# Desarrollo de una aplicación móvil con Android Studio y OpenCV para la clasificación de tuercas, tornillos y arandelas

**Victor Andres Carvajal Sierra**

**Id. 000271497**

**Universidad Pontificia Bolivariana – Seccional Bucaramanga**

**Escuela de Ingenierías**

**Facultad de Ingeniería Eléctrica y Electrónica**

**Ingeniería Electrónica**

**2021**

# Desarrollo de una aplicación móvil con Android Studio y OpenCV para la clasificación de tuercas, tornillos y arandelas

**Victor Andres Carvajal Sierra**

**Id. 000271497**

**Proyecto de grado presentado como requisito para optar al título de:**

**INGENIERO ELECTRÓNICO**

**Director del proyecto**

**Luis Angel Silva, PhD**

**Universidad Pontificia Bolivariana – Seccional Bucaramanga**

**Escuela de Ingenierías**

**Facultad de Ingeniería Eléctrica y Electrónica**

**Ingeniería Electrónica**

**2021**

**Tabla de contenido**

## Lista de Tablas

## Lista de Figuras

Universidad Pontificia Bolivariana
SECCIONAL BUCARAMANGA

# RESUMEN GENERAL DE TRABAJO DE GRADO

**TITULO:** Desarrollo de una aplicación móvil con Android Studio y OpenCV para la clasificación de tuercas, tornillos y arandelas.

**AUTOR(ES):** Victor Andres Carvajal Sierra

**PROGRAMA:** Facultad de Ingeniería Electrónica

**DIRECTOR(A):** Luis Angel Silva, PhD

## RESUMEN

En este proyecto se desarrolló una aplicación móvil (DetectApp) usando el IDE Android Studio enlazado a las librerías OpenCV, donde se implementó el algoritmo KNN logrando la clasificación 3 objetos (tuercas, tonillos y arandelas). Inicialmente, se realizó una breve revisión bibliografía del algoritmo KNN y su uso en la visión artificial y el machine learning. El funcionamiento de la aplicación móvil consiste en leer la imagen, convertirla del espacio de color RGB al espacio de color HSV, luego se hallan los umbrales para detectar solo los objetos de interés, se aplica un filtro gaussiano para reducir el ruido y luego se realiza un procesamiento de imagen que extrae características únicas de los objetos, en este caso relacionadas con los primeros seis momentos invariantes de Hu. Posteriormente el algoritmo KNN se encarga de clasificar cada objeto en una clase determinada (tuerca, tornillo, arandela), el cual fue validado mediante pruebas de funcionamiento que consistieron en tomar varias fotografías de los objetos variando la distancia entre la cámara y el objeto, la posición y la rotación de los objetos de donde se obtuvieron buenos resultados que permiten afirmar que este algoritmo puede ser implementado en otro tipo de aplicaciones. El funcionamiento del algoritmo se comprobó usando la matriz de confusión dando como resultado una exactitud mayor al 95% para cada uno de los 3 objetos. Finalmente, se diseñó una interfaz de usuario que permite seleccionar si se desea hacer una clasificación con video en línea, tomar una fotografía o buscar una fotografía en la galería del teléfono.

**PALABRAS CLAVE:**

OpenCV, Visión Artificial, KNN, Clasificación, Android Studio, Java, Machine Learning.

Luis Angel Silva

**V° B° DIRECTOR DE TRABAJO DE GRADO**

Universidad Pontificia Bolivariana
SECCIONAL BUCARAMANGA

## GENERAL SUMMARY OF WORK OF GRADE

**TITLE:** Development of a mobile application with Android Studio and OpenCV for the classification of nuts, bolts and washers.

**AUTHOR(S):** Victor Andres Carvajal Sierra

**FACULTY:** Facultad de Ingeniería Electrónica

**DIRECTOR:** Luis Angel Silva, PhD

### ABSTRACT

In this research project it was developed a mobile application (DetectApp) using the Android Studio IDE linked to OpenCV libraries, where the KNN algorithm was implemented obtaining the classification of 3 objects (nuts, bolts and washers). First a a brief literature review of the KNN algorithm is explained and its use in computer vision and machine learning was performed. The operation of the mobile application consists in image reading, converting the image from RGB color space to HSV color space, then thresholds are found to detect only the objects of interest, a Gaussian filter is applied to reduce noise, and then image processing is performed to extract unique features of the objects, in this case related to the first six invariant Hu moments. Subsequently, the KNN algorithm is in charge of classifying each object in a specific class (nut, screw, washer), which was validated through performance tests that consist in taking several photographs of the objects by varying the distance between the camera and the object, the position and the rotation of the objects from which good results were obtained that allow us to affirm that this algorithm can be implemented in other types of applications. The performance of the algorithm was tested using the confusion matrix resulting in 95% of accuracy for each one of the 3 objects. Finally, an user interface was designed to allow the user to select whether to make a classification with online video, take a photograph or search for a photograph in the phone's gallery.

**KEYWORDS:**

OpenCV, Computer Vision, KNN, Classification, Android Studio, Java, Machine Learning.

Luis Angel Silva

**V° B° DIRECTOR OF GRADUATE WORK**

## Introducción

Los avances tecnológicos que se ven diariamente en nuestra sociedad son cada día más sorprendentes, tenemos avances en todas las ramas de la ciencia que nos benefician de una u otra manera. Estos avances están relacionados con la industria 4.0 la cual aborda desarrollos en diferentes aspectos relacionados con el internet de las cosas, big data, inteligencia artificial, aprendizaje automático entre otros. Dentro del aprendizaje automático se encuentra la visión artificial que permite extraer características propias de uno o varios objetos presentes en una imagen para así poder realizar un procesamiento y clasificación según la aplicación.

La visión artificial aplicada a la industria busca suplir las falencias que el personal humano tiene relacionadas con la capacidad de detección de defectos en el control de calidad de productos. En este aspecto la visión artificial ha logrado aumentar la precisión y agilidad con los que se realizan muchos procesos en el control de calidad relacionados con la clasificación de objetos por su forma, color y tamaño, entre otros. Otra aplicación está relacionada con la seguridad donde es usada para la detección de rostros y detección de movimiento. En la industria de los videojuegos la visión artificial es altamente usada para la creación de juegos con realidad virtual.

En la actualidad los teléfonos desde la gama baja hasta la gama alta traen consigo aplicaciones de visión artificial. El uso más habitual es el desbloqueo facial capaz de reconocer el rostro de la persona dueña del teléfono previamente configurado con una base de datos almacenada en el mismo. Además, las capacidades de procesamiento con la que vienen estos teléfonos móviles son capaces de correr casi cualquier aplicación que se diseñe sin presentar inconvenientes.

Los algoritmos de clasificación usados en la visión artificial son muchos y depende si el aprendizaje es supervisado o no supervisado. Los algoritmos de aprendizaje supervisado

necesitan datos de entrada y datos de salida. Los datos de entrada generalmente se conocen como características y definen parámetros como tamaño, forma y color que caracterizan a un objeto y los datos de salida se conocen como clases a las cuales se asignara cada objeto a clasificar. Entre los algoritmos de aprendizaje supervisado más empleados se encuentran el clasificador de mínima distancia, clasificador bayesiano, KNN, máquinas de soporte vectorial entre otros. Los algoritmos de aprendizaje no supervisado solo necesitan datos de entrada. Entre los algoritmos de clasificación no supervisada más utilizados se encuentran el K-Means.

En este proyecto se propone el desarrollo de una aplicación móvil para la clasificación de tuercas, tornillos y arandelas empleando el algoritmo de clasificación supervisada KNN. La aplicación móvil se implementará en el entorno de desarrollo Android Studio y utilizará las librerías OpenCV para el procesamiento de la imagen, la extracción de características y el diseño del clasificador. La aplicación móvil diseñada dará la facilidad de clasificar diferentes objetos utilizando un equipo móvil como lo es el teléfono celular en sistemas de control de calidad industrial.

# 1. Planteamiento del problema

El sector industrial se ve afectado directamente en sus procesos de control de calidad y producción cuando son manejados por una persona ya que los productos son clasificados por el criterio personal de quien está realizando este trabajo, que adicional se puede ver afectado por el agotamiento físico del mismo o un cambio de turno, lo que hace que el proceso de clasificación pierda objetividad en la selección y que además este sea un proceso lento lo que va a afectar la producción y la calidad del producto final.

Con la llegada de la industria 4.0 el sector industrial ha encontrado un aliado fuerte para la producción y clasificación masiva de productos, la mayoría de los procesos los puede automatizar, lo que aumenta considerablemente la producción y mejora la calidad de los productos, pues todos son clasificados bajo los mismos criterios en todo momento y de una manera muy rápida.

## Formulación del problema

¿Es posible realizar una aplicación móvil usando visión artificial y el algoritmo KNN para la clasificación de 3 objetos?

# 2. Objetivos

## Objetivo General

Implementar una aplicación móvil para la clasificación de tuercas, tornillos y arandelas usando el algoritmo KNN desarrollado en Android Studio enlazado a las librerías OpenCV.

## Objetivos Específicos

- Realizar una breve revisión bibliográfica del algoritmo KNN y su aplicación en la clasificación de objetos.
- Instalar el software Android Studio, enlazarlo con las librerías OpenCV y ejecutar algoritmos básicos de captura y procesamiento de imágenes.
- Crear la base de datos de 150 imágenes de los objetos a clasificar arandelas, tuercas y tonillos.
- Diseñar el algoritmo KNN basado en el clasificador de mínima distancia Euclidiana, empleando la base de datos fotográfica y el lenguaje de programación Java.
- Implementar la aplicación móvil en Android Studio para la clasificación en línea de tuercas, tornillos y arandelas.

## 3. Alcance

El diseño de aplicaciones móviles con visión artificial integrada tiene un campo de acción amplio en la sociedad actual ya que la mayoría de las personas cuentas con un dispositivo móvil lo que hace que quieran acceder a todo tipo de información y posibilidades en el mismo, adicional estos software, algoritmos y librerías en su mayoría son de código abierto lo que facilita el acceso para cualquier persona o empresa que desee ingresar en este mundo y realizar su propio desarrollo.

En este proyecto inicialmente se realizará una revisión bibliográfica sobre la aplicación del algoritmo de clasificación supervisada KNN para la detección y clasificación de objetos presentes en una imagen. La base de datos fotográfica se realizará tomando 50 imágenes por cada objeto (tuerca, tornillo y arandela) para un total de 150 imágenes de las cuales 90 serán empleadas para el diseño del clasificador y 60 para la validación de este. El diseño del clasificador KNN se realizará utilizando las librerías OpenCV y el lenguaje de programación Java. La aplicación móvil se desarrollará en Android Studio y permitirá la clasificación en línea de los objetos seleccionados.

## 4. Marco teórico

### Librerías OpenCV

OpenCV (Open Source Computer Vision), es una biblioteca de código abierto para el aprendizaje automático y la visión artificial. Al ser un producto con licencia BSD (licencia de software libre permisiva, que permite el uso del código fuente en software no libre), facilita a las empresas aprovechamiento y modificación del código.

**Figura 1**

*Logo de las librerías OpenCV*


*Nota:* En la figura se puede observar el logo con el que se conocen las librerías OpenCV, tomado de (Comunidad OpenCV, 2020)

Esta cuenta con más de 2500 algoritmos optimizados, que incluyen aprendizaje automático y visión por computador, que pueden ser usados para detectar y reconocer personas, identificar objetos, detectar movimientos, mejorar y crear imágenes. Esta biblioteca es ampliamente usada por empresas, grupos de investigación y organismos gubernamentales. Esta escrita en C/C++, pero tiene interfaces con Python, Ruby, Matlab, Java y es compatible con Windows, GNU/Linux, Mac OS y Android, está desarrollada con un fuerte enfoque a crear aplicaciones capaces de ejecutarse en tiempo real.

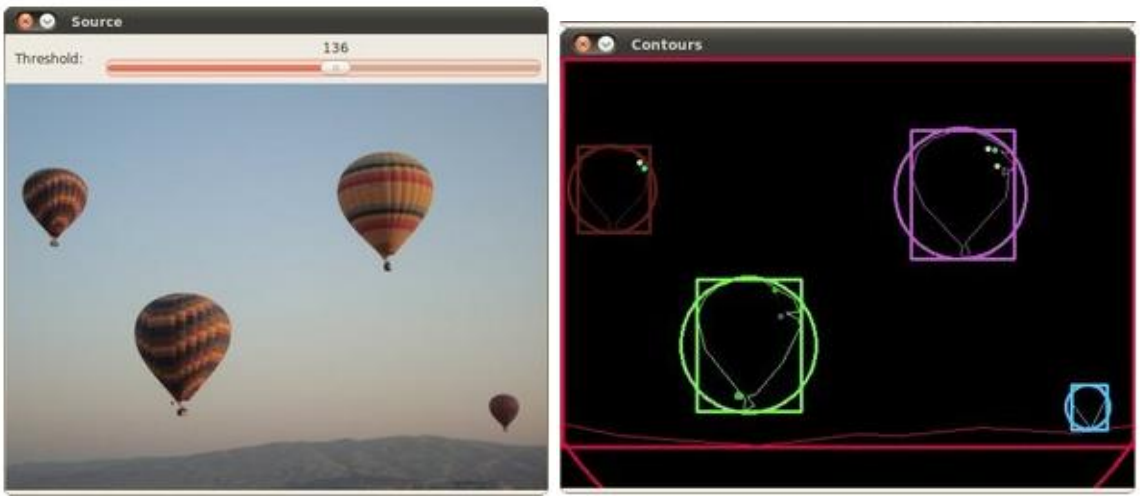
Esta biblioteca está dividida en varios módulos muy potentes ya integrados que son capaces de resolver la mayoría de los problemas de la visión artificial. Se pueden recortar imágenes, mejorarles el brillo, detectar formas, segmentar regiones, reconocer objetos, detectar movimiento entre otras funcionalidades, Estos módulos están optimizados para ejecutarse de forma muy eficaz. (Comunidad OpenCV, 2020)

Para la implementación de la aplicación propuesta en este proyecto se utilizarán los siguientes módulos:

- Core: Estructuras de datos, tipos de datos, y gestión de la memoria.
- Imageproc: Filtrado de imágenes, transformaciones geométricas de imágenes, estructura y análisis de la forma.
- Imagecodecs: Lectura y escritura de imágenes.

