisis para así detectar los eventos deseados.

Figura 34. Detección de arma sin presencia humana

En parte izquierda de la figura 34 se observa como el sistema identifica correctamente el objeto y así mismo lo clasifica como neutral al no encontrar la presencia de un ser humano y continua escaneando el sistema hasta que encuentre una figura o reanuda todo el ciclo de análisis, este fue uno de los mejores clasificadores que se pudo crear con un nivel de entrenamiento de 20 stages o niveles y también con 1000 imágenes de muestras

positivas con un total de 9432 imágenes negativas , brindando mejores resultados en la detección con entornos cambiantes como son los ambientes exteriores, en este caso la terraza de un edificio.

Figura 35. Detección de arma con presencia humana

En cambio, al existir la presencia humana como es el caso de la figura 35, el sistema manda la señal de alerta origen de la detección del arma previamente y luego de dar como resultado la presencia humana como un positivo, en este caso con el reconocimiento facial del mismo sujeto. Como alternativa con el objeto de implementar resultados más confiables en la figura 36, se puede observar como el sujeto de análisis al tener el rostro cubierto, es identificado en base a la forma del cuerpo humano de manera satisfactoria.

Figura 36. Detección de arma con rostro cubierto de persona

Tabla 5

Matriz de confusión LBP cuchillo

MATRIZ DE CONFUSION CUCHILLO		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	45	11
	Negativos	30	5

Fuente: Adaptado por el autor

Tabla 6

Matriz de confusión LBP Pistola

MATRIZ DE CONFUSION PISTOLA		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	67	22
	Negativos	26	6

Fuente: Adaptado por el autor

4.1.3. HAAR FEATURES

El uso de las características de tipo HAAR con algoritmos basados en AdaBoost son los más populares y potentes que existen para este tipo de detección por características, su propósito inicial fue el de detección de rostros en tiempo real, este clasificador realiza un análisis de la imagen de entrada en base a un conjunto de características calculadas por cada imagen o frame, luego esta pasa por las distintas etapas para así ser capaz de descartar las porciones de imagen que no correspondan a lo requerido y así poder realizar la detección.

Figura 37. Detección de arma sin interacción de una persona

De igual forma el clasificador HAAR que fue entrenado con un total de 1014 imágenes positivas para la pistola y 946 imágenes negativas para el cuchillo, con las mismas imágenes negativas que fueron alrededor de 10000, se obtuvo los mejores resultados en la detección, tanto para niveles de entrenamiento que partieron desde los 15 stages hasta 20 stages que fue el que se utilizó como cascada final para estos análisis. En la figura 37 se aprecia como el arma es detectada, al no existir presencia humana el arma no se convierte en amenaza, y posteriormente al entrar en escena un ser humano como se aprecia en la figura 38 se presenta la amenaza, pero en este caso como el rostro del sujeto no se puede reconocer el sistema busca la coincidencia más cercana dando como resultado un tercer participante que valida la detección, para posteriormente volver a analizar el ser humano encontrando esta vez la detección del objeto con el reconocimiento facial.

Figura 38. Detección positiva de arma con interacción humana

Como depende en gran parte las configuraciones del programa de acuerdo al entorno en el que se encuentre , mediante la realizacion de otras pruebas para determinar si el sistema esta detectando una silueta de la persona en la que su rostro es de dificil deteccion , se analizo un entorno exterior reduciendo el numero de vecinos cercanos para que el sistema sea capaz de detectar mas facilmente al humano, al hacer esto el sistema encuentra y genera una respuesta mas rapida pero con la posibilidad de aumentar el numero de falsos positivos. Como se indica en la imagen 39 el sujeto de analisis esta de espalda con el arma en su mano, el sitema es capaz de identifcar los valores requridos y tomando en cuenta que para que esto se aposible el humano debe estar claramente identificable ya que los entrenamientos para la busqueda de la silueta de las personas esta en base a posiciones naturales (con los brazos y piernas extendidos sin entrecruzarse).

