Facultad # 4

Título: Visión por computadora para la detección de anomalías en una placa Arduino

---

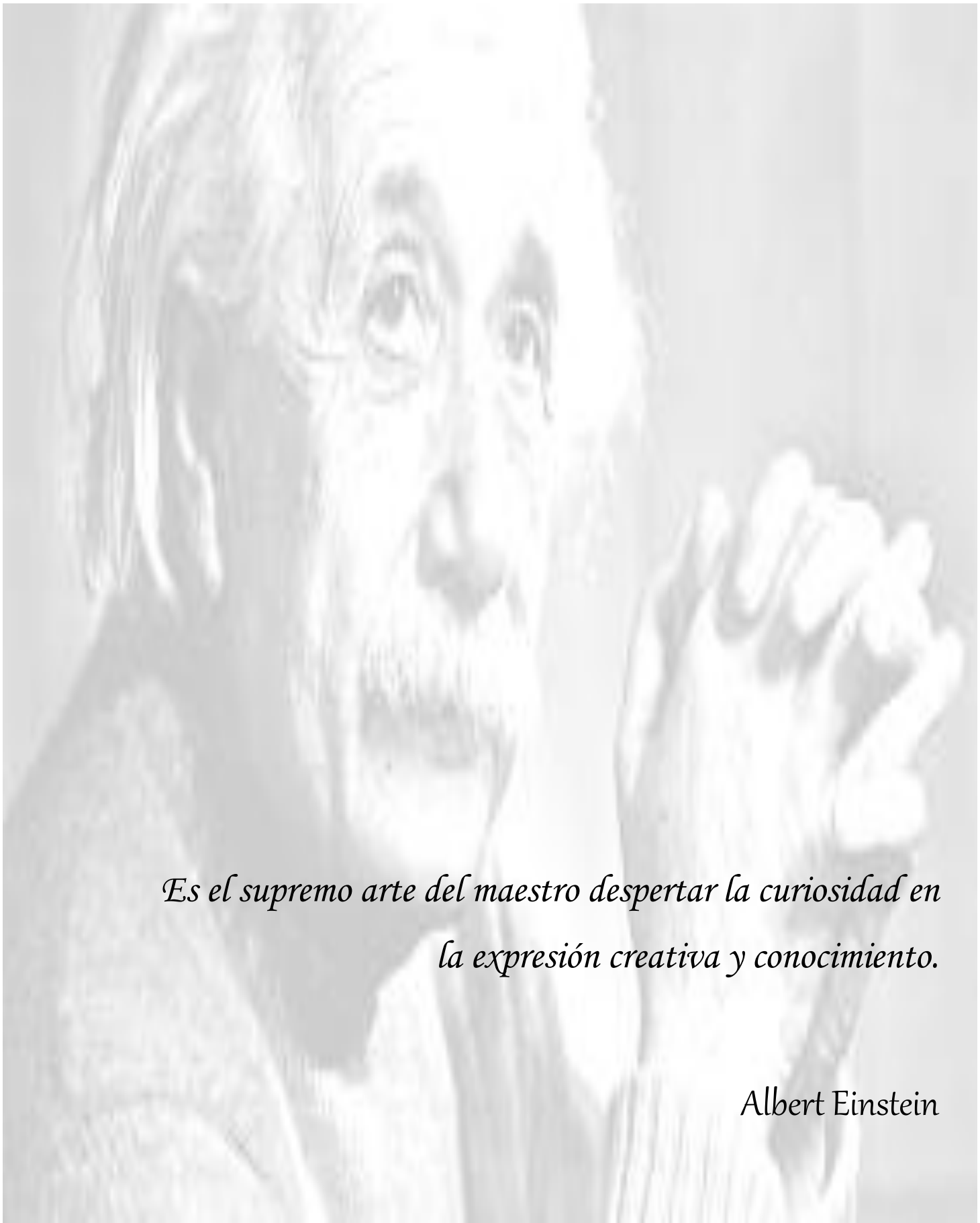
Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Raudel Hernández González

Tutores: Ing. Manuel Fornés Martínez

Ing. Patricia Ponce de León Atiés

La Habana, 2020



Es el supremo arte del maestro despertar la curiosidad en la expresión creativa y conocimiento.

Albert Einstein

Declaración de autoría

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____

Firma del autor
Raudel Hernández González

_____

Firma del tutor
Ing. Manuel Fornés Martínez

_____

Firma del tutor
Ing. Patricia Ponce de León Áties

Agradecimientos

Quisiera agradecerle en primer lugar, a mis padres por ser las personas más especiales de mi vida, por su comprensión, esmero, por darme tanto amor y por apoyarme en todas mis decisiones.

También le agradezco a mi hermanita por darme tanto amor, por sus chistes y por estar siempre a mi lado.

A mi abuela, una mujer increíble, la razón por la que creo que siempre se puede y que nada es imposible, por estar siempre en los momentos más difíciles y por ser un ejemplo de persona con grandes valores humanos.

A todos mis compañeros del apartamento, de ellos me quedo con los mejores momentos, las mejores bromas y jodederas: Angel, Leo, Luis, Yasmany, Yan y Yoandy.