**Figura 2**

*Contornos con OpenCV*




*Nota:* En la figura se puede ver el proceso de encontrar contornos y generar figuras en el mismo, esto usando OpenCV. Tomado de (Comunidad OpenCV, 2020).

**Android Studio**

Android Studio es el entorno de desarrollo integrado (IDE), basado en IntelliJ IDEA de la compañía JetBrains para la plataforma Android, fue desarrollado por Google para el desarrollo de aplicaciones móviles Android y reemplazo a eclipse como el IDE oficial, además tiene licencia de software libre. **(Google, 2020)**

**Figura 3**

*Logo Android Studio*

*Nota.* En la figura se puede observar el logo de Android Studio. Tomado de (Google, 2020)

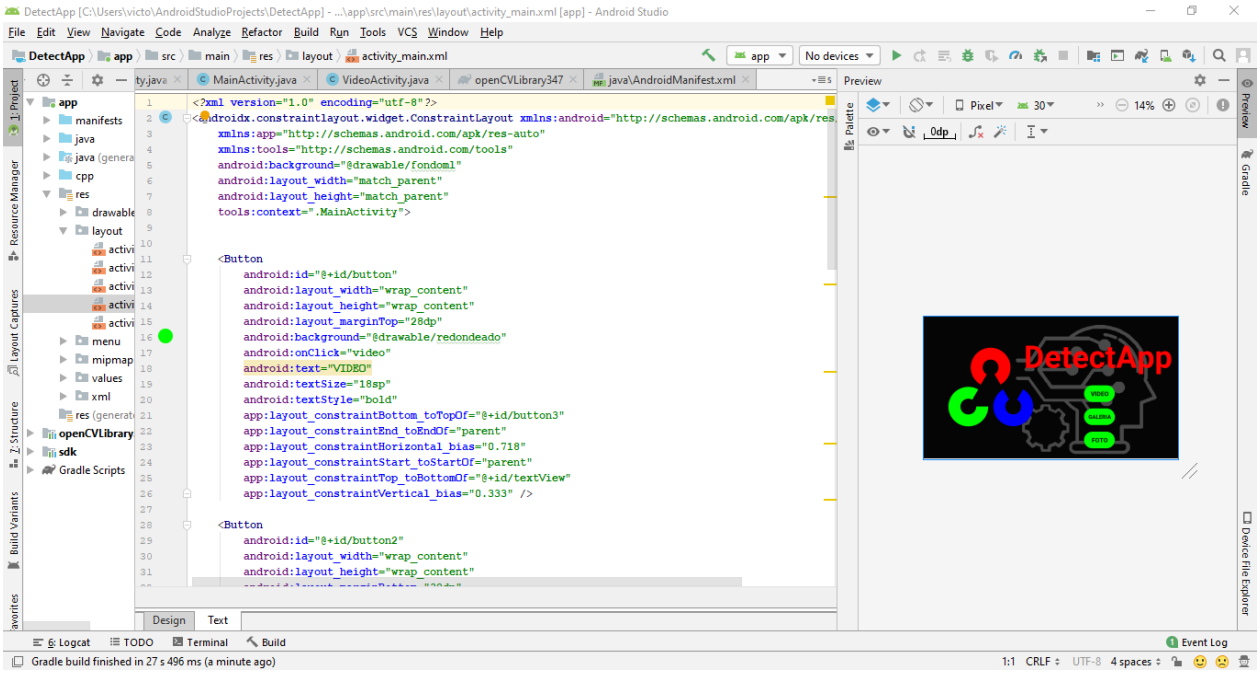
Algunas de sus principales características son:

- Entorno unificado para desarrollar aplicaciones en cualquier dispositivo y resolución.
- Integración de la herramienta Gradle encargada de gestionar y automatizar la construcción de proyectos, como pueden ser las tareas de testing, compilación o empaquetado
- Permite la importación de proyectos realizados en el entorno Eclipse, que a diferencia de Android Studio (Gradle) utiliza ANT
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de aplicaciones comunes y también importar código de muestra
- Compatibilidad con C++ y NDK

- Editor de diseño que muestra una vista previa de los cambios realizados directamente en el archivo XML.
- Alertas en tiempo real de errores sintácticos, compatibilidad o rendimiento antes de compilar la aplicación, lo que hace más eficiente el desarrollo.

**Figura 4**

*Interfaz de usuario Android Studio*




Nota: En la figura se observa la interfaz de usuario en el IDE Android Studio.

Se eligió este software por ser el IDE oficial de Android y por la cantidad de informacion que se encuentra en la web acerca del su manejo. Es posible escribir su código en Kotlin o en Java, sin embargo, en este proyecto se eligió el lenguaje Java pues se tiene un conocimiento previo sobre este, también se debe tener en cuenta que el soporte que brinda Google para este IDE lo hace aún más solicitado en ámbito laboral y educativo.

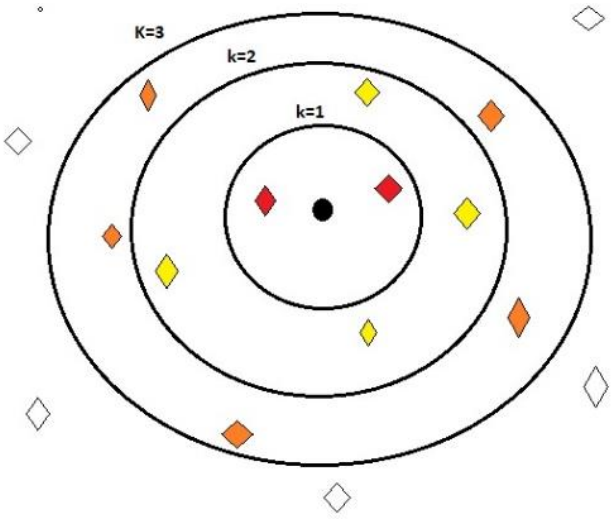
**Clasificador KNN**

El algoritmo K-vecinos cercanos o KNN (K- Nearest Neighbord) es un algoritmo de tipo supervisado. En su forma más sencilla, consiste en clasificar uno o varios datos de entrada según su proximidad con los datos de entrenamiento, esta proximidad se calcula hallando la distancia entre los datos de entrenamiento y los datos de entrada donde los objetos más cercanos serán clasificados o no según el valor de K (vecinos más cercanos), este valor de K depende de cada caso no hay un valor estándar para el diseño del algoritmo. Para el cálculo de la distancia se puede usar la distancia Euclidiana, Chebyshev, Manhattan o Kullback-Lebler. (Kuang & Zhao, 2009)

En la figura 5 se puede observar el funcionamiento del algoritmo KNN que según el valor de K clasifica o no objetos como pertenecientes a una clase, en este caso la clase es el punto en el centro, como se observa en la figura entre más grande es K más objetos clasifica como pertenecientes a la clase y los que están fuera del valor de K no son clasificados en ninguna clase.

**Figura 5**

*Funcionamiento de la clasificación del vecino más cercano*



En el desarrollo de este proyecto se usará la distancia Euclidiana, la cual se calcula con la siguiente ecuación. (Kuang & Zhao, 2009)

$$d_{(p,q)} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \quad (1)$$

d = distancia entre dos puntos

p, q =puntos de referencia

La distancia euclidiana se define como la distancia entre dos puntos en el espacio y se deduce usando el teorema de Pitágoras, esta distancia se puede calcular entre puntos de n-dimensiones.

**Espacio de Color HSV**

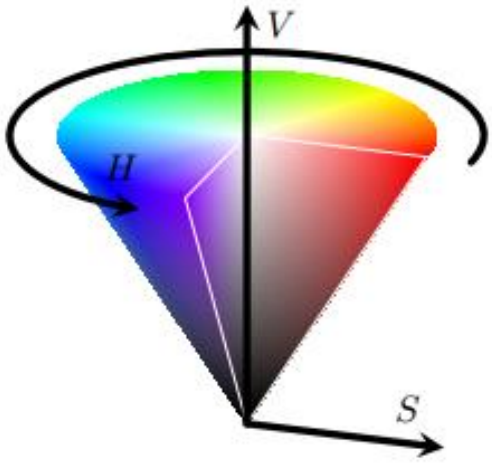
El espacio de colores HSV (del inglés Hue, Saturation, Value - Matiz, Saturación,Valor), el ser humano reconoce mayormente los colores por atributos perceptuales de luminancia o intensidad, saturación y matiz. HSV se adapta mejor a la forma sobre como el ojo humano percibe los colores, permitiendo que a partir de un color se puedan obtener varias intensidades de un mismo matiz. (Estrebou, 2021)

Los componentes de este espacio de colores son: matiz (hue), saturación (saturation), valor (value). El matiz es la esencia del color y corresponde a un Angulo cuyos valores varían en el rango de H: [0-360] grados. La saturación o pureza del color oscila entre S: [0-100], siendo un valor de distancia radial; mientras que el valor indica el brillo y adopta el rango de V: [0-100]. (Estrebou, 2021)

OpenCV tiene la función *Core.inRange(),* donde se indican los umbrales superior e inferior del color buscado.

**Figura 6**

*Espacio de color HSV*




*Nota:* En la figura se observa la geometría del espacio HSV. Tomado de (Estrebou, 2021).

**Filtro Gaussiano**

El filtro gaussiano es un filtro lineal de suavizado que se usa para eliminar pequeños detalles en una imagen, disminuyendo la nitidez y aumentando de borrosidad de esta, tiene un suavizado bastante uniforme. (Kapur & Thakkar, 2015)

En la figura 7 es posible visualizar el cambio de una imagen aplicando un filtro gaussiano, la imagen de la izquierda es la original y la imagen de la derecha tiene el filtro gaussiano aplicado, la imagen con el filtro se ve un poco borrosa en comparación con la original.

**Figura 7**

*Filtro Gaussiano*

*Nota*: En la figura se observa una imagen antes y después de aplicar el filtro Gaussiano Tomado de (Kapur & Thakkar, 2015)

Para aplicar el filtro gaussiano a una imagen procesado en OpenCV se usa la función *Imageproc.GaussianBlur()*, allí se deben indicar el valor de la matriz que se usara en el filtro.

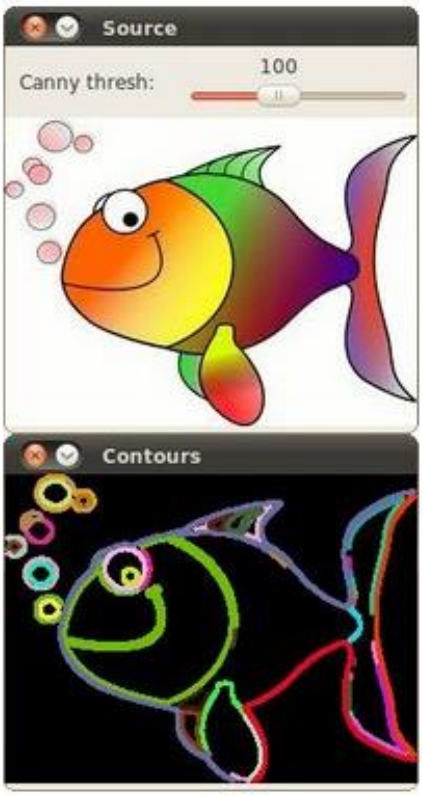
**Contornos**

Los contornos corresponden a transiciones bruscas en la intensidad de la imagen, estas transiciones con la misma intensidad generan puntos continuos que marcan el límite de un objeto, para encontrar los contornos en una imagen OpenCV tiene la función *findContours(),* esta función devuelve los contornos que se encuentren en la imagen, es posible seleccionar si todos los hallados  o si solamente el contorno exterior. (Comunidad OpenCV, 2020)

En este proyecto al ser los objetos el contorno más grande presente en la imagen usaremos únicamente el contorno exterior.