Figura 39. Detección positiva cuando la persona se encuentra de espalda

Tabla 7

Matriz de confusión HAAR cuchillo

MATRIZ DE CONFUSION CUCHILLO		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	50	11
	Negativos	25	10

Fuente: Adaptado por el autor

Tabla 8

Matriz de confusión HAAR Pistola

MATRIZ DE CONFUSION PISTOLA		Valor de predicción	
		Positivos	Negativos
Valor real	Positivos	72	11
	Negativos	20	6

Fuente: Adaptado por el autor

La estimación cuantitativa de la calidad del detector se la determina utilizando la función de precisión dada por:

$$Presicion = \frac{VP}{VP + FP} * 100\ \%$$

Ecuación 5. Porcentaje de Precisión

Fuente: (Torres, 2018)

En donde: VP es un verdadero positivo y se da cuando se detecta un evento como positivo si existe un evento real de la misma naturaleza y un falso positivo FP detecta un evento como positivo sin la existencia del evento real.

La tasa de verdaderos positivos (TVP) o sensibilidad del detector se cuantifica con la exprtesion siguiente:

$$TVP = \frac{VP}{VP + FN} * 100$$

Ecuación 6. Porcentaje de TVP

Fuente: (Torres, 2018)

Donde: FN o falso negativo se da cuando existe un evento real y el detector no lo detecta como positivo.

Tabla 9

Resumen de Resultados

OBJETO	TIPO	Precision	TVP	VP	FP	FN
PISTOLA	HOG	64.28 %	54 %	54	30	26
	LBP	72.04 %	67 %	67	26	22
	HAAR	78.26 %	72 %	72	20	6
CUCHILLO	HOG	48.27 %	28 %	28	30	16
	LBP	60.00 %	45 %	45	30	11
	HAAR	66.66 %	50 %	50	25	11

En la tabla 9 que corresponde a los resultados de cada evaluación se puede apreciar cuan preciso es cada clasificador en base a todas la imágenes que se analizaron, el más satisfactorio fue el de tipo HAAR con una precisión del 78% para la pistola y un 66% en la búsqueda del cuchillo, aun cuando el clasificador LBO obtuvo el segundo lugar no es del todo confiable porque pese a tener una buena precisión entre el 60 y 72% puede ser una alternativa ya que su entrenamiento es más rápido en comparación con los otros clasificadores, en cambio para este proyecto el empleo de clasificadores HOG-SVM no fue muy satisfactorio ya que fue el escenario con mayores errores en la detección, esto se debió a que las variaciones de entorno, factores de iluminación entre otros no pudieron brindarle la estabilidad requerida para que pueda desarrollar sus características.