A mi compañero de cuarto Angel que lo considero como mi hermano y mejor amigo, con el que jodía todos los días y que siempre estuvo en momentos difíciles de mi carrera.

A Yannier por tener tanta paciencia y ayudarme con mis problemas en la programación.

Al profesor Reinier por su enorme ayuda y dedicación de su tiempo cuando más me hizo falta.

A mis tutores, gracias por sus comentarios y por toda la ayuda que me brindaron.

A la UCI por haberme hecho crecer estos 5 años y formarme como profesional.

A todos los profesores de la facultad que de una forma u otra influyeron en mi preparación profesional.

De una forma u otra a todas aquellas personas que a lo largo de mi carrera como estudiante aportaron un granito de arena para que estuviese aquí hoy.

Gracias a todos.

Raudel

Dedicatoria

Quiero dedicar esta tesis a todas las personas que han estado conmigo en las buenas y en las malas, en especial a mis padres que se merecen todo lo que les pueda dar y mucho más.

Raudel

Resumen

La visión artificial representa una de las herramientas transversales más relevantes dentro de la industria, permitiendo identificar y clasificar imágenes a través de dispositivos electrónicos con el uso de una cámara digital. En Cuba existen un grupo de instituciones encargadas de informatizar los diferentes procesos que se realizan en las empresas del país. Una de esas instituciones es la Universidad de las Ciencias informáticas, en la cual se encuentra el Centro de Informática Industrial, especializado en el desarrollo de sistemas para la industria. En dicho centro se realizan varias aplicaciones, aunque aún no se ha desarrollado alguna capaz de detectar anomalías en una placa Arduino. Por esta razón la presente investigación tiene el objetivo de desarrollar una aplicación que permita la detección de anomalías en una placa Arduino con el uso de visión por computadoras o como también llamada visión artificial. Para lograr este objetivo, se utilizó como metodología de desarrollo el Proceso Unificado Ágil, versión establecida por la Universidad de las Ciencias Informáticas, además se empleó el entorno de desarrollo integrado oficial para la plataforma Android, Android Studio, permitiendo de esta forma el desarrollo de la presente investigación. Además, dos lenguajes para la implementación, Kotlin y Python, este último, el más indicado para el trabajo con la visión artificial. Los resultados fueron validados a través de pruebas de software, donde se pudo comprobar la calidad de las funcionalidades internas y externas de la solución propuesta.

Palabras claves: Android, anomalías, aplicación, detección, placa Arduino, redes neuronales, visión artificial, visión por computadora.

Índice general

Introducción....11
CAPÍTULO 1. Fundamentación teórica....14
1.1. Conceptos asociados con el objeto de estudio de la investigación....14
1.2. Visión artificial y redes neuronales para la detección de anomalías....15
1.2.1. Red Neuronal Convolucional....19
1.3. Sistemas similares....23
1.3.1. Aplicación Android que detecta objetos para una imagen tomada de la cámara....23
1.3.2. Aplicación Android que detecta objetos en tiempo real utilizando la cámara del dispositivo....24
1.3.3. Aplicación Android “Visión General”....25
1.4. Metodología de desarrollo....26
1.4.1. Metodología AUP-UCI....26
1.5. Herramientas y tecnologías utilizadas....28
1.5.1. Herramienta de modelo....28
1.5.2. Lenguajes de programación....29
1.5.3. Bibliotecas para el desarrollo de la visión por computador....31
1.5.4. Entorno de desarrollo integrado....32
1.6. Conclusiones del capítulo....33
CAPÍTULO 2. Solución propuesta....34
2.1. Descripción de la solución propuesta....34
2.2. Captura de requisitos....34
2.2.1. Requisitos funcionales....34
2.2.2. Requisitos no funcionales....38
2.3. Historias de usuario....38
2.4. Diseño de la solución propuesta....41
2.4.1. Arquitectura de software....41
2.4.2. Diagrama de clase del diseño....43
2.4.3. Patrones de diseño....44
2.4.4. Diagrama de paquetes....45
2.5. Conclusiones del capítulo....46
CAPÍTULO 3. Implementación y pruebas....47
3.1. Modelo de implementación....47
3.1.1. Estándares de codificación....47
3.1.2. Diagrama de componente....48
3.1.3. Diagrama de despliegue....49

3.2. Pruebas del software 50
3.2.1. Estrategia de prueba.....50
3.3. Conclusiones del capítulo 58
Conclusiones generales 59
Recomendaciones.....60
Referencias bibliográficas 61

Índice de figuras

Figura 1: Ejemplo de detección de anomalías en imágenes. 18
Figura 2: Aproximación de deep learning en la clasificación de imágenes. 19
Figura 3: CNN entrenada para reconocer clases como barco, bicicleta y coche [41]. 21
Figura 4: Funcionamiento de la aplicación AndroidTensorFlowLearning 24
Figura 5: Funcionamiento de la aplicación desarrollada por Prabhakar Thota 25
Figura 6: Funcionamiento de la aplicación Android "Visión General" 26
Figura 7: Fases de la Metodología AUP-UCI [28] 27
Figura 8: Estructura de la Arquitectura Clean 42
Figura 9: Diagrama clase de diseño. Elaboración propia. 44
Figura 10: Diagrama de paquetes. Elaboración propia. 46
Figura 11: Diagrama de componentes. Elaboración propia. 49
Figura 12: Diagrama de despliegue. Elaboración propia. 50
Figura 13: Grafo de flujo para el método recognizeImage() 53
Figura 14: Resultados de las pruebas de caja negra 57