**Figura 8**

*Contornos OpenCV*




*Nota:* En la figura se puede observar el contorno de una imagen hallado con OpenCV. Tomado de (Comunidad OpenCV, 2020)

**Momentos Invariantes de Hu**

Estos momentos pueden ser considerados como un promedio ponderado de los píxeles de una imagen. Hu computo sus invariantes utilizando los momentos geométricos, los cuales son variantes a la rotación y al escalamiento. Los momentos geométricos se definen como (Gonzalez & Woods, 2008):

$$\mu_{pq} = \iint (x - \bar{x})^p (y - \bar{y})^q f(x, y) dx dy \quad (2)$$

Donde $\mu_{pq}$ es el momento geométrico de orden (p+q), $f$ (x, y) es el valor del píxel en la posición (x, y) de la imagen y $(\bar{x},\bar{y})$ es el centroide de esta. Partiendo de estos momentos podemos obtener $\eta_{pq}$, un momento de orden (p + q) que es invariante al escalamiento:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{1+\frac{p+q}{2}}} \quad (3)$$

Utilizando $\eta_{pq}$, Hu logro un conjunto de momentos invariantes, todos con un grado menor o igual a 3. Sus siete momentos invariantes a rotación, traslación y escala son:

***h* 1** = η20 + η02

***h* 2** = $(\eta 20 + \eta 02)^2 + 4(\eta 11)^2$

***h* 3** = $(\eta 30 - \eta 12)^2 + (3\eta 21 - \eta 03)^2$

***h* 4** = $(\eta 30 + \eta 12)^2 + (\eta 03 + \eta 21)^2$

***h* 5** = $(\eta 30 - 3\eta 12)(\eta 03 + \eta 12)[(\eta 30 + \eta 12)^2 - 3(\eta 03 + \eta 21)^2] + (3\eta 21 - \eta 03)(\eta 03 + \eta 21)[3(\eta 30 + \eta 12)^2 - (\eta 03 + \eta 21)^2]$

***h* 6** = $(\eta 20 - \eta 02)[(\eta 30 + \eta 12)^2 - (\eta 03 + \eta 21)^2] + 4\eta 11(\eta 30 + \eta 12)(\eta 03 + \eta 21)$

***h* 7** = $(3\eta 21 - \eta 03)(\eta 30 + \eta 12)[(\eta 30 + \eta 12)^2 - 3(\eta 03 + \eta 21)^2] - (\eta 30 - 3\eta 12)(\eta 03 + \eta 21)[3(\eta 30 + \eta 12)^2 - (\eta 03 + \eta 21)^2]$.

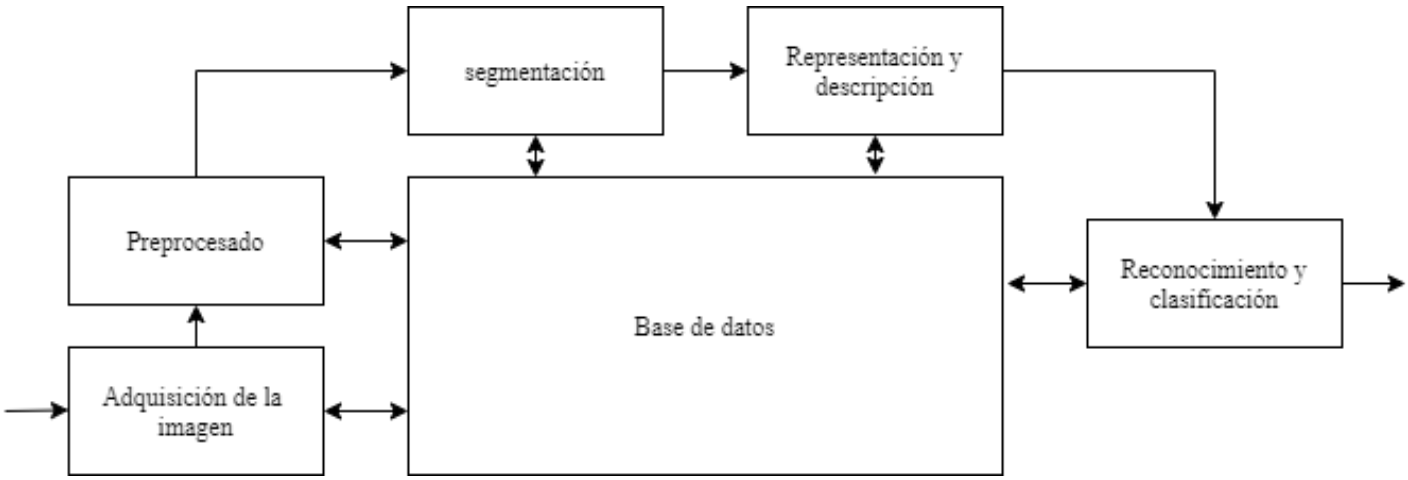
OpenCV tiene la función *Imgproc.moments()* que permite encontrar los momentos centrales para luego normalizarlos y hallar los momentos invariantes de Hu, para hallar estos momentos primero se debe hallar el contorno de los objetos.

**Etapas de una Aplicación con Visión Artificial**

Todas las aplicaciones con visión artificial tienen su especificación, sin embargo, existen unas etapas básicas que estas aplicaciones debe llevar. (Dueñas, 2009) En el desarrollo de este proyecto estas serán tenidas en cuenta para mejorar los resultados y agilizar el proceso.

**Figura 9**

*Diagrama de etapas de la visión artificia*




*Nota*. En la figura se puede observar el diagrama común de un sistema de visión artificial. Tomado de (Dueñas, 2009)

Cada una de estas etapas en la visión artificial se describe a continuación:

1. Adquisición de la imagen: tomar la fotografía que se usara en el proceso.
2. Preprocesamiento: Realizar un proceso previo de la imagen para mejorar su calidad tal como reducir el ruido, mejorar la iluminación entre otros.
3. Segmentación: Esta fase es la que se lleva el mayor tiempo en todo el proceso pues acá es donde se centra toda la informacion de la imagen, donde se decide cuáles son las características únicas para clasificar la imagen.

4. Representación y descripción: En esta fase se calculan los parámetros extraídos de la imagen para alimentar la base de datos y su posterior reconocimiento.
5. Reconocimiento y clasificación: Esta es la fase final del algoritmo en la cual se toma una decisión acerca de donde se va a clasificar uno o más objetos de la imagen, realizando una comparación de la características de la imagen de entrada con las características de la base de datos.

## 6. Metodología

Para la realización de este proyecto se tendrán en cuenta 6 fases, las cuales se relacionan a continuación:

Fase 1: Revisión bibliográfica. Se realizará una breve revisión bibliográfica de trabajos que hallan usado la visión artificial y el algoritmo KNN para la clasificación de objetos presentes en una imagen. (Aporta a los objetivos específicos 1, 3 y 4).

Fase 2: Manejo del software Android Studio: Se instalará el software y se desarrollaran aplicaciones básicas. (Aporta a los objetivos específicos 2, 3, 4 y 5).

Fase 3: Enlace del software Android Studio y las librerías OpenCV: Se instalarán las librerías OpenCV y se enlazarán con el software Android Studio para la captura y procesamiento básico de imágenes. (Aporta a los objetivos específicos 3, 4 y 5).

Fase 4: Toma de datos: Se creará la base de datos fotográfica para el diseño del clasificador KNN. (Aporta a los objetivos específicos 3).

Fase 5: Diseño del clasificador: Diseñar e implementar el algoritmo KNN en el software Android Studio, teniendo en cuenta la base de datos fotográfica de los objetos a clasificar. (Aporta a los objetivos específicos 4 y 5).

Fase 6: Resultados. Entregar la aplicación móvil funcional, además de realizar el informe final del trabajo. (Aporta al objetivo específico 5).

**Fase 1: Revisión Bibliográfica Del Algoritmo KNN**

Al realizar una breve revisión bibliográfica sobre trabajos realizados con temas asociados, se destacan los siguientes:

1. Elías García Santillán, 2008, *Detección y clasificación de objetos dentro de un salón de clases empleando técnicas de procesamiento digital de imágenes*, Universidad Autónoma Metropolitana, este proyecto realiza una detección y reconocimiento de objetos dentro de un salón de clases usando el procesamiento de imágenes, para esto se tuvo en cuenta características propias de los objetos a detectar, se usa el software Matlab para diseñar el programa y el algoritmo KNN para realizar la clasificación de los objetos. (Santillán, 2008)

   En este proyecto se puede observar el buen funcionamiento que tiene el algoritmo KNN en la clasificación de objetos, además se puede evidenciar que las características escogidas del objeto a clasificar son de gran importancia y se verán reflejadas en la calidad del clasificador.

2. Juan Carlos Miranda, 2018, *clasificación automática de naranjas por tamaño y por defectos utilizando técnicas de visión por computadora*, universidad nacional de asunción, El proceso realizado en este proyecto fue proponer una metodología para lograr clasificar naranjas por su tamaño y sus defectos usando la visión artificial, el proceso consistió en crear una base de datos con imágenes de naranjas de todo tipo para luego clasificarlas y poder realizar el entrenamiento de los algoritmos usados, se usó el algoritmo de MSV y el KNN para el proceso de clasificación. (Miranda, 2018)

La segmentación de imágenes es una etapa muy importante en la visión artificial, ya que en esta etapa se toman la características propias de los objetos que se usaran en el clasificador para tomar una decisión.

3. Aida Alexandra Granda Cárdenas, 2019, *comparativa de extractores de características para clasificación de rostros, escuela politécnica nacional*, este proyecto realiza una comparativa de 2 métodos automáticos para la extracción de características de imágenes usando la visión artificial para clasificar rostros, estos métodos son una red neuronal convolucional y el otro esta misma red pero agregando una capa de klusterizacion con el algoritmo k-Means, para la clasificación se entrenaran los algoritmos MSV y el KNN, posteriormente se evaluara cuál de los dos algoritmo realiza una mejor clasificación. (Cardenas, 2019)

   El algoritmo KNN presento un mejor funcionamiento en comparación con el algoritmo MSV al realizar la clasificación de rostros, esto nos da un antecedente importante para usar este algoritmo en el proyecto planteado.

4. Sol Angel Rodríguez Vázquez, Andy Vidal Martínez Borges, Juan Valentín Lorenzo Ginori, 2016, *Clasificación de células cervicales mediante el algoritmo KNN usando rasgos del núcleo*, Universidad de las Ciencias Informáticas, los cambios en la morfología de la celular del cuello uterino pueden alertar sobre patologías frecuentes en la mujer como lo es el cáncer de cuello uterino, por esto es necesario desarrollar un método que permita realizar un análisis practico de estos cambios para detectar a tiempo este tipo de patologías, con la ayuda de la visión artificial y el clasificador KNN se busca encontrar estas anomalías creando una base de datos de celular normales para comparar y encontrar si hay o no anomalías. (Vázquez, Martínez , & Lozano Ginori, 2016)

El uso del algoritmo KNN para clasificar imágenes se puede diseñar usando algunas métricas de distancia, en este proyecto se usó la distancia euclidiana y la manhattan, la distancia euclidiana fue la elegida para desarrollar el proyecto planteado. Aunque en este proyecto la distancia euclidiana no fue la mejor estuvo muy cerca de la manhattan que tuvo el mejor desempeño, sin embargo, este resultado puede variar por la cantidad de muestras que se tomen en la base de datos y del tipo de características a utilizar.

5. Tran Son Hai , Le Hoang Thai, Nguyen Thanh Thuy, 2015, *Facial Expression Classification Using Artificial Neural Network and K-Nearest Neighbor*, las expresiones faciales tienen un papel importante en evaluación de los sentimientos, intenciones y características de una persona, estas expresiones tienen un papel importante en la iteración persona-maquina, para lograr esta iteración se trabajara una red neuronal para extraer las características de las expresiones y luego se clasificaran con el algoritmo KNN. (Hai, Hoang Thai, & Thanh Thuy, 2015)

   El porcentaje de expresiones clasificadas fue del 92.38% lo que nos muy buen antecedente que el algoritmo KNN funciona efectivamente en la clasificación de imágenes usando la visión artificial.

6. Martha R. Quispe Ayala, Krista Asalde Alvarez, Avid Roman Gonzalez, *Clasificación de Imágenes Usando Técnicas de Compresión de Datos*, Universidad Nacional San Antonio Abad del Cusco, en este proyecto se propuso un método para clasificar imágenes basado en dos técnicas de comprensión de datos JPEG y ZIP, para la clasificación se usaron los algoritmos supervisados KNN y MSV, además el algoritmo no supervisado K Means, para el desarrollo se creó una base de datos con la imágenes y se realizaron las pruebas de

clasificación correspondientes a cada algoritmo, arrojando como resultado que la mejor clasificación la tuvo el algoritmo supervisado KNN. (Quispe-Ayala, Asalde-Alvarez , & Roman-Gonzalez)

Inicialmente se planteó usar para este proyecto el algoritmo no supervisado K-Means sin embargo en el transcurso del proyecto se identificó que con este método no se obtendrían los resultados esperados ya que este clasifica todos los datos de entrada en alguna de las clases indicadas, es decir no excluye ningún objeto en la clasificación, mientras que el KNN al ser supervisado le introducimos los parámetros de clasificación y todo lo que no esté dentro esos parámetros no es clasificado en ninguna clase.

7. A. Cazorla, F.J. Olmo y L. Alados-Arboledas, *Estimación de la cubierta nubosa en imágenes de cielo mediante el algoritmo de clasificación KNN*, cundo obtenemos imágenes del cielo podemos determinar la cubierta nebulosa esto justificado por el hecho de las influencia de las nubes sobre la radiación recibida en la superficie, es común que se use el umbral sin embargo cuando se está cerca al sol no es adecuado usar este método, por eso se diseñara un método de clasificación para resolver este problema y obtener una estimación correcta, para esta clasificación se creara una base de datos y se entrenara el algoritmo para luego comprobar si el algoritmo produce mejores resultados que la umbralización. El algoritmo arrojo buenos resultados en varias condiciones sin embargo el algoritmo no es capaz de clasificar ciertos tipos de nubes, aunque esto se puede corregir aumentando el número de imágenes de entrenamiento. (Cazorla, Olmo, & Alados-Arboledas, 2005)

   Al diseñar un algoritmo de clasificación se debe realizar una base de datos con la cantidad de imágenes necesarias para lograr abarcar la mayor variedad de condiciones

posibles que puedan llegar a tener los objetos, con esto aumentaremos la eficiencia del algoritmo clasificador, en este caso el algoritmo KNN.