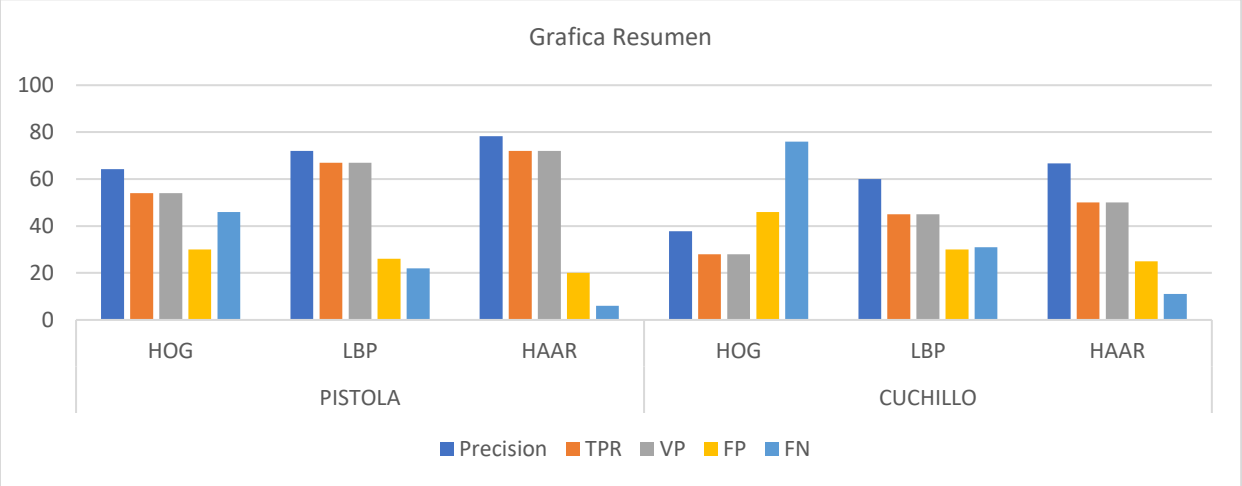

Figura 40. Grafica Resumen

La grafica resumen representa los resultados obtenidos de cada evaluación hecha por el detector y para determinar cuál de ellos es más eficaz, se realiza mediante el análisis de la precisión que al comparar los resultados obtenidos se observa como el detector basado en características de Tipo HOG es el considerado menos preciso en comparación con el detector de tipo HAAR que presenta los mejores resultados como se aprecia mejor en la figura 41 de comparativa de detectores.

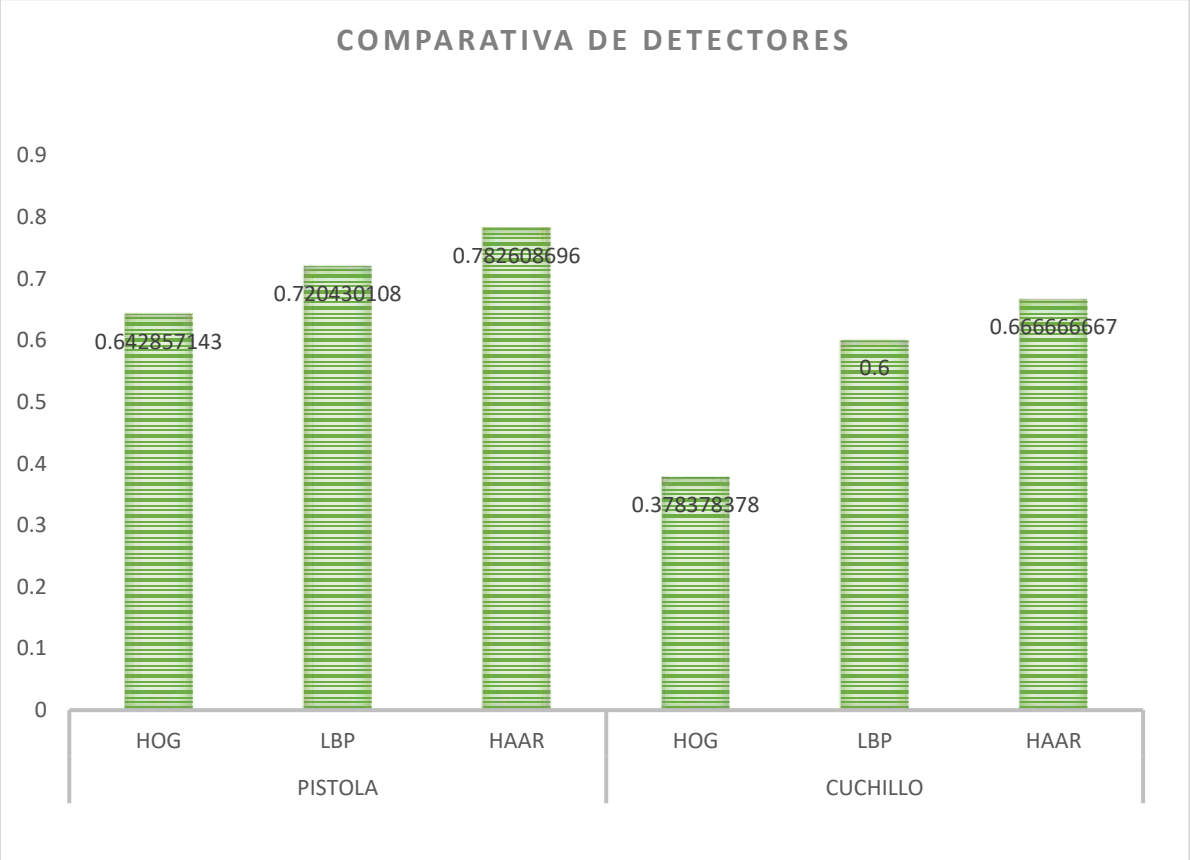

Figura 41. Grafica comparativa

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5. CONCLUSIONES Y RECOMEDACIONES