Índice de tablas

Tabla 1: Descripción de los requisitos identificados 35
Tabla 2: HU_ Establecer las capturas de pantalla en la interfaz gráfica de la aplicación 39
Tabla 3: HU_ Identificar imagen 40
Tabla 4: HU_ Clasificar imagen 40
Tabla 5: Caso de prueba. Ruta independiente #1 54
Tabla 6: Caso de prueba. Ruta independiente #2 54
Tabla 7: Escenarios del Diseño de Caso de Prueba 55
Tabla 8: Descripción de la variable del Diseño de caso de prueba 56
Tabla 9: Resumen de las pruebas de caja negra 57

Introducción

La visión artificial representa una de las herramientas transversales más relevantes dentro de la industria, ya que está claramente integrada en cada uno de los apartados de un proceso productivo. La visión artificial son el conjunto de herramientas y métodos que permiten obtener, procesar y analizar imágenes del mundo real con la finalidad de ser tratadas por un ordenador. Esto permite automatizar una amplia gama de tareas al aportar a las máquinas la información que necesitan para la toma de decisiones correctas en cada una de las tareas que les han sido asignadas, y en muchos casos resuelven problemas que una persona no podría asumir [1]. Además, esta herramienta es capaz de identificar y clasificar imágenes a través de dispositivos electrónicos con el uso de una cámara digital.

Un buen ejemplo, es el de los sensores de imagen que forman una cámara. Cuanto mayor es el número de elementos fotosensibles, también llamados pixeles, mayor es la resolución de la imagen que puede proporcionar la cámara. Como consecuencia de esto, el usuario que visualiza una imagen o una sucesión de imágenes, procedentes de una cámara, podrá obtener información más detallada, o con mayor precisión. Es posible detectar un crecimiento de la célula en un organismo o medir la progresiva deforestación de un bosque [1].

La aplicación de la visión artificial en la industria permite que los procesos de fabricación sean automatizados, lo que se traduce en mejores resultados de producción mediante la aplicación de controles de calidad y mayor agilidad en cada una de las fases. Además, tiene especial importancia por el enorme rendimiento que aporta en la automatización de tareas repetitivas y de alta precisión. Los entornos abiertos siempre constituyen retos para la visión artificial por las variaciones lumínicas incontroladas, por lo que las condiciones cerradas de un entorno industrial favorecen en mayor medida la consecución de los objetivos de la visión artificial en dicho entorno [1].

Los robots inteligentes y los sistemas de visión artificial son de los agentes que hacen posible que la tecnología más disruptiva se aplique en diferentes procesos de producción y que se consigan resultados jamás pensados, debido al uso óptimo de la información. Si hay algo que ha ido cobrando mayor relevancia con la transformación digital es el poder de la información y la importancia de contar con sistemas capaces de tomar, medir y analizar datos, y en esto, los sistemas de visión artificial son los verdaderos expertos. Las cámaras de visión artificial, junto a los softwares de análisis de imagen, permiten que cualquier proceso de la línea de montaje se pueda automatizar y que las máquinas actúen de acuerdo a la información que procesan y contrastan con los parámetros con las que han sido programados [1].

Actualmente, el uso de la visión artificial industrial ha permitido mejorar los procesos de producción de manera inimaginable hasta hace unas décadas. Esto ha permitido obtener productos de mayor calidad a costes más bajos y en prácticamente todos los campos de la industria, desde la automoción y la alimentación, a la electrónica y la logística. El control de calidad es el conjunto de técnicas y herramientas que permiten detectar errores en el proceso productivo, así como tomar las medidas adecuadas para suprimirlos. Esto proporciona un control mucho más exhaustivo sobre el producto final, lo que garantiza que, cuando llega al consumidor, sean productos que cumplan con unos estándares de calidad concretos y determinados. De este modo, los productos que no cumplen con los requisitos de calidad mínimos son eliminados del proceso, permitiendo solventar los posibles fallos que tengan lugar durante el proceso productivo. Esto se consigue mediante la realización de inspecciones y pruebas de muestreo de forma continuada [2].

En general, la automatización industrial con la intervención de la visión artificial da más autonomía a las máquinas**,** ya que estas actúan en concordancia con datos reales, exactos y sin márgenes de error o valoraciones subjetivas [1].

En Cuba existen un grupo de instituciones encargadas de informatizar los diferentes procesos que se realizan en las empresas del país. Una de esas instituciones es la Universidad de las Ciencias informáticas (UCI), en la cual se encuentra el Centro de Informática Industrial (CEDIN) especializado en el desarrollo de sistemas para la industria. En este centro se realizan varias aplicaciones, aunque aún no se ha desarrollado alguna capaz de detectar anomalías en una placa Arduino, ya se encuentra dicha actividad entre los objetivos estratégicos de dicho centro, con el fin de cumplir con estos requisitos y así con el proceso de informatización.