8. Dr. Wilfrido Gómez Flores, *Reconocimiento de objetos en fotografías,* el reconocimiento de imágenes con visión artificial es un reto muy grande ya que no existe un sistema artificial que reconozco todo, a pesar de ello sus campos de acción son muchos tales como: medicina, robótica, seguridad, entre otros. muchos de estos problemas presentados en estas ramas se han solucionado combinando métodos para diseñar una solución, en la clasificación de imágenes siempre se sigue un proceso básico. Adquisición de imágenes, preprocesamiento, segmentación, extracción de rasgos y clasificación, siguiendo el método de clasificación de imágenes y usando el algoritmo KNN en Matlab se clasificaron 4 clases de objetos. (Flores, 2015)

   En este proyecto se identificaron las fases que debe llevar todo proceso de clasificación de imágenes con visión artificial para obtener un buen resultado y también evitar hacer procesos innecesarios o que tal vez no den el resultado esperado, además se usaron 2 objetos que se usaran en el proyecto planteado, el tornillo y la arandela.

9. Joyce Daniela Aramayo Salgueiro, Alberto de la Piedad Gascueña, Eduardo Sánchez Muñoz, *Procesado de imágenes para plataformas móviles,* Universidad Complutense de Madrid, la tecnología móvil está en pleno desarrollo hay variedad de marcas y de sistema operativos, este proyecto desarrolla en la plataforma Android una aplicación capaz de reconocer prendas de vestir usando visión artificial y OpenCV. (Salgueiro, De la piedad Gascueña, & Sanchez Muñoz, 2013)

   En el proyecto se utilizaron las librerías OpenCV y el IDE Android Studio para lograr la clasificación de prendas de vestir, por lo que se puede tomar como base en el

desarrollo de este proyecto, pero clasificando tuercas, tornillos y arandelas y usando el algoritmo KNN para realizar la clasificación.

10. Jaime Blanco Alambiaga, *desarrollo de una aplicación móvil para le detección y clasificación de hojas de árboles*, Universidad Politécnica de Valencia, este trabajo desarrolla una aplicación móvil de visión artificial usando Android Studio y OpenCV para clasificar 6 tipos de hojas según su morfología, el algoritmo usado en la clasificación se base en calcular las medias de cada parámetro y crear un prototipo para comparar con los datos de entrada y así tomar una decisión. (Alambiaga, 2015)

    En este proyecto se observa el proceso para crear una aplicación móvil con visión artificial usando el IDE Android Studio y las librerías OpenCV, aunque se usa un algoritmo diferente el proceso a seguir para desarrollar la aplicación es muy similar.

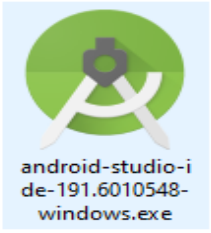
**Fase 2: Instalación del Software Android Studio**

El software para instalar es Android Studio el cual se descarga de la página oficial del desarrollador. (Google, 2020)

La versión usada en el desarrollo del proyecto es la 3.5.3, luego de descargar el instalador se procede a instalar.

**Figura 10**

*Instalador de Android Studio*




*Nota:* En la figura se observa el instalador de Android Studio cuando se descarga en el equipo.

Para el funcionamiento del IDE se debe tener en cuenta lo requisitos técnicos mínimos que necesita el equipo:

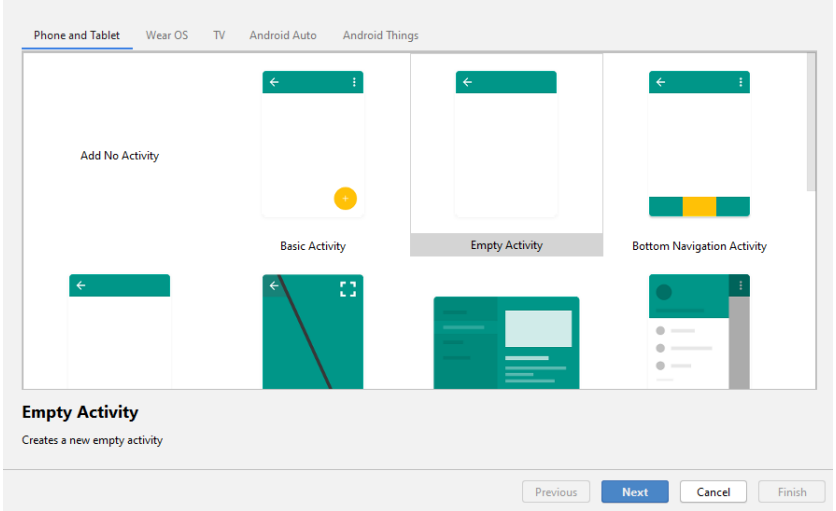
- Windows 7/8/10 (64 bits), Mac OS X o superior, Linux GNOME o KDE
- 4 GB de RAM mínimo, 8 GB RAM recomendada
- 2 GB de espacio en el disco, 4 GB recomendado.
- Resolución mínima de 1280 x 800

Cuando se termina la instalación y se ejecuta Android Studio por primera vez este descarga e instala otros componentes necesarios para su funcionamiento, al terminar todo el proceso de instalación y configuración se puede iniciar con un primer proyecto que permite seleccionar si

será un proyecto en blanco o también es posible seleccionar alguna de las plantillas que proporciona el IDE para iniciar una nueva aplicación, estas opciones se observan en la figura 11.

**Figura 11**

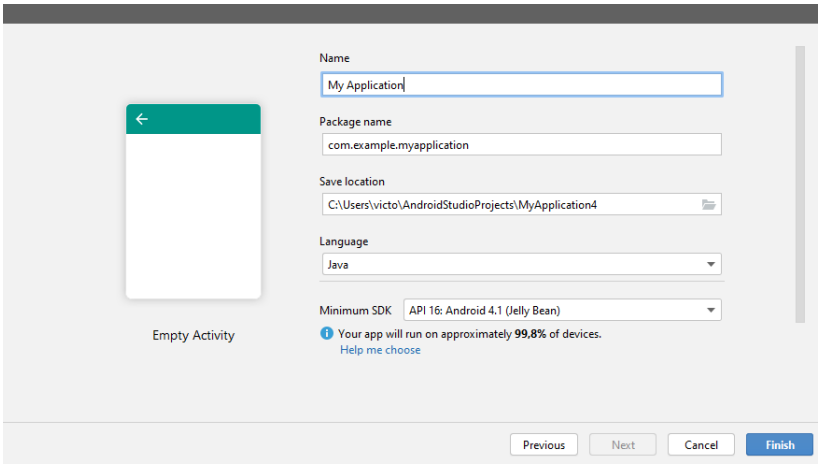
*Inicio Android Studio*




Luego de seleccionar la plantilla con la que se iniciará el proyecto se debe dar un nombre a la aplicación, seleccionar la carpeta donde se guardará, seleccionar el lenguaje de desarrollo y las versiones de Android para las cuales estará disponible, como se observa en la figura 12.

**Figura 12**

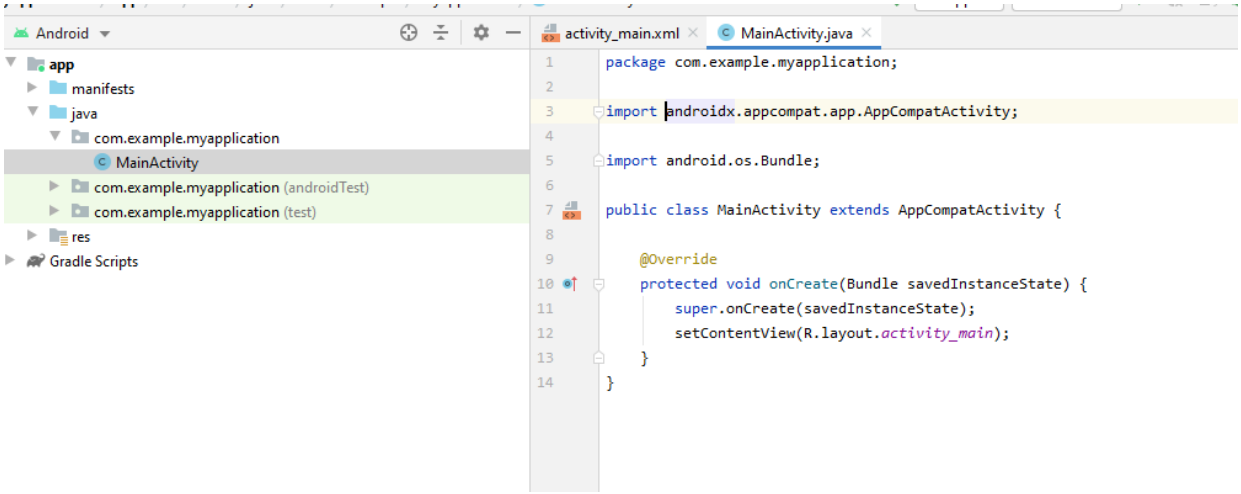
*Datos para la nueva aplicación.*



Luego de llenar estos datos de da click en finish para que se inicie el entorno de trabajo.

**Figura 13**

*Entorno de trabajo Android Studio*

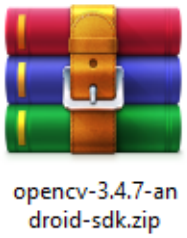



**Fase 3: Enlace del software Android Studio y las librerías OpenCV**

Al terminar la instalación de Android Studio se debe descargar las librerías OpenCV desde la página oficial. (Comunidad OpenCV, 2020) allí se selecciona la versión para Android ya que cada plataforma tiene su librería, en este proyecto se usará la versión 3.4.7.

**Figura 14**

*Archivo con las OpenCV*



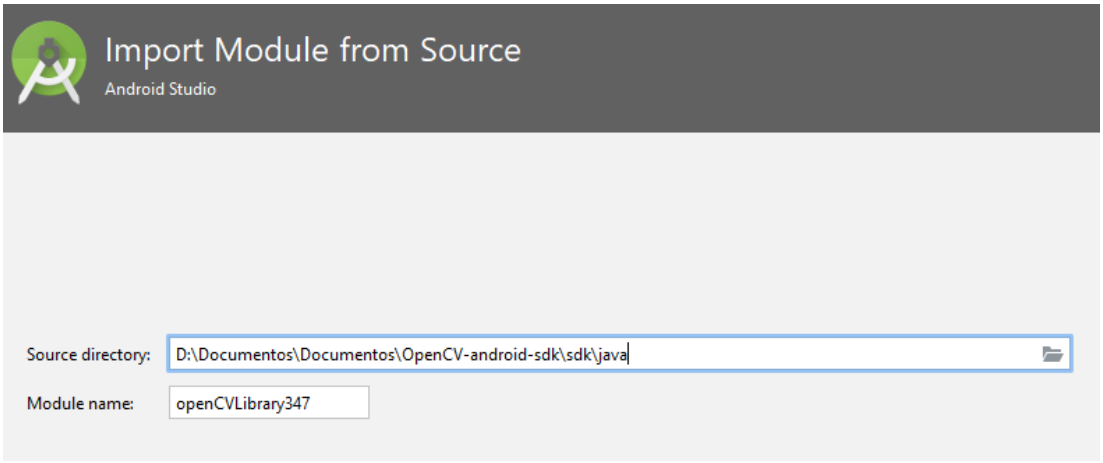

*Nota:* En la figura se observa el archivo .zip que se descarga de la página oficial con las librerías OpenCV.

Luego de descargar y descomprimir las librerías se debe abrir Android Studio para iniciar a enlazar dichas librerías al IDE, se debe seguir los siguientes pasos:

1. Importar OpenCV a Android Studio: Desde archivo / Nuevo / Importar módulo, elegir la carpeta sdk/java en el archivo OpenCV descomprimido, en la figura 12 se observa la ruta para buscar las librerías e importarlas, aunque esta depende de en qué ubicación del equipo se guardó la carpeta descomprimida.