- Se logro realizar la detección de eventos en tiempo real dentro de un ambiente video-vigilado compuesto por varios entornos en los que se detectó satisfactoriamente a las personas que estaban en posesión de armas, para este proyecto se incluyó únicamente dos objetos de análisis que fueron la pistola y el cuchillo, en las que con la interpretación de varios eventos que incluyeron: Identificación de arma sin presencia humana, en la que el arma no debería ser considerada una amenaza, la identificación de armas con presencia humana , en la que el arma es considerada una amenaza y este emite una alerta visual, y el ultimo evento de análisis que incluye la detección de una persona que tiene su rostro cubierto o de difícil detección, igualmente emitiendo una alerta visual indicando ”PELIGRO”, en todos estos casos fue posible la detección en la mayoría de los entornos con los resultados positivos esperados.
- Con la ayuda de la investigación previa de los conceptos teórico-prácticos sobre procesamiento de imágenes aplicado en la detección e identificación de objetos se pudo diseñar el modelo del *framework* que utilizo nuestro sistema, ya que en base a este diseño se logró implementar un algoritmo propio para la detección más avanzado, que no solo permita la identificación de objetos, sino que va más allá en inteligencia

artificial y sea capaz de discernir si un arma es una amenaza o no, así mismo en base a la optimización de la programación conjuntamente con las librerías de OpenCV sobre Python se pudo implementar en una tarjeta embebida de tipo Raspberry PI 3 model B+, en la que se optimizo su funcionamiento y también se redujo el costo computacional para su correcto funcionamiento sobre la tarjeta.

- Como otro factor importante se logró crear una base de imágenes o banco de imágenes que contienen varias muestras tanto positivas como negativas para cada entrenamiento previo en las que están contenidas las imágenes de entrenamiento del clasificador que corresponden a la pistola y al cuchillo, así como también todas las muestras negativas que se utilizaron un total de 12000 imágenes que pueden ser utilizadas como un banco de imágenes para próximos entrenamientos o para mejoras futuras de este sistema detector.
- A pesar de ser objetos pequeños y de difícil detección en comparación a las de personas o vehículos, que son trabajos previamente realizados, se obtuvieron los siguientes resultados: con una tasa de 78% de precisión para el clasificador de tipo HAAR, frente a una del 72% del clasificador tipo LBP, y en este caso el que nos presentó menores resultados fue el de tipo HOG con un 64%, todos ellos con 20 niveles de entrenamiento del clasificador, un minHitRate de 0.9995 y compuestos por una base de imágenes de aproximadamente 1000 imágenes positivas y 10000 negativas para cada objeto entrenado.

- Para un mejor resultado y en base a experiencias previas de entrenamientos se considera que de imágenes positivas que se ven similares a las condiciones en las que se desea detectar el objeto dan los mejores resultados pues el sistema conoce únicamente las características para el cual fue entrenado, esto quiere decir que si en un sistema de seguridad en el que se requiere una detección más exhaustiva, por ejemplo en un banco o un cuarto de vigilancia se necesita crear un clasificador independiente que contenga como banco de imágenes tanto positivas como negativas al entorno de análisis (en si el cuarto donde está colocada la cámara), de ser posible realizar la captura de imágenes positivas y negativas con la misma cámara que será utilizada para la lectura de datos y realizar capturas con varios ambientes de iluminación , así mismo para lograr clasificadores más fuertes se puede complementar estas imágenes con fondos neutros con combinaciones de colores y difuminaciones que ayudan a realizar entrenamientos más completos con resultados más satisfactorios a la hora de detectar un evento.
- Para reducir el costo computacional que estos sistemas generan en las tarjetas embebidas se recomienda realizar una programación estructurada basada en generación de funciones propias que sean utilizadas bajo demanda, así únicamente se realizan cálculos mediante solicitud y no su ejecución permanente como ocurre normalmente en los sistemas de programación simples.