Tomando en consideración la problemática planteada anteriormente se define como **problema de la investigación:** ¿Cómo lograr la detección de anomalías en una placa Arduino con el uso de visión por computadoras?

Siendo el **objeto de estudio**: La informatización de la detección de las anomalías en una placa Arduino con el uso de visión por computadoras, enmarcado en el **campo de acción**: Aplicación informática en el uso de visión por computadoras para la detección de anomalías en una placa Arduino.

Como **objetivo general** se propone: Desarrollar una aplicación que permita la detección de anomalías en una placa Arduino con el uso de visión por computadoras.

Para alcanzar el objetivo general propuesto se definen los siguientes **objetivos específicos:**

- Fundamentar la investigación a partir de los principales conceptos y basamentos de la detección de anomalías a través de la visión artificial.
- Selección de la biblioteca a utilizar para realizar la red neuronal.
- Definir funcionalidades para el proceso de la detección de anomalías en la placa Arduino.
- Implementar las funcionalidades definidas.
- Realizar pruebas para garantizar la funcionalidad del sistema.

Para la realización de la investigación se adopta como **idea a defender** que, el desarrollo de una aplicación informática para la detección de anomalías en una placa Arduino con el uso de la visión artificial, permitirá identificar a través de fotos que captura la cámara del teléfono móvil a la placa Arduino posibles anomalías que pueda presentar esta placa.

Para todo el proceso de investigación y elaboración de este trabajo se tomó en cuenta la utilización de varios métodos científicos de investigación como:

Métodos teóricos:

- **Análisis y síntesis**: se empleó para el análisis de los resultados y para la valoración en el cumplimiento de los requisitos de diseño y los objetivos trazados en la investigación, de forma tal que pueda arribarse a conclusiones satisfactorias que lleven a posibles variantes de solución.
- **Modelación:** se utilizó para la realización de los diagramas necesarios en el proceso de desarrollo de software, haciendo una representación abstracta de la solución, facilitando así el desarrollo de la misma.

Método empírico:

- **Observación**: se empleó para caracterizar las soluciones, teniendo en cuenta distintos datos de otras aplicaciones y así establecer los requisitos con las principales funcionalidades de estos.
- **Consulta de fuentes de información**: Método empírico que permite tener acceso a la información disponible sobre las bibliotecas para el desarrollo de la visión por computador y sus funcionalidades.

CAPÍTULO 1. Fundamentación teórica

En este capítulo se recogen los principales conceptos vinculados a la investigación. Se caracteriza la metodología de desarrollo que guía la presente investigación y de igual forma se realiza un estudio de sistemas similares, de forma tal que se utilice como punto de partida para la solución propuesta. Por último, se caracterizan las tecnologías y los lenguajes de programación a utilizar durante el desarrollo de la solución.

1.1. Conceptos asociados con el objeto de estudio de la investigación

Con el objetivo de lograr un esclarecimiento del objeto de estudio de la investigación, a continuación, se establecen los principales conceptos asociados al mismo para una mejor comprensión de la presente investigación:

Visión artificial:

La visión artificial industrial es de las tecnologías que cuenta con los sistemas necesarios para capturar y procesar datos a través de imágenes. Sus múltiples aplicaciones y posibilidades para obtener información por medio del análisis de imágenes hacen que este sistema sea muy eficiente en procesos de control de calidad y trazabilidad de productos [1].

Conocida de igual forma como "Visión por Computadora", se denomina como el conjunto de todas aquellas técnicas y modelos que permiten la adquisición, procesamiento, análisis y explicación de cualquier tipo de información espacial del mundo real obtenida a través de imágenes digitales [3].

Detección de anomalías:

En la actualidad, se han desarrollado algoritmos de detección muy fiables. Este hecho ha ayudado a que se puedan descubrir determinadas acciones de los objetivos en la escena de interés. La detección de anomalías, es la identificación de eventos en un escenario que no se relacionan con los esperados. A partir de esto, existen sistemas que facilitan la seguridad en entornos controlados o ayudan en el orden del tráfico.

El objetivo de la detección de anomalías es desarrollar sistemas que sean capaces de aprender cual es el comportamiento normal de unos determinados conjuntos de datos. Esta actividad se realiza mediante modelos estadísticos que son construidos en base a la norma establecida, capaz de generalizar dicho comportamiento de los datos.

Estos modelos son capaces de detectar los puntos anómalos, que se pueden definir como aquellas muestras que caen en regiones de baja densidad o concentración dentro el dominio de los datos de entrada del sistema [1].

Arduino:

Plataforma de prototipos electrónica de código abierto, basada en hardware y software flexibles y fáciles de usar. Se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Toda la plataforma, tanto para sus

componentes de hardware como de software, son liberados con licencia de código abierto que permite libertad de acceso a ellos [5].