**Figura 15**

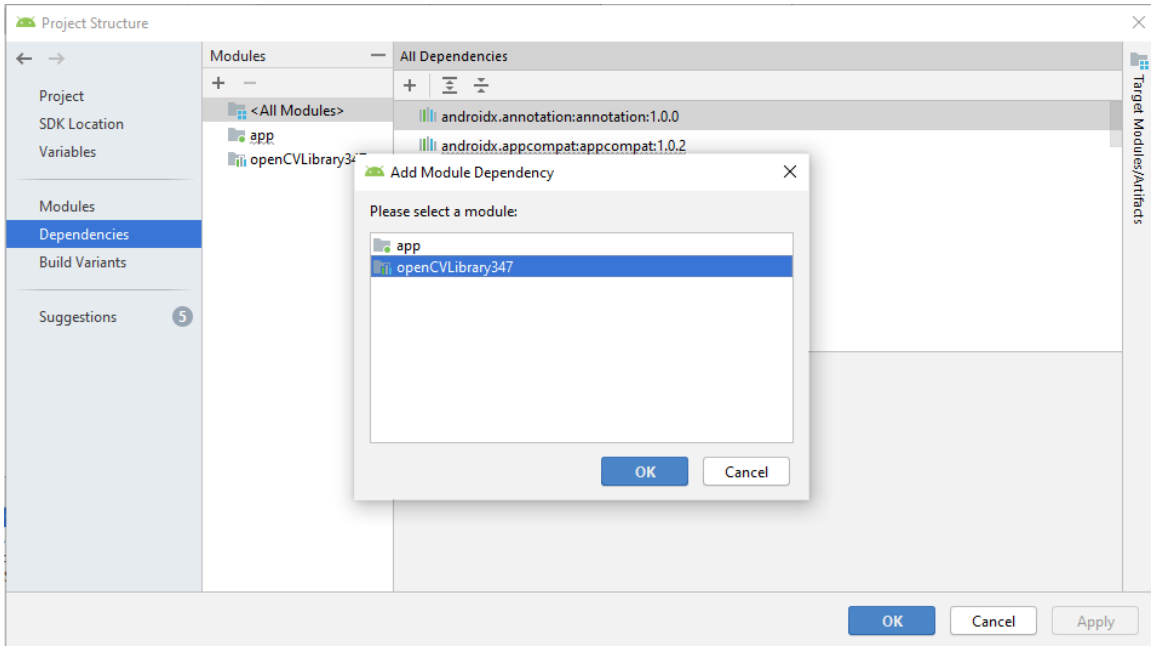
*Importar el módulo OpenCV a Android Studio.*




2. Agregar la dependencia del módulo: Esto se hace en la ruta Aplicación / Configuración del módulo y seleccionar la pestaña Dependencias. hacer clic en el icono + en la parte inferior, elegir Dependencia del módulo y seleccionar el módulo OpenCV importado, en la figura 14 se observa el módulo dependencias

**Figura 16**

*Configuración del módulo dependencias*




3. Crear una nueva carpeta en JNI folder con el nombre jniLibs que tenga la ruta src/main, luego copiar y pegar los archivos que están en la carpeta descomprimida de OpenCV en la ruta sdk/native/libs.

Luego de realizar este proceso las librerías estarán enlazadas con Android Studio y podemos iniciar las pruebas de procesado de imágenes.

***Pruebas de enlace entre las librerías OpenCV y Android Studio***

Posterior a realizar el enlace entre Android Studio y las librerías OpenCV se realizaron algunas pruebas de procesamiento de imágenes, estas pruebas consistieron en visualizar imágenes en escala de grises y también en aplicar el filtro Canny, este procesamiento se realizó con video en vivo.

En la figura 15 se puede observar el código utilizado para visualizar una imagen en escala de grises con la función Imgproc.*cvtColor*(), disponible en las librerías OpenCV.

**Figura 15**

*Código de Android Studio*

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode==100){
        Uri imageuri=data.getData();
        try {
            InputStream inputStream=getContentResolver().openInputStream(imageuri);
            bitmap= BitmapFactory.decodeStream(inputStream);
            Size matrix=new Size( width: 7, height: 7);
            Mat tmp = new Mat(bitmap.getWidth(), bitmap.getHeight(), CvType.CV_8UC1);
            Mat tmp1 = new Mat(bitmap.getWidth(), bitmap.getHeight(), CvType.CV_8UC1);
            Utils.bitmapToMat(bitmap, tmp);
            Utils.bitmapToMat(bitmap, tmp1);
            //convertir a escala de grises
            Imgproc.cvtColor(tmp, tmp, Imgproc.COLOR_BGR2HSV);

            Utils.matToBitmap(tmp1, bitmap);
            imageView.setImageBitmap(bitmap);
        }
```

**Figura 14**

*Escala de grises.*

*Nota:* En la parte superior se observa imagen sin procesar y en la parte inferior la imagen ya procesado en escala de grises con las librerías OpenCV y Android Studio.

Luego de procesar la imagen y visualizarla en escala de grises se le aplica el filtro Canny disponible también en la librería OpenCV con la función Imgproc.*Canny*(), en la figura 15 se observa el código escrito para realizar el proceso.

**Figura 15**

*Código de Android Studio Filtro Canny*

```
//convertir a escala de grises
Imgproc.cvtColor(tmp, tmp, Imgproc.COLOR_BGR2HSV);
// aplicar el filtro canny
Imgproc.Canny(tmp,tmp, threshold1: 80, threshold2: 150);
Utils.matToBitmap(tmp, bitmap);
imageView.setImageBitmap(bitmap);
```

**Figura 16**

*Filtro Canny*


*Nota:* En la figura se observa cómo se ve la imagen aplicando el filtro Canny.

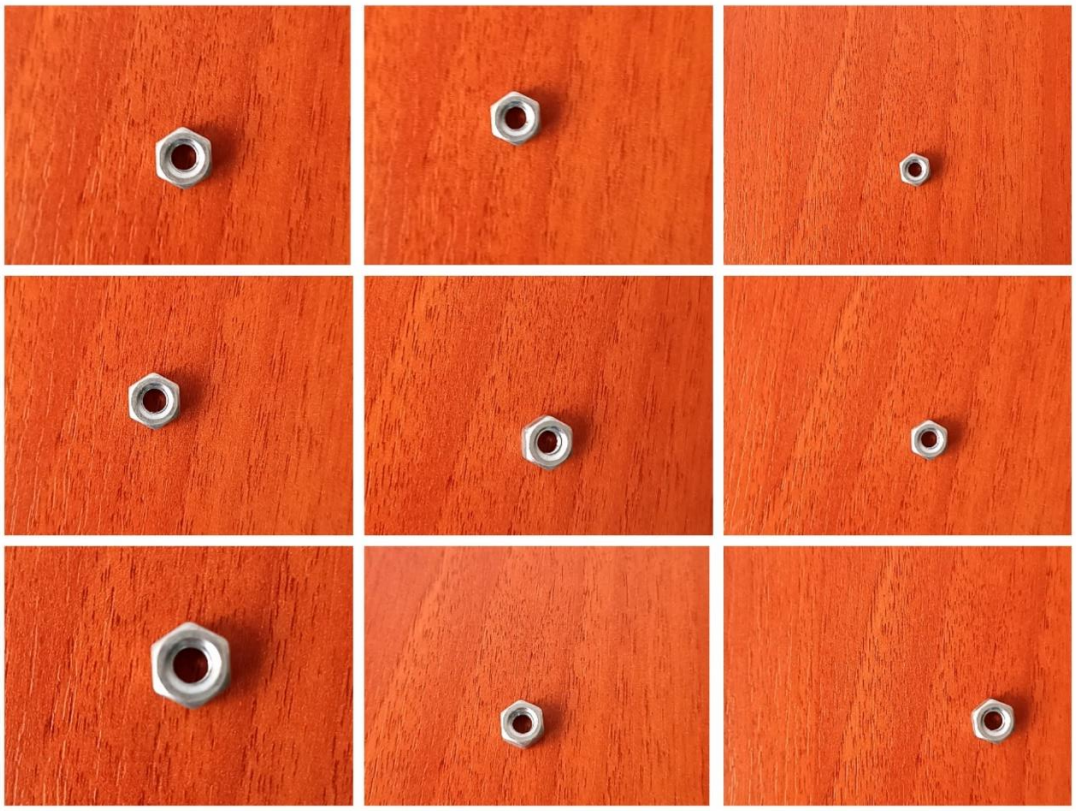
**Fase 4: Creación de la Base de Datos de Imágenes de Tuercas, Tonillos y Arandelas**

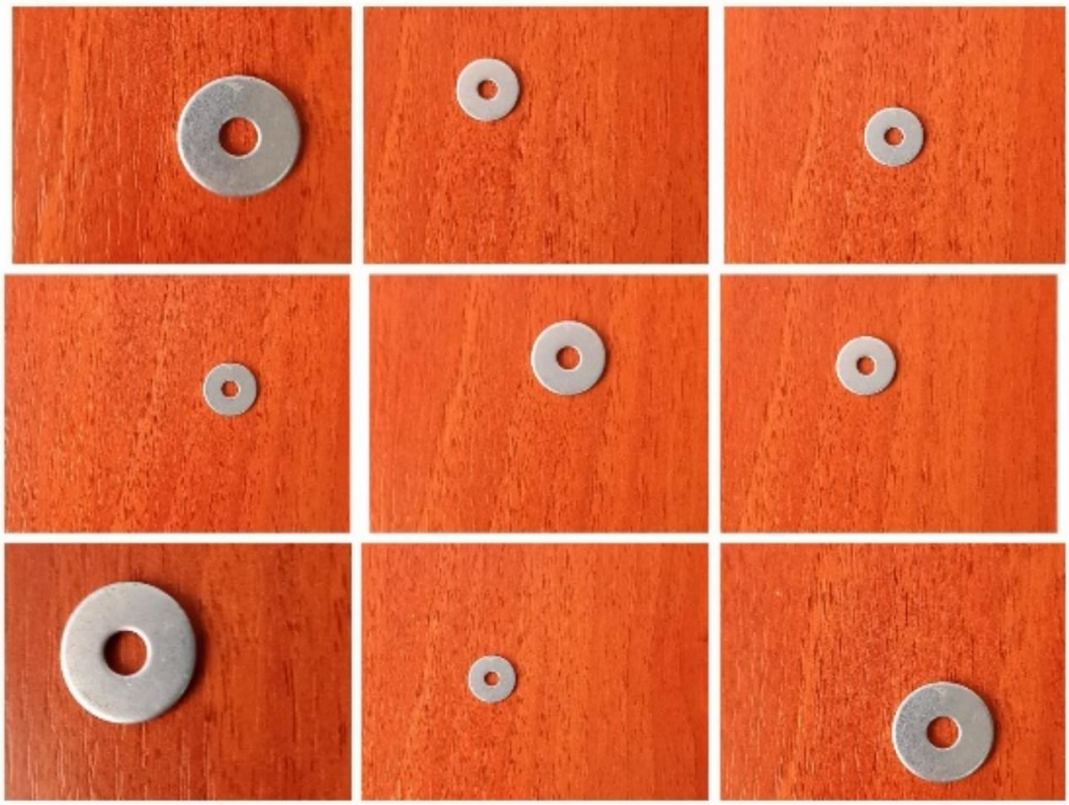
Luego de verificar la conexión de las librerías OpenCV con Android Studio y realizar algunos algoritmos sencillos para comprobar su correcto funcionamiento, se realiza una base de datos con 50 imágenes para cada uno de los 3 objetos, de los cuales 30 serán para hallar el valor de K que usara el algoritmo KNN y las otras 20 serán para validar su funcionamiento, estas imágenes fueron tomadas variando la posición, distancia y rotación de los objetos además se usó un fondo neutro para mejorar el funcionamiento del algoritmo.

En la figura 17 se observan algunas de las imágenes tomadas para la creación de la base de datos, de arriba hacia abajo primero están los tornillos luego la tuercas y por último están las arandelas.

**Figura 17**

*Base de datos de Tornillos, tuercas y arandelas.*





**Fase 5: Diseño del Clasificador**

Luego de tener la base de datos con la imágenes de las tuercas, tonillos y arandelas se inicia a diseñar el clasificador con los siguiente pasos

1. Encontrar los umbrales de HSV para tuercas, tonillos y arandelas.
2. Aplicar el filtro gaussiano.
3. Hallar contornos de las tuercas tornillos y arandelas.
4. Encontrar los momentos invariantes de Hu para cado uno de los tres objetos.
5. Hallar los centroides y el valor de K para cada objeto.
6. Ingresar el valor de los centroide y el valor de K de cada objeto al algoritmo KNN.
7. Realizar pruebas de validación del clasificador.
8. Diseñar una interfaz de usuario para la aplicación de clasificación.

***Umbrales HSV para tuercas tornillos y arandelas.***

Para encontrar los valores de estos umbrales HSV en los objetos se desarrolló una aplicación independiente de la principal con unas barras deslizadoras que permitieran cambiar los valores de cada umbral hasta encontrar los umbrales que detecten los objetos, además en la misma aplicación se insertó una barra para seleccionar el valor de la matriz a usar en el filtro Gaussiano, esta detección se realizó con video en línea y el código se muestra a en la figura 18.

**Figura 18**

*Código para hallar los umbrales HSV*

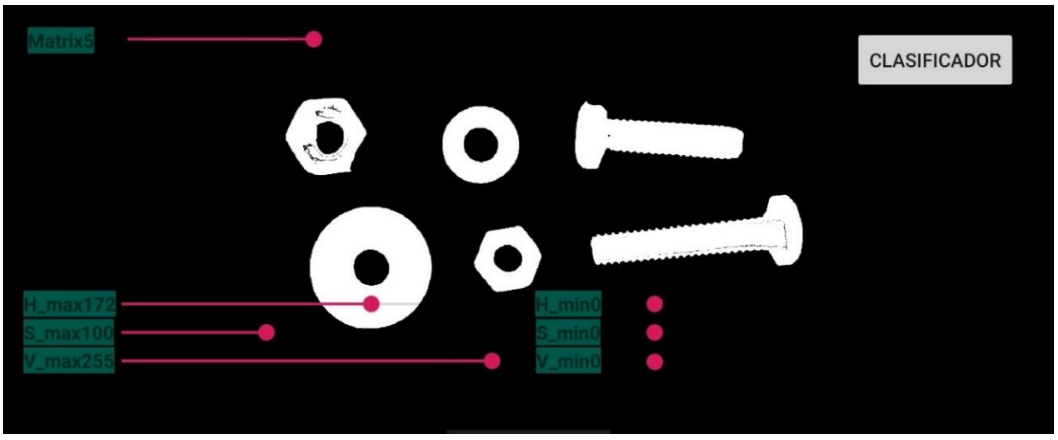
```
h_max=(SeekBar)findViewById(R.id.h_max);
s_max=(SeekBar)findViewById(R.id.s_max);
v_max=(SeekBar)findViewById(R.id.v_max);
h_min=(SeekBar)findViewById(R.id.h_min);
s_min=(SeekBar)findViewById(R.id.s_min);
v_min=(SeekBar)findViewById(R.id.v_min);
matriz=(SeekBar)findViewById(R.id.matriz);
```