TRABAJOS FUTUROS

6. PROPUESTAS PARA TRABAJOS FUTUROS

- Tomando como base el presento proyecto se pueden mejorar sus características de desempeño del detector, es necesario ampliar el rango de cobertura en cuanto a rotación y dirección en la que se encuentra el objeto a ser detectado, se puede continuar con este trabajo y desarrollar un clasificador más potente y especifico, ampliando la cantidad de imágenes positivas para el entrenamiento y utilizando el entrenamiento final de este proyecto para aumentar también los niveles de aprendizaje del clasificador en cascada y así obtener un clasificador más fuerte.
- Se puede crear más clasificadores para este sistema y reutilizar el programa detector, ya que no depende del objeto sino del clasificador, juntamente con el empleo de un clúster se puede reducir los tiempos de entrenamiento de varios días hasta solo unas horas, esto es muy beneficioso ya que se pueden realizar más pruebas, niveles de entrenamiento e incluso aumentar la cantidad de muestras positivas y negativas.
- Trabajar con tarjetas embebidas con capacidad de procesamiento en paralelo tales como las Nvidia diseñadas para visión artificial como son la serie Jetson que brinda muchas prestaciones como un gran número de núcleos de procesamiento GPUs, NPUs internas, programación en paralelismo, puertos de mayor velocidad y capacidades específicas para proyectos de visión artificial con muchas librerías ya disponibles listas para utilizar con Python y que brindan soporte en línea desde su propia página.

- Para mejorar este trabajo y para que sea adaptable a sistemas de videovigilancia sean públicos o privados se puede combinar este tipo de proyecto con la investigación de redes neuronales convolucionales (CNN), y mediante otras técnicas de paralelismo mejorar el algoritmo, su ejecución y así lograr más retos en la detección de eventos más complejos.

7. Bibliografía

Aaron Damashek,(s.f.). *"Detecting guns using parametric edge matching"*. (stanford university),obtenido de: http://cvgl.stanford.edu/teaching/cs231a_winter1415/prev/projects/cs231agun.pdf

Albany, c. o. (august, 2017). IEEE *smart world 2017 nvidia city callenge.* san francisco: smart world congress 2017. obtenido de http://smart-city-sjsu.net/aicitychallenge/

Al-shoukry, s. (08 de 2017). *"An automated hybrid approach to detect concealed weapons using deep learning"*. (arpn) obtenido de: http://www.arpnjournals.org/jeas/research_papers/rp_2017/jeas_0817_6268.pdf

ANT, e. (2017). *Transporte seguro (ant).* (ecu911) obtenido de: http://www.ecu911.gob.ec/transporte-seguro/

Bart de decker, j. d. (2013). C*ommunications and multimedia security.* magdeburg, germany: springer.

Beyeler, m. (2017). *Machine learning for opencv.* birmingham, uk: packt publishing.

commons, w. (2011). *rgb channels separation.* china: wikimedia commons.

corp, n. (august de 2017). *deep learning ai*. (nvidia) obtenido de https://www.nvidia.com/en-sg/deep-learning-ai/industries/ai-cities/

Cyganek, b. (2013). *Object detection and recognition in digital images.* poland: wiley.

Davic, w. m. (2011). *Advances in computing and information technology.* chennai, india: springer.

developers, o. s. (2017). *github*. obtenido de: https://github.com/opencv/opencv/tree/master/data/haarcascades

Dimashova, m. (12 de 11 de 2012). *opencv org*. obtenido de community forum for opencv.

Florez, r. f. (2008). *las redes neuronales artificiales, fundamentos teoricos y aplpicaciones.* la coruña: netbibio.

Francisco e, m. c. (s.f.). *inteligencia artificial, modelo, tecnicas y areas de aplicacion.* madrid: thomson.

Gabriel, g. (2018). *opencv 3.x with python by example.* birminham, mumbai: packt.