Red neuronal artificial:

Las Redes Neuronales Artificiales se inspiran en la estructura y funciones de las neuronas biológicas. Una red neuronal artificial es esencialmente una colección de neuronas interconectadas, agrupadas en capas. Haciendo un paralelo con el esquema recién descrito de procesamiento del cerebro, la neurona artificial recibe distintos valores de entrada que son multiplicados por una ponderación. En el escenario más simple, estos productos son sumados para obtener un valor de salida. La forma más básica de red neuronal se encuentra estrechamente vinculada con las técnicas econométricas de regresión estándar. Este tipo de red simplificada posee dos capas, una de inputs (entrada) y otra de output (salida) [6].

Las Redes Neuronales Artificiales al margen de "parecerse" al cerebro, presentan una serie de características propias del cerebro. Por ejemplo, estas redes aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos y abstraen las características principales de una serie de datos [6].

En las Redes Neuronales Artificiales, la unidad análoga a la neurona biológica es el elemento procesador (PE). Un elemento procesador tiene varias entradas y las combina, normalmente con una suma básica. La suma de las entradas es modificada por una función de transferencia y el valor de la salida de esta función de transferencia se pasa directamente a la salida del elemento procesador [7].

1.2. Visión artificial y redes neuronales para la detección de anomalías

La visión artificial industrial es de las tecnologías que cuenta con los sistemas necesarios para capturar y procesar datos a través de imágenes. Sus múltiples aplicaciones y posibilidades para obtener información por medio del análisis de imágenes hacen que este sistema sea muy eficiente en procesos de control de calidad y trazabilidad de productos [1].

Los sistemas de visión artificial son de los agentes que hacen posible que la tecnología más disruptiva se aplique en diferentes procesos de producción, y que se consigan resultados jamás pensados, debido al uso óptimo de la información. Las cámaras de visión artificial, junto a los softwares de análisis de imagen, permiten que cualquier proceso de la línea de montaje se pueda automatizar. De igual forma permiten que las máquinas actúen de acuerdo a la información que procesan y contrastan con los parámetros con las que han sido programados. De esta forma, la automatización puede ser posible en procesos de identificación de productos, controles de calidad, clasificación y selección de productos, verificación de medidas o detección de defectos o anomalías. En general, la automatización industrial con la intervención de la visión artificial da más

autonomía a las máquinas, ya que estas actúan en concordancia con datos reales, exactos y sin márgenes de error o valoraciones subjetivas [1].

Los objetivos de la visión artificial son esencialmente mejorar la productividad y la calidad en las operaciones de fabricación, disminuir el número de piezas defectuosas, cumplir con las expectativas impuestas por el cliente final, maximizar el rendimiento de la maquinaria, así como detectar, identificar y visualizar defectos antes de añadir valor al producto. Los sistemas de visión artificial deben ser considerados como un elemento para la protección de fallos en el proceso de producción y detectar anomalías antes de añadir valor al producto. En ningún caso podrán ser considerados sistemas para mejorar directamente la producción ya que no intervienen en el proceso de producción [3].

El uso de herramientas de inteligencia artificial, como las redes neuronales artificiales (RNA), es cada vez más habitual en distintos ámbitos industriales, como pueden ser el control de procesos y de la calidad y la predicción de fallos operacionales. Existen herramientas basadas en RNA capaces de facilitar aspectos de la ingeniería de procesos con un esfuerzo de aprendizaje relativamente pequeño, hasta el punto de que algunas de esas herramientas pueden ser usadas como funciones o *plug-in* de hojas de cálculo comerciales, sumando su potencia computacional a las funciones de cálculo y análisis que estas últimas ofrecen. La utilización eficaz de un modelo de RNA requiere de una sistemática, para obtener el máximo partido de la información disponible [40].

Las RNA son un conjunto de técnicas computacionales con una extraordinaria capacidad para ajustar modelos tipo caja negra de un sistema. Su concepción está inspirada en la forma en que las neuronas se conectan e interaccionan entre sí para producir pensamientos y conclusiones. Sin embargo, su diseño y funcionamiento tienen una fuerte base matemática, renunciando a ser una simulación exacta de las corrientes eléctricas y sinapsis de las neuronas biológicas [40].

El funcionamiento básico de una RNA consiste en entrenarla para que aprenda a asociar unas características del sistema (las salidas deseadas) con otras características del mismo (entradas), a partir de datos del sistema que se quiere modelar. Además, presentan una capacidad de generalización que les permite, por un lado, pronosticar salidas (comportamientos) ante entradas (situaciones) que no han visto nunca antes y, por otro, abstraerse del ruido en las señales. En sistemas estocásticos, en los que el desconocimiento de algunas de las variables relevantes lleva a una incertidumbre inherente, esto es especialmente útil [40].

Las características de las RNA las hacen bastante apropiadas para aplicaciones en las que no se dispone a priori de un modelo identificable que pueda ser programado, pero se dispone de un conjunto básico de ejemplos de entrada (previamente clasificados o no). Asimismo, son altamente

robustas tanto al ruido como a la disfunción de elementos concretos y son fácilmente paralelizables. Esto incluye problemas de clasificación y reconocimiento de patrones de voz, imágenes, señales, entre otros. Asimismo, se han utilizado para encontrar patrones de fraude económico, hacer predicciones en el mercado financiero, hacer predicciones de tiempo atmosférico y otros factores más [40].