```
h_max.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        th_max.setText("H_max"+h_max.getProgress());
    }
    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
    }
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    mRGBA = inputFrame.rgba();
    m=matriz.getProgress();
    n=2*m+1;
    Size matrix=new Size(n,n);
    hh_max=h_max.getProgress();
    ss_max=s_max.getProgress();
    vv_max=v_max.getProgress();
    hh_min=h_min.getProgress();
    ss_min=s_min.getProgress();
    vv_min=v_min.getProgress();
    Imgproc.GaussianBlur(mRGBA,mRGBA,matrix, sigmaX: 0);
    Imgproc.cvtColor(mRGBA,mHSV,Imgproc.COLOR_BGR2HSV);
    Core.inRange(mHSV,new Scalar(hh_min,ss_min,vv_min),new Scalar(hh_max,ss_max,vv_max),mHSV);
    return mHSV;
}
```

*Nota:* en la figura se observa parte del código usado para encontrar los umbrales HSV y el valor de la matriz nxn de los objetos.

**Figura 19**

*Aplicación para detección HSV.*




*Nota:* En la figura se observa la aplicación diseñada para hallar los umbrales HSV y la matriz nxn del filtro Gaussiano.

Los valores de los umbrales encontrados que se observan en la figura 19 serán los usados en la aplicación de clasificación de tuercas, tornillos y arandelas.

***Hallar contornos de las tuercas tornillos y arandelas***

Cuando ya se ha detectado la zona que pertenece a cada uno de los objetos se procede a encontrar el contorno de estos objetos, en este proyecto al ser los objetos el contorno más grande en la imagen usaremos el contorno exterior para decir que es el que pertenece a cado objeto hallado.

En la figura 20 se observa el código utilizado para detectar los contornos, acá usamos las funciones que pone OpenCV a nuestra disposición y en la figura 21 se observa como la aplicación encuentra y dibuja los contornos de cada objeto.

**Figura 20**

*Hallar Contorno con OpenCV*

```
InputStream inputStream = getContentResolver().openInputStream(imageuri);
bitmap = BitmapFactory.decodeStream(inputStream);
bitmap2 = bitmap.copy(bitmap.getConfig(), isMutable: true);
tmp = new Mat(bitmap.getWidth(), bitmap.getHeight(), CvType.CV_8UC4);
tmp1 = new Mat(bitmap.getWidth(), bitmap.getHeight(), CvType.CV_8UC4);
Utils.bitmapToMat(bitmap, tmp);
Utils.bitmapToMat(bitmap, tmp1);
//Mat mhsv = new Mat();
mhsv=Mat.zeros(tmp.size(), CvType.CV_8UC4);
//filtro gaus
Size matrix = new Size( width: 5,  height: 5);
Imgproc.GaussianBlur(tmp, tmp, matrix,  sigmaX: 0);
Imgproc.cvtColor(tmp, mhsv, Imgproc.COLOR_BGR2HSV);
Scalar lower = new Scalar(0, 0, 0);
Scalar upper = new Scalar(172,100, 255);
Core.inRange(mhsv, lower, upper, mhsv);
//Contornos
ArrayList<MatOfPoint> contours = new ArrayList<~>();
Mat hierarchy = new Mat();
Imgproc.findContours(mhsv, contours, hierarchy, Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_NONE);
//dibujar
for (int i = 0; i < contours.size(); i++) {
    Imgproc.drawContours(tmp1, contours, i, new Scalar(0,255,0),  thickness: 3);
}
```

**Figura 21**

*Contornos Encontrados*

***Momentos invariantes de Hu para tuercas, tornillos y arandelas.***

Los momentos invariantes de Hu son siete valores sin embargo en proyecto solo se usarán los primeros seis y no se tendrá en cuenta el séptimo momento que hace referencia a si el objeto está reflejado, este no se usará debido a que la forma del objeto seguirá siendo la misma.

*E*n la figura 22 se puede observar el código para calcular los momentos invariantes de Hu y en la figura 23 se observa la aplicación calculando los estos momentos en una tuerca, un tornillo y una arandela, los valores se multiplican por 1000 pues los son muy pequeños y dificultan algunos cálculos posteriores.

**Figura 22**

*Encontrar los Momentos Invariantes de Hu*

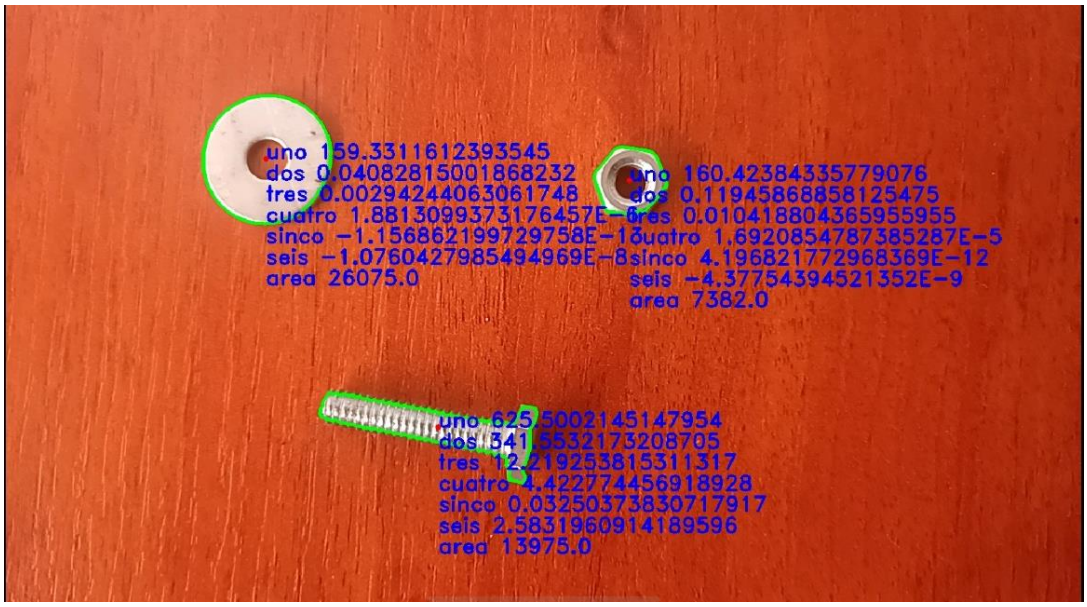
```
//Momentos
List<Moments> mo = new ArrayList<>(contours.size());

    for (int i=0;i<contours.size();i++){
        mo.add(Imgproc.moments(contours.get(i)));
```

```
//almacear momentos
double n20 = mo.get(i).nu20;
double n11 = mo.get(i).nu11;
double n02 = mo.get(i).nu02;
double n30 = mo.get(i).nu30;
double n21 = mo.get(i).nu21;
double n12 = mo.get(i).nu12;
double n03 = mo.get(i).nu03;
//Hallar Momentos Normalizados y multiplicarlos por 1000
//Primer Momento
uno = (n20 + n02) * multiplo;
//Segundo Momento
dos = (Math.pow((n20 - n02), 2) + 4 * Math.pow(n11, 2)) * multiplo;
//Tercer Momento
tres = (Math.pow(n30 - (3 * (n12)), 2)
       + Math.pow((3 * n21 - n03), 2)) * multiplo;
//Cuarto Momento
cuatro = (Math.pow((n30 + n12), 2) + Math.pow((n03 + n21), 2)) * multiplo;
//Quinto Momento
cinco = ((n30 - 3 * n12) * (n30 + n12)
       * (Math.pow((n30 + n12), 2) - 3 * Math.pow((n21 + n03), 2))
       + (3 * n21 - n03) * (n21 + n03)
       * (3 * Math.pow((n30 + n12), 2) - Math.pow((n21 + n03), 2))) * multiplo;
//Sexto Momento
seis = ((n20 - n02)
       * (Math.pow((n30 + n12), 2) - Math.pow((n21 + n03), 2))
       + 4 * n11 * (n30 + n12) * (n21 + n03)) * multiplo;
```

**Figura 23**

*Momentos invariantes de Hu vistos en cada objeto dentro de la aplicación.*



***Encontrar el valor de los centroides y los valores de K que usará el algoritmo KNN para clasificar las tuercas, tornillos y arandelas.***

Luego de realizar todos los procesos necesarios a las imágenes de la base de datos se procede a indicar los valores que necesita el algoritmo KNN en la aplicación para realizar una clasificación, anteriormente se halló el umbral HSV en el que se encuentran los objetos y el valor de los 6 momentos invariantes de Hu, esto para cada una de las 90 imágenes de la base de datos.

El valor del centroide para cada objeto en el algoritmo está determinado por un vector de 6 puntos, un punto por cada momento invariante de Hu, los valores de este único vector se hallaron realizando un promedio de los momentos obtenidos de cada una de las 30 imágenes tomadas a cada objeto (tuerca tornillo y arandela), es decir cada punto del vector es el promedio de 30 valores que se obtuvieron de las imágenes almacenadas en la base de datos.

En la tabla 1 se indican los valores hallados equivalentes al valor del centroide para cada uno de los objetos, estos valores esta multiplicados por 1000, pues por sí solos son demasiado pequeños y se dificulta el manejo:

**Tabla 1**

*Momentos Invariantes de Hu para cada Objeto*

| | | Objeto | | |
|---|---|---|---|---|
| | | Tuerca | Tornillo | Arandela |
| Momento invariante de Hu | 1 | 160.47328419286163 | 598.5887648 | 159.26675626003896 |
| | 2 | 0.1286606972008456 | 316.1574858 | 0.011942288322436715 |
| | 3 | 0.005132048704714567 | 13.42605958 | 9.854068628610245E-4 |
| | 4 | 2.4720022523478207E-5 | 6.165027711 | 1.3447804889561117E-6 |

| | | | | |
|---|---|---|---|---|
| | 5 | 2.1962688992282986E-11 | 0.115772624 | 1.2423613080356162E-13 |
| | 6 | 2.2877752904012297E-7 | 3.708490524 | 2.790782005602527E-9 |

Luego de encontrar el valor del centroide a utilizar en cada imagen se halla el valor de K que en este caso será la máxima distancia euclidiana presentada entre el centroide hallado y los momentos de las imágenes almacenadas, en la tabla 2 se observan los valores de K para cada objeto.

**Tabla 2**

*Valor de K para cada objeto*

| Objeto | K |
|---|---|
| Tuerca | 0.5972514743030677 |
| Tornillo | 2.41291716810571 |
| Arandela | 229.683665 |

En la tabla 3 se observa los umbrales HSV superior e inferior en el que se encuentran los objetos teniendo en cuanta el fondo neutro utilizado, estos valores fueron encontrados anteriormente junto con la matriz nxn del filtro gaussiano que tiene un valor de 5x5.

**Tabla 3**

*Umbrales HSV para los objetos*

| Umbral | H | S | V |
|---|---|---|---|
| Superior | 172 | 100 | 255 |
| Inferior | 0 | 0 | 0 |

Luego de tener los valores necesarios los ingresamos en el algoritmo KNN implementado en la aplicación móvil y esta debe decidir si un objeto de entrada pertenece o no pertenece a una de las 3 clases indicadas; tuerca, tornillo o arandela.

***Ingresar Datos en el Algoritmo KNN para la Clasificación de Tuercas, Tornillos y Arandelas.***

Teniendo los datos que usará en algoritmo se procese a implementar el algoritmo en Android Studio, lo primero es indicar las variables de entrada en el código las cuales se observan en la figura 24, luego de indicar las variables se inicia con el procesamiento de la imagen que se diseñó anteriormente por separado y ya en este punto se consolidan todos los procesos para poner en funcionamiento el algoritmo, este código se muestra en la figura 25.

La decisión que toma el algoritmo se realiza usando un condicional if que verifica si las características de los objetos de entrada se encuentran dentro de los parámetros indicados para así clasificarlos en una de las tres clases o no clasificarla en ninguna de estas, el código se muestra en la figura 26.

**Figura 24**

*Declaración de las variables*