Greja, m. (01 de 01 de 2016). *"automated detection of firearms and knives in cctv image"* . recuperado el 22 de 02 de 2018, de sensors mdpi: http://www.mdpi.com/1424-8220/16/1/47

Howse, j. (s.f.). *Opencv computer vision with python.* packt open source.

Jahne, b. (1997). *Digital image processing.* san diego, california: university of california.

Jhon m, z. (2004). *Python programming: an introduction to computing science.* usa: tom summer.

Jitendra, p. (2012). *c# programming.* usa: jitendra.

Jo, c.-y. (2008). *Face detection using lbp features.* stanford: cs final project report.

Joseph, h. q. (2015). *Opencv 3 blueprints.* birmingham , uk: packt.

Joyce, f. (2016). *Java programming.* boston , usa: cengage learning.

Juliano, c. (2015). A comparison of haar-like, lbp and hog approaches toc oncrete and asphalt runway detecion in high reolution imagery. *pan-american associacion of computational interdisciplinary sciences-pacis*, 124.

Justin lai, s. m. (s.f.). *“Developing a real-time gun detection classifier”* . (stanford) recuperado el 22 de 02 de 2018 , de http://cs231n.stanford.edu/reports/2017/pdfs/716.pdf

Lopez, a. (2017). *Modelos no holisticos.* barcelona: coursera university.

Mahdi, r. (s.f.). *Creating a cascade of haar-like classifiers: step by step.* auckland: department of computer science.

Martin, j. c. (2012). *Infraestructuras comunes de telecomunicacion en viviendas y efidicios.* editex.

Micha grega, s. l. (05 de 2012). *instreet-aplication for urban photograph localization*. recuperado el 20 de 02 de 2018, de www.researchgate.net/publication/264205083_instreet_-_application_for_urban_photograph_localization/amp

Oakley, j. (2003). *Digital imaging.* new york: cambridge univsersity press.

Pajankar, a. (2017). *raspberry python image processing and programming.* nashik-maharashtra, india: springer.

Piyush modi, p. (april de 2017). *AI for industrial iot & smart infraestructure.* obtenido de https://www.aps.org/units/fiap/meetings/conference/upload/6-2-modi-nvidia.pdf

Ponce, j. (2006). *toward category-level object recognition.* oxford: springer.

Raspberry org, f. (26 de 06 de 2018). *rasoberry.org*. obtenido de https://www.raspberrypi.org/

Reyjavik, 1. i. (2015). *Mathematical morpholofy and its applications to signal and image processing.* iceland: spring.

Rohit kumar tiwari, g. v. (s.f.). *"A computer vision based framework for visual gun detection using harris interest point detector"*. obtenido de https://ac.els-cdn.com/s1877050915014076/1-s2.0-s1877050915014076-main.pdf?_tid=spdf-464c77d9-065a-456d-96d9-66df1f6c9883&acdnat=1519601271_50dbaac80112bc6cae7fe5ef45f9b1ce

Rohit vajhala, m. p. (09 de 2016). *“weapon detection in surveillance camera images”*. Obtenido de : https://www.diva-portal.org/smash/get/diva2:1054902/fulltext02.pdf

Shanker trivedi, s. v. (2017). *A new computing era,nvidia company, deep learning.* obtenido de https://www.nvidia.co.kr/content/apac/event/kr/deep-learning-day-2017/keynote/deep-learning-day-keynote-shanker-trivedi.pdf

Skyler alsever, g. k. (04 de 2017). *“Awed (automatic weapons detection)"*. (wpi) recuperado el 22 de 02 de 2018, de https://web.wpi.edu/pubs/e-project/available/e-project-042717-130517/unrestricted/awed_mqp_final_paper.pdf

Stephen, c. (2016). *Matlab programming for engineers.* boston, usa: bae systems australia.

Thomas b, m. (2012). *Introuduction to video and image processing.* denmark: springer.

Tinku acharya, a. (2005). *Image processing, principles and aplications.* tucson, arizona: wiley interscience.

Torres, j. (2018). *Deep learning, introduccion practica con keras.* barcelona: watch this space.