También se pueden utilizar cuando no existen modelos matemáticos precisos o algoritmos con complejidad razonable, por ejemplo la red de Kohonen ha sido aplicada con un éxito más que razonable al clásico problema del viajante (un problema para el que no se conoce solución algorítmica de complejidad polinómica) [40].

El uso de la visión artificial y las redes neuronales son factores que se evidencian en la detección de anomalías a ciertos objetos.

El objetivo de la detección de anomalías es desarrollar sistemas que sean capaces de aprender cual es el comportamiento normal de unos determinados conjuntos de datos. Mediante modelos estadísticos que son construidos en base a la norma establecida, esta es capaz de generalizar dicho comportamiento de los datos. Estos modelos son capaces de detectar los puntos anómalos, los cuales podemos definir como aquellas muestras que caen en regiones de baja densidad o concentración dentro el dominio de los datos de entrada del sistema [1].

Entre las principales aplicaciones que tiene el detectar anomalías están los sistemas de vigilancia en video, los sistemas contra fraudes bancarios e inclusive son de utilidad como parte del pre-procesamiento de los datos para algoritmos supervisados, ya que estos pueden tener mejoras en cuanto a su precisión considerables al ser eliminados sus datos atípicos.

En la siguiente figura se muestra un ejemplo del uso de detección de anomalías, donde los datos que son usados para entrenar el modelo representan únicamente la norma establecida, peatones caminando para este caso; y como datos de prueba se introducen ciertos eventos u objetos atípicos al conjunto, como las personas en bicicleta o patineta que se pueden observar en la imagen [1].

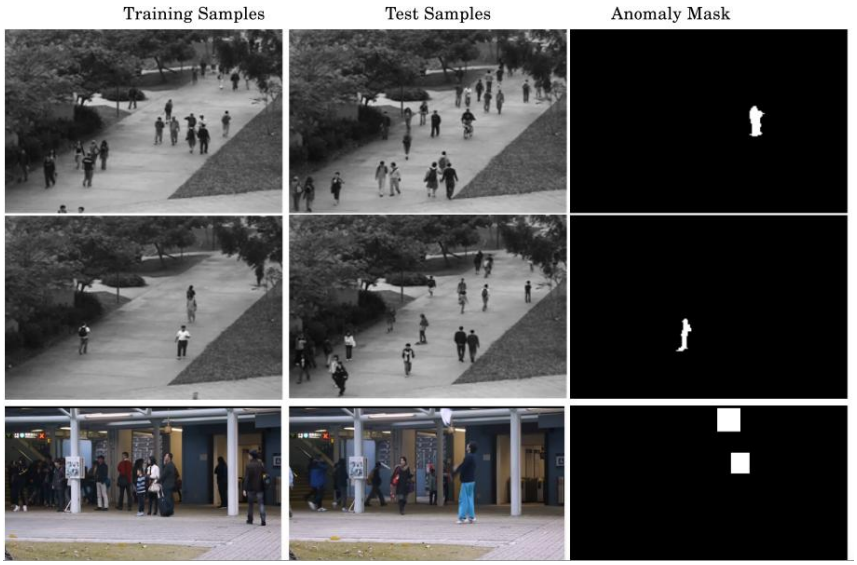


Figura 1: Ejemplo de detección de anomalías en imágenes.

Este tipo de sistemas se pueden considerar como semi-supervisados, ya que desde un inicio conoce cuál es la norma, pero con los datos de prueba, estos pueden estar susceptibles a poseer características atípicas al conjunto [1].

El *machine learning* o aprendizaje automático es un tipo de inteligencia artificial que dota a los ordenadores de la habilidad de aprender y llevar a cabo tareas determinadas sin haber sido explícitamente programados para ello. Cada vez más investigadores lo ven como el camino idóneo para alcanzar una inteligencia artificial a nivel humano [41].

Considerando una tarea de clasificación de una imagen en distintas categorías, si se realiza una aproximación tradicional el primer paso sería una extracción manual de características de la imagen. Estas técnicas no funcionan del todo cuando se aplican directamente a las imágenes, puesto que no se conoce la estructura completa y la composición natural de las mismas [41].

Actualmente ha surgido una nueva tendencia de extraer estas características mediante el uso de *deep learning*, una técnica de *machine learning* que puede aprender representaciones útiles directamente de datos como imágenes, texto y audio. Estas características son de naturalezas distintas (texturas, colores, ejes u otras más específicas) [41]. Dentro de las técnicas que se utilizan para el uso del *deep learning*, se tendrá en cuenta en la presente investigación la técnica de la red neuronal convolucional.

En una clasificación de imágenes mediante técnicas de deep learning, como es el caso de una red neuronal convolucional (CNN), el algoritmo aprende estas características de las imágenes de forma automática generando representaciones de bajo nivel en las primeras capas de la red

neuronal convolucional como ejes y esquinas, a representaciones más específicas del problema en capas superiores. Es aquí donde se introduce la diferencia, puesto que los algoritmos de *deep learning* llevan a cabo de forma natural el aprendizaje de principio a fin. Esto significa que las capas de la red neuronal convolucional llevan a cabo el proceso tanto de extracción de representaciones como de clasificación (última capa de la red neuronal) [41].