```
JavaCameraView javaCameraView;
//Imagenes Mat
Mat mRGBA, mHSV, mGAUS;
//Momentos
double uno, dos, tres, cuatro, cinco, seis;
//multiplo momentos
int multiplo=1000,contador=0;
//centroide de los objetos
//ARANDELA
double[] centro_o_1 = {159.26675626003896, 0.011942288322436715, 9.8540686286102 45E-4,
                       1.3447804889561117E-6, 1.2423613080356162E-13, 2.790782005602527E-9};
//TUERCA
double[] centro_o_2 = {160.47328419286163, 0.1286606972008456, 0.005132048704714567,
                       2.4720022523478207E-5, 2.1962688992282986E-11, 2.2877752904012297E-7};
//TORNILLO
double[] centro_o_3={598.5887648,   316.1574858,    13.42605958,    6.165027711,    0.115772624,    3.708490524};
//valores de K
double max_2=2.41291716810571,max_1=0.5972514743030677,max_3=229.683664,dis_1,dis_2,dis_3;
boolean empezar=false;
```

**Figura 25**

*Código que Realiza el Procesamiento de la Imagen*

```
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    //Imagen entrada
    mRGBA = inputFrame.rgba();
    //filtro gaus
    Size matrix = new Size( width: 5,  height: 5);
    Imgproc.GaussianBlur(mRGBA, mGAUS, matrix,  sigmaX: 0);

    //espacio HSV
    Imgproc.cvtColor(mGAUS, mHSV, Imgproc.COLOR_BGR2HSV);
    //umbrales
    Scalar lower = new Scalar(0, 0, 0);
    Scalar upper = new Scalar(172,100, 255);
    Core.inRange(mHSV, lower, upper, mHSV);

    //hallar contornos
    ArrayList<MatOfPoint> contours = new ArrayList<~>();
    Mat hierarchy = new Mat();
    Imgproc.findContours(mHSV, contours, hierarchy, Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_NONE);

    //hallar momentos
    List<Moments> mo = new ArrayList<>(contours.size());
//dibujar contornos
for (int i = 0; i < contours.size(); i++) {
    //Area de cada contorno
    double area = Imgproc.contourArea(contours.get(i));

    //guardar momentos en mo
    mo.add(Imgproc.moments(contours.get(i)));

    //centroide del contorno
    //agregar 1e-5 para evitar la division por cero
    double mx = mo.get(i).m10 / (mo.get(i).m00 + 1e-5);
    double my = mo.get(i).m01 / (mo.get(i).m00 + 1e-5);

    //almacear momentos
    double n20 = mo.get(i).nu20;
    double n11 = mo.get(i).nu11;
    double n02 = mo.get(i).nu02;
    double n30 = mo.get(i).nu30;
    double n21 = mo.get(i).nu21;
    double n12 = mo.get(i).nu12;
    double n03 = mo.get(i).nu03;
```

```
//Hallar Momentos Normalizados y multiplicarlos por 1000
//Primer Momento
uno = (n20 + n02) * multiplo;
//Segundo Momento
dos = (Math.pow((n20 - n02), 2) + 4 * Math.pow(n11, 2)) * multiplo;
//Tercer Momento
tres = (Math.pow(n30 - (3 * (n12)), 2)
        + Math.pow((3 * n21 - n03), 2)) * multiplo;
//Cuarto Momento
cuatro = (Math.pow((n30 + n12), 2) + Math.pow((n03 + n21), 2)) * multiplo;
//Quinto Momento
cinco = ((n30 - 3 * n12) * (n30 + n12)
        * (Math.pow((n30 + n12), 2) - 3 * Math.pow((n21 + n03), 2))
        + (3 * n21 - n03) * (n21 + n03)
        * (3 * Math.pow((n30 + n12), 2) - Math.pow((n21 + n03), 2))) * multiplo;
//Sexto Momento
seis = ((n20 - n02)
        * (Math.pow((n30 + n12), 2) - Math.pow((n21 + n03), 2))
        + 4 * n11 * (n30 + n12) * (n21 + n03)) * multiplo;
//solo Dibujar contorno con area mayor que 2000
if (area > 3000 && area <100000) {
    //distancia euclidiana
    dis_1 = Math.sqrt(
            Math.pow(uno - centro_o_1[0], 2)
                    + Math.pow(dos - centro_o_1[1], 2)
                    + Math.pow(tres - centro_o_1[2], 2)
                    + Math.pow(cuatro - centro_o_1[3], 2)
                    + Math.pow(cinco - centro_o_1[4], 2)
                    + Math.pow(seis - centro_o_1[5], 2));
```

**Figura 26**

*Código para tomar la decisión en el clasificador*

```
    //Clasificar
    if (dis_1 < max_1) {

        Imgproc.rectangle(mRGBA, boundRect[i].tl(), boundRect[i].br(), verde,  thickness: 3);
        Imgproc.circle(mRGBA, new Point(mx, my),  radius: 4, amarillo,  thickness: -2);
        Imgproc.putText(mRGBA,  text: "ARANDELA", new Point(mx, my),  fontFace: 2,  fontScale: 1, amarillo,  thickness: 2);
    } else if (dis_2 < max_2) {

        Imgproc.rectangle(mRGBA, boundRect[i].tl(), boundRect[i].br(), verde,  thickness: 3);
        Imgproc.circle(mRGBA, new Point(mx, my),  radius: 4, azul,  thickness: -2);
        Imgproc.putText(mRGBA,  text: "TUERCA", new Point(mx, my),  fontFace: 2,  fontScale: 1, azul,  thickness: 2);
    } else if (dis_3 < max_3) {

        Imgproc.rectangle(mRGBA, boundRect[i].tl(), boundRect[i].br(), verde,  thickness: 3);
        Imgproc.circle(mRGBA, new Point(mx, my),  radius: 4, rojo,  thickness: -2);
        Imgproc.putText(mRGBA,  text: "TORNILLO", new Point(mx, my),  fontFace: 2,  fontScale: 1, rojo,  thickness: 2);
    } else {
        Imgproc.rectangle(mRGBA, boundRect[i].tl(), boundRect[i].br(), rojo,  thickness: 2);
        Imgproc.circle(mRGBA, new Point(mx, my),  radius: 4, rojo,  thickness: -2);
    }
}
```

**Fase 6: Resultados**

En esta fase se realizarán las pruebas para validar el funcionamiento del algoritmo en línea y con la ayuda de la matriz de confusión validaremos su exactitud, también se hará el diseño de la interfaz gráfica.

***Pruebas de validación del algoritmo de clasificación KNN.***

Luego implementar el algoritmo en Android Studio se compila la aplicación y se ejecuta en un dispositivo móvil para verificar su funcionamiento, la clasificación de cada uno de los tres objetos por separado y juntos se muestra en las figura 27.

Para facilitar la visualización de la clasificación se creó un código de colores para cada objeto:

- Los objetos clasificados en una de las tres clases son encerrados en un rectángulo de color verde y aquellos que no perteneces a una de estas clases son encerrados en un rectángulo de color rojo.
- Cuando el algoritmo encuentra un objeto y lo clasifica en una de las tres clases le da un nombre, un número y un color según la clase a la cual pertenece.

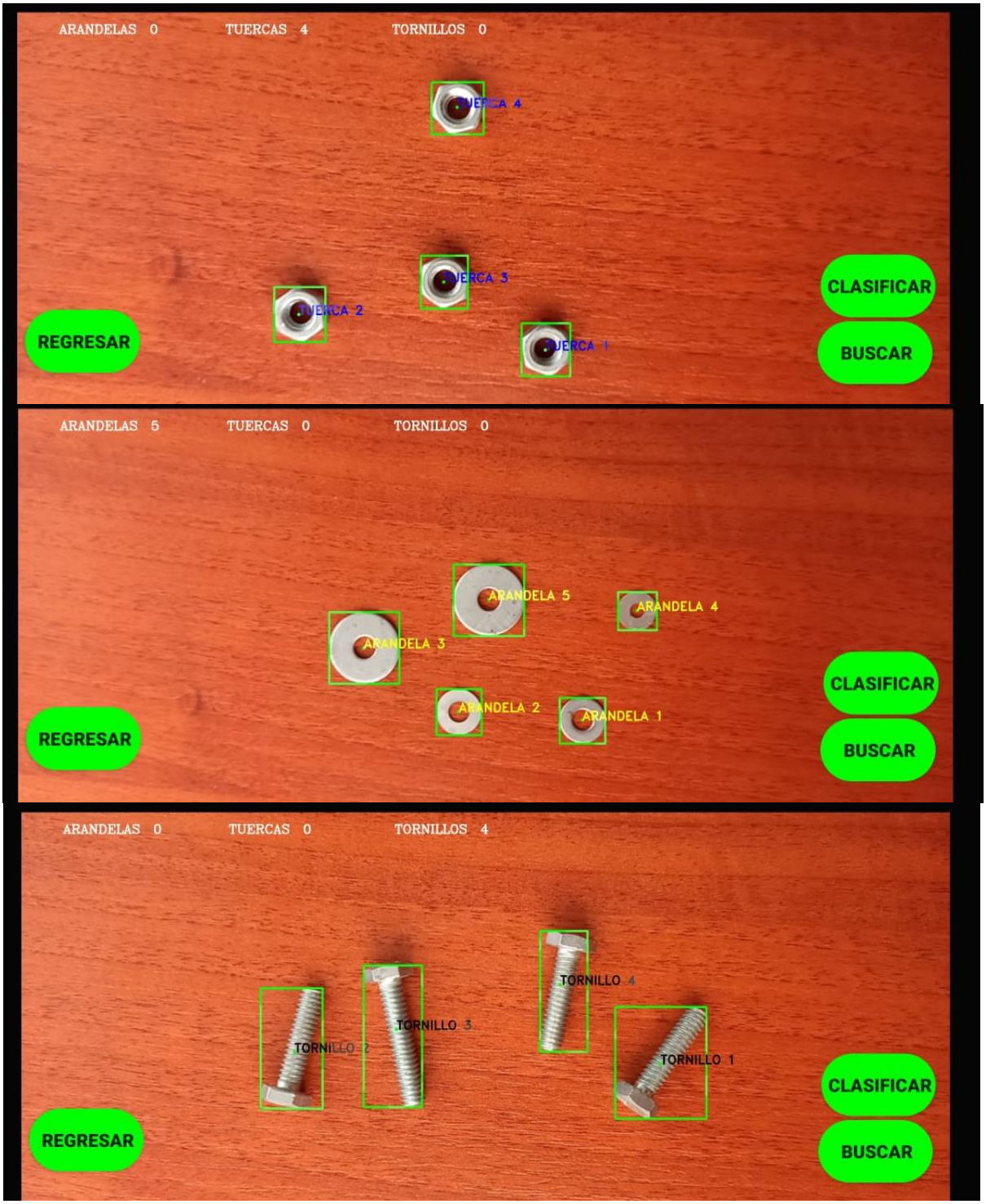
  Tuerca color azul

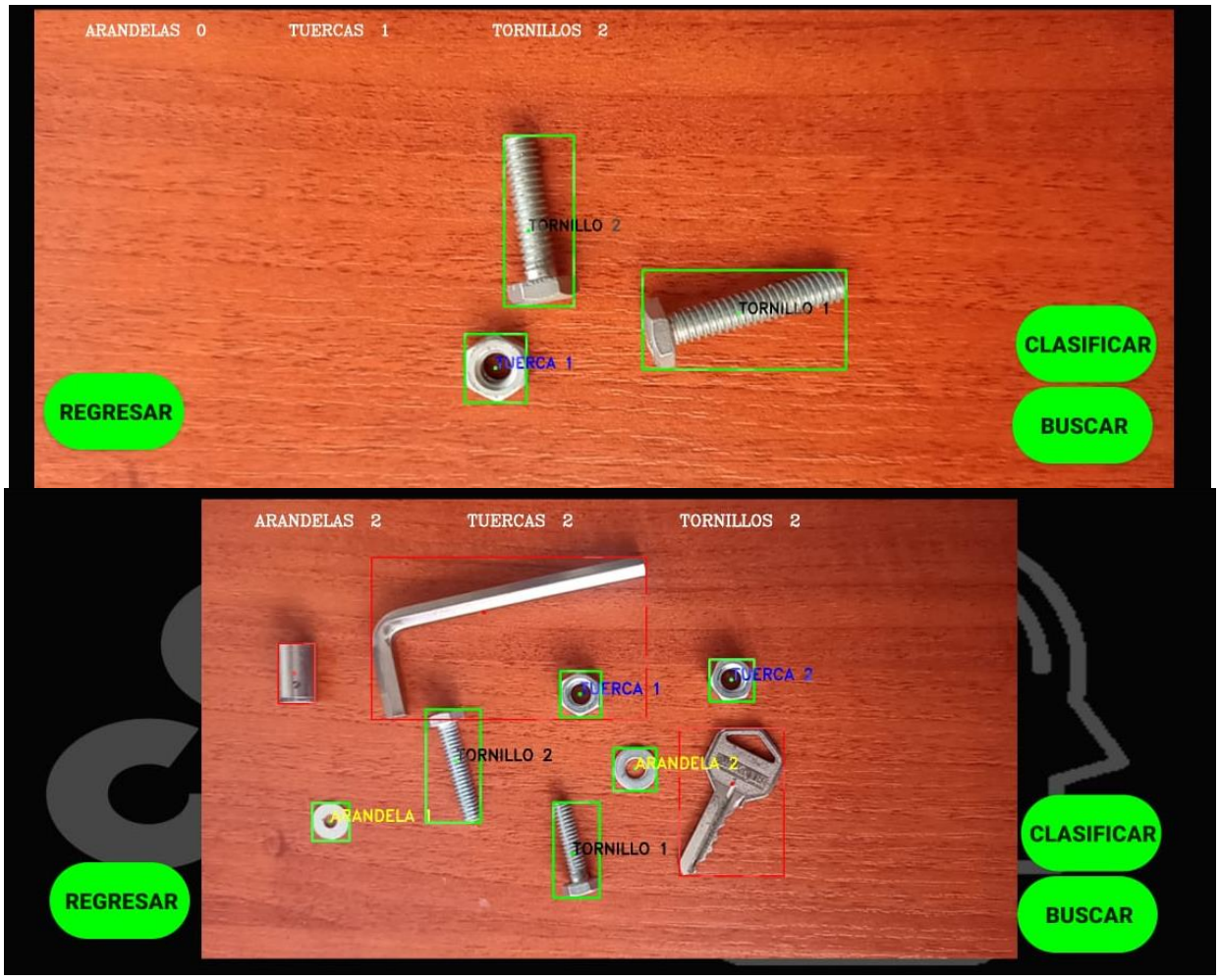
  Arandela color amarillo

  Tornillo color negro
- En la parte superior de la imagen se puede visualizar el conteo de los objetos según su clase, esto está escrito con color blanco y fijo en la pantalla.

**Figura 27**

*Pruebas de clasificación*






*Nota*: En la figura se observa el funcionamiento de la aplicación.

***Matriz de confusión***

Esta matriz es usada para evaluar el desempeño de un algoritmo de aprendizaje supervisado en machine learning.

Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa las instancias de la clase real, lo que no permite ver los aciertos y errores que tiene nuestro algoritmo. En la figura 28 se observa la matriz de confusión donde:

- VP: Es el número de predicciones positivas que el algoritmo clasifica como positivas. Verdaderos positivos.
- FP: Es el número de predicciones incorrectas que el algoritmo clasifico como negativos, es decir clasifico como negativo, pero en realidad eran positivos. Falsos positivos.
- FN: el número de predicciones incorrectas que el algoritmo clasifico como positivas, es decir clasifico como positivo, pero en realidad eran negativos. Falsos negativos.
- VN: Es el número de predicciones negativas que el algoritmo clasifica como negativas. Verdaderos negativos.

**Figura 28**

*Matriz de Confusión*

| | CLASIFICADOR | | |
|---|---|---|---|
| VALORES REALES | | POSITIVOS | NEGATIVOS |
| | POSITIVOS | VP | FP |
| | NEGATIVOS | FN | VN |

- Exactitud: Cantidad de predicciones positivas que fueron correctas, se calcula usando la proporción entre los positivos reales predichos por el algoritmo y todos los casos positivos. Tal como se muestra en la ecuación (5).

$$Exactitud = \frac{VP+VN}{VP+FP+FN+VN} \quad (5)$$

- Precisión: Dispersión del conjunto de valores obtenidos a partir de mediciones repetidas de una magnitud, se calcula usando la proporción entre el número de predicciones correctas y el total de predicciones. Tal como se muestra en la ecuación (6).

$$Precisión = \frac{VP}{VP+FP} \quad (6)$$

- Especificidad: Eficiencia en la clasificación de todos los elementos que no son de su misma clase. Tal como se muestra en la ecuación (7).

$$Especificidad = \frac{VN}{VN+FP} \quad (7)$$

- Sensibilidad: Eficiencia en la clasificación de todos los elementos que son de la misma clase. Tal como se muestra en la ecuación (8).

$$Sensibilidad = \frac{VP}{VP+FN} \quad (8)$$

- F1 Score: Esta medida tiene en cuenta la precisión y la sensibilidad. Tal como se muestra en la ecuación (9).

$$F1\ Score = 2 * \frac{Precisión * Sensibilidad}{Precisión + Sensibilidad} \quad (9)$$

Teniendo en cuenta la informacion anterior se realizó la matriz de confusión para el clasificador KNN diseñado, donde se toma el valor de la exactitud ya que nuestra prioridad son los verdaderos positivos que se tienen en el algoritmo, adicional a los tres objetos tuercas,

tornillos y arandelas se creó un cuarto objeto llamado “otro” haciendo referencia a que el algoritmo no clasifica un objeto en una de las tres clases principales.

**Tabla 4**

*Resultado de la Exactitud en la Matriz de Confusión*

| | | Clasificador | | | |
|---|---|---|---|---|---|
| | | Tuerca | Tornillo | Arandela | Otro |
| Valores reales | Tuerca | 18 | 0 | 2 | 0 |
| | Tornillo | 0 | 16 | 0 | 4 |
| | Arandela | 1 | 0 | 19 | 0 |
| | Otro | 0 | 0 | 0 | 20 |
| | Exactitud | 96,250% | 95,000% | 96,250% | 95,000% |

***Diseño de una Interfaz de Usuario para la Aplicación de Clasificación de Tuercas, Tonillos y Arandelas.***

Luego de implementar el algoritmo y verificar su funcionamiento se da un nombre a la aplicación (DetectApp) y diseña una interfaz gráfica para mejorar la experiencia de usuario al usar la aplicación.

Las aplicaciones Android están divididas en activities que son el componente principal de la interfaz gráfica. Casi todas las actividades interactúan con el usuario, por lo que la clase Activity se encarga de crear una ventana en la que puede colocar la interfaz de usuario.

En la interfaz diseñada se crean cuatro activities a las cuales se les cambia el fondo y se les agrega botones con diferentes opciones, en la Activity principal se diseña un menú de inicio donde se puede seleccionar entre las tres activities secundarias creadas:

Video: Permite abrir la cámara y realizar una clasificación línea de los objetos que este visualizando la cámara.

Galería: Permite cargar una imagen desde la galería del teléfono celular para procesarla y realizar la clasificación, dentro de esta Activity se tiene la opción de buscar que permite ingresar a la galería del teléfono para buscar la imagen a procesar tal cual se ve en la figura 30.

Foto: Permite tomar una fotografía, almacenarla en el dispositivo y realizar la clasificación, dentro de esta Activity se tiene la opción de FOTO para captar la imagen y procesarla, como se observa en la figura 31.

Además, dentro de cada una de las tres Activities secundarias se tienen dos opciones: REGRESAR que nos permite regresar al menú principal y CLASIFICAR que inicia el procesamiento de la imagen para hacer la clasificación.

**Figura 29**

*Menú de la aplicación.*


**Figura 30**

*Opción galería*

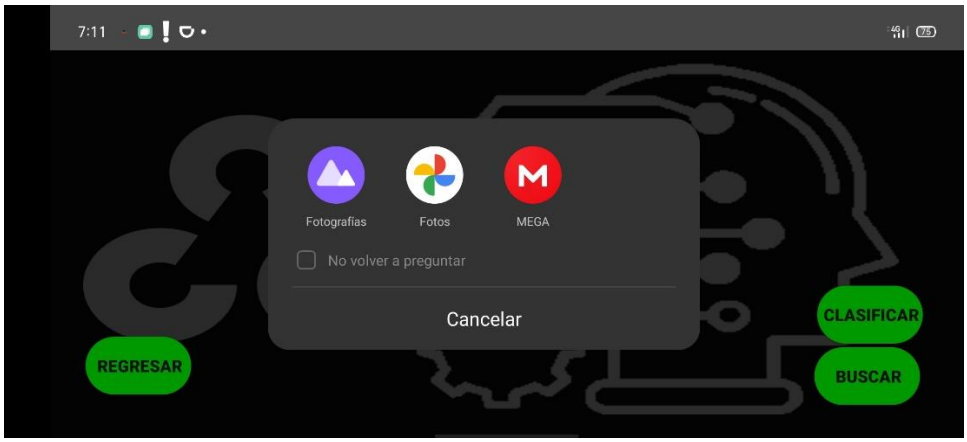



**Figura 31**

*Opción foto*

Adicional a la interfaz de usuario se modificó el icono para que sea más llamativo, este se diseña con el logo de las librerías OpenCV junto con un fondo negro como se visualiza en la figura 28.

**Figura 32**

*Icono de la aplicación*


Para facilitar la visualización total de la aplicación se realizó un video explicativo el cual se subió a la plataforma YouTube, donde se muestra la interfaz gráfica y el funcionamiento total de la aplicación, https://youtu.be/7KUbl5_wMc0 .

# 7. Conclusiones y Recomendaciones

- Al realizar una breve revisión bibliográfica se encontró informacion suficiente para implementar el algoritmo KNN en el software Android Studio enlazado en la librerías OpenCV, logrando su implementación y correcto funcionamiento.
- El algoritmo diseñado da las bases para elaborar algoritmos de clasificación más robustos y que puedan ser usados en el desarrollo de futuros proyectos de investigación e incluso implementar dicho algoritmo en la industria, como por ejemplo clasificación del grado de madures de un producto, clasificación de defectos en productos, entre otros.
- El desarrollo de este proyecto permite ver los procesos que se deben realizar en el diseño de una algoritmo que use visión artificial y machine learning, a pesar de que son campos muy extensos se pueden diseñar proyectos pequeños y aplicados a la solución de problemas concretos.
- La base de datos para este proyecto está conformada por 150 imágenes, 90 para diseñar el algoritmo y 60 para la validación, a través de las cuales se logró captar toda la informacion necesaria para diseñar el algoritmo KNN capaz de clasificar, tuercas, tornillos y arandelas. Sin embargo, si se desean abarcar otros ambientes y otro tipo de objetos es posible que la base de datos se deba aumentar considerablemente para así no perder calidad en el algoritmo.
- La aplicación móvil desarrollada para ejecutar el algoritmo es bastante fiable y eficiente, sin embargo, es trabajos futuros puede ser posible mejorar el algoritmo para que tenga un mejor funcionamiento en ambientes con variaciones de iluminación más marcados, y también adicionar más objetos a clasificar**.**

- La segmentación de imágenes es sin duda una parte muy importante en cualquier proceso que incluya visión artificial, de una buena segmentación depende un buen funcionamiento del algoritmo, en este proyecto en ambientes que presenten variaciones de iluminación la segmentación no se realiza correctamente por lo cual el algoritmo no realiza la clasificación de los objetos como se espera, sin embargo esto puede mejorarse cambiando el celular a uno que tenga una mejor cámara para que responda mejor ante estas variaciones y así capte los detalles en las imágenes tomadas

## Bibliografía

(15 de 12 de 2020). Obtenido de Ecured: https://www.ecured.cu/Matrices_de_confusi%C3%B3n

Alambiaga, J. B. (2015). Desarrollo de una aplicación móvil para la detección y clasificación de hojas de árboles. Gandia, España.

Cardenas, A. A. (Diciembre de 2019). comparativa de extractores de características para clasificación de rostros. Quito, Ecuador.

Cazorla, A., Olmo, F. J., & Alados-Arboledas, L. (2005). Estimación de la cubierta nubosa en imágenes de cielo mediante el. *XI Congreso Nacional de Teledetección*, (págs. 335-336). puerto de la cruz.

Comunidad OpenCV. (1 de MAYO de 2020). *OpenCV*. Obtenido de https://opencv.org/

Dueñas, C. p. (2009). Introducción a la Visión Artificial. *Apuntes de Visión Artificial*, Madrid, España.

Estrebou, C. (Abril de 2021). Trabajo Final presentado para obtener el grado de Especialista en Computación Gráfica, Imágenes y Visión por Computadora. La plata, Argentina.

Flores, W. G. (3 de Junio de 2015). Reconocimiento de objetos en fotografías.

Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing.* New Jersey: Pearson Education International.

Google. (1 de MAYO de 2020). *Android developer*. Obtenido de https://developer.android.com/

Hai, T. S., Hoang Thai, L., & Thanh Thuy, N. (Febrero de 2015). Facial Expression Classification UsingArtificial Neural Network and K-Nearest Neighbor. Ho Chi Minh, Vietnan.

Howse, J. (2015). *Android Application.* Birmingham: Packt Publishing Ltd.

Kapur, S., & Thakkar, N. (2015). *Mastering OpenCV Android Application Programming.* Birmingham: Packt Publishing Ltd.

Kuang, Q., & Zhao, L. (diciembre de 2009). A Practical GPU Based KNN Algorithm. *A Practical GPU Based KNN Algorithm*. Huangshan, China.

Ledesma, V. C., Montero, R. S., & Ornelas Rodríguez, M. (s.f.). Clasificación de sonidos mediante el análisis de imágenes bidimensionales del espectro de frecuencias.

Liu, B. (2010). *Web Data Mining.* Chicago: Springer.

Miranda, J. C. (marzo de 2018). clasificación automática de naranjas por tamaño y por defectos utilizando técnicas de visión por computadora. San lorenso, paraguay.

Muhammad, A. (2015). *OpenCV Android Programming.* Birmingham: Packt Publishing Ltd.

Quispe-Ayala, M. R., Asalde-Alvarez , K., & Roman-Gonzalez, A. (s.f.). Clasificación de Imágenes Usando Técnicas de Compresión de Datos. Peru.

Salgueiro, J. D., De la piedad Gascueña, A., & Sanchez Muñoz, E. (2013). Procesado de imágenes para. *Procesado de imágenes para*. Madrid, España.

Santillán, E. G. (Mayo de 2008). Detección y clasificación de objetos dentro de un salón de clases empleando técnicas de procesamiento digital de imágenes. Mexico, Mexico.

Vázquez, S. R., Martínez, A. V., & Lozano Ginori, J. V. (2016). Clasificación de células cervicales mediante el algoritmo KNN usando rasgos del núcleo. *Revista Cubana de Ciencias Informáticas*.