En la figura que se muestra a continuación se evidencia una aproximación de *deep learning* en la clasificación de imágenes, a través de la extracción de características y clasificación automática mediante un sistema de caja negra [41]:

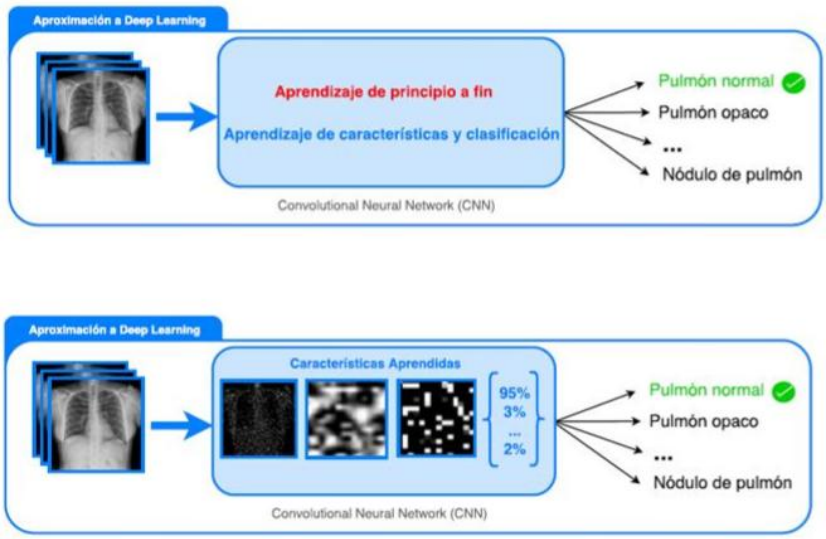


Figura 2: Aproximación de *deep learning* en la clasificación de imágenes.

Teniendo en cuenta los elementos descritos anteriormente, el autor del presente trabajo propone la utilización de la técnica *machine learning*, y dentro de esta el *deep learning*, tomando como referencia de la misma, la red neuronal convucional para darle solución al problema que da paso a esta investigación. A continuación, se explica dicha técnica.

1.2.1. Red Neuronal Convolucional

Una red neuronal convolucional (CNN o Convnet) es una poderosa técnica de *deep learning*, que se define básicamente como una secuencia de múltiples capas de características. Cada capa transforma un volumen de entrada 3D en un volumen de salida 3D, a través de funciones diferenciables que pueden tener o no parámetros. Las capas de la red se constituyen de neuronas dispuestas en tres dimensiones (anchura, altura y profundidad) que tienen pesos y sesgos que se pueden aprender. Cada neurona recibe entradas, lleva a cabo un producto escalar (entre la entrada y el peso) y a veces puede seguirse con una no linealidad [41].

Dos caminos son posibles a la hora de usar las CNNs en reconocimiento visual por ordenador. Una opción es entrenar una red neuronal desde 0, lo que implicaría disponer de miles de millones de imágenes de entrenamiento. Estas se utilizarían con el fin de aprender los pesos necesarios para que, de forma automática, la red sea capaz de reconocer una imagen no usada en el entrenamiento. Sin embargo, este procedimiento requiere un conocimiento profundo en el campo, para poder escoger los parámetros e hiperparámetros que no conduzcan a un sobreajuste. Este camino podría llevar días, o incluso semanas de entrenamiento [41].

Otra alternativa sería utilizar una red neuronal convolucional como extractor de características, recomendable en conjuntos no muy extensos de imágenes de entrenamiento (de cientos a miles). En la literatura, a este caso se le suele denominar transferencia de conocimiento de un problema a otro. La transferencia de conocimiento se define como la habilidad de un sistema a reconocer y aplicar conocimiento y habilidades aprendidas en tareas previas en una tarea nueva. Estudios como (Pan y Yang, 2010; Zeiler y Fergus, 2013; Oquab et al, 2014) han puesto en evidencia la efectividad de esta técnica en tareas de reconocimiento visual [41].

En otras palabras, se puede definir a la transferencia de conocimiento como el hecho de enseñar a una persona a detectar diferencias entre objetos. Si se le enseña a fijarse en las formas, colores, organización de los mismos, a partir de muchos ejemplos, cuando ya lo ha aprendido, es capaz de hacerlo para otros tipos de imágenes nuevas y diferenciarlas.

La extracción de características es el proceso de obtener información discriminante de una imagen en una forma reducida normalmente conocida como vector de características. Una CNN bien entrenada puede servir como extractor de características. Se puede tomar un modelo CNN entrenado en un problema distinto, donde la CNN puede aprender a reconocer objetos determinados que no coinciden con los que se desean clasificar. Se puede utilizar este modelo como extractor de características internamente de una forma composicional para una tarea en concreto, y luego usar estas características para entrenar un clasificador de imágenes [41].

A continuación, se muestra una CNN entrenada para reconocer clases como barco, bicicleta y coche. Para esta tarea en concreto, en lugar de extraer características de forma manual, se transfiere el conocimiento de la CNN entrenada. Extrayendo así internamente características de las clases de la nueva tarea de reconocimiento que se utilizaran para entrenar un clasificador [41].

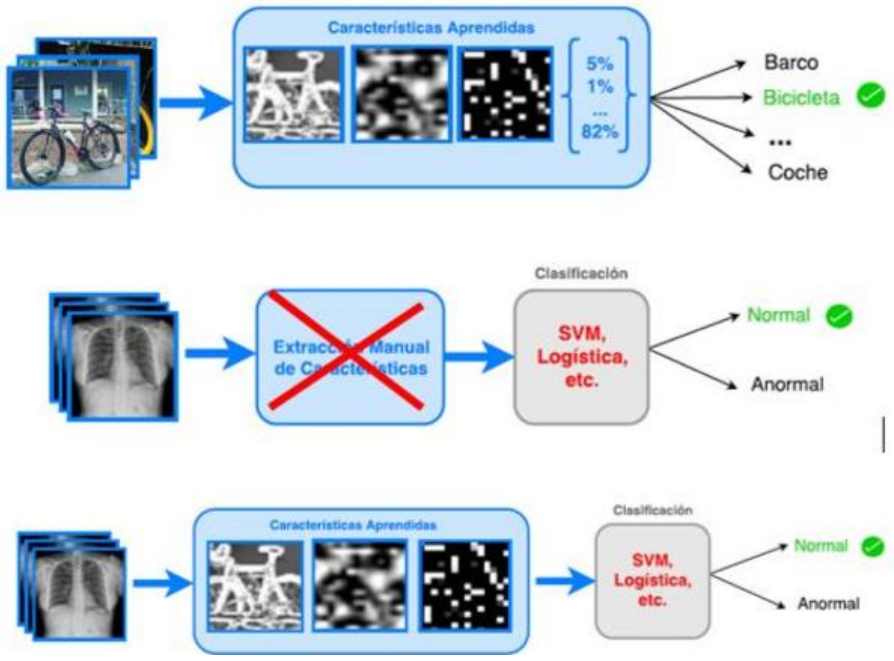


Figura 3: CNN entrenada para reconocer clases como barco, bicicleta y coche [41].

En una red neuronal en caso de que las entradas sean imágenes completas, es decir, entradas de alta dimensión, las redes neuronales no funcionan bien puesto que la conectividad total de las neuronas de una capa con las de la siguiente es un desperdicio (dejando sin aprovechar la estructura de la imagen. Ejemplo de ello son los píxeles adyacentes de una imagen que tienden a estar muy correlados mientras que los que están separados no, y el gran número de parámetros conduciría a un sobreajuste. En este caso se usan las redes neuronales convolucionales. Estas últimas suponen explícitamente que las entradas son imágenes, lo que permite codificar ciertas propiedades de las mismas en la arquitectura, haciendo la función hacia adelante más eficiente de implementar y reduciendo de forma considerable los parámetros de la red [41].

Hay cinco capas principales que se apilan para construir arquitecturas de redes neuronales convolucionales completas, transformando los valores de los píxeles de la imagen original a unas puntuaciones de clase finales. Estas capas son [41]:

- Entrada: Mantiene los valores de los píxeles de la imagen de entrada (el alto, el ancho y los tres canales de color).
- Convolucionales: Es el bloque central de construcción de la red. Sus parámetros consisten en un conjunto de filtros que se pueden aprender. Estos filtros son pequeños espacialmente (ancho y largo) pero se extienden a lo largo de toda la profundidad del volumen de entrada. La ConvNet aprenderá los filtros que se activan cuando algún tipo de características específicas se detectan en la entrada. Esta capa calcula la salida de las

neuronas conectadas a regiones locales en la entrada, siendo cada cálculo un producto escalar entre los pesos y la pequeña región a la que se conectan del volumen de entrada.

- *Pooling* o submuestreo: Esta capa reduce el tamaño espacial de la entrada para disminuir el número de parámetros reduciendo las respuestas convolucionales, disminuir la computación en la ConvNet y proporcionar un grado de invarianza translacional al modelo. Para ello se lleva a cabo una operación de reducción de muestreo a lo largo de las dimensiones espaciales (ancho y alto). Se toman las respuestas convolucionales y se dividen en x·y bloques. Tras aplicarle el *pooling* a estas respuestas, se tiene en cuenta un mapa de x·y respuestas, cada una representativa de un bloque tomando de cada uno de estos bloques se toma el mayor valor (*max pooling*) o el promedio (*mean pooling*).
- Totalmente conectadas: Neuronas entre dos capas adyacentes están todas conectadas por parejas, mientras las neuronas de la misma capa no. Pueden ser interpretadas como capas convolucionales de 1x1. Estas capas calcularán la puntuación de clase.
- ReLU (no linealidad): Esta capa calcula la función de f(x)=max(0,x), que umbraliza en 0. Esta capa mantiene el tamaño del volumen sin cambios.

Las capas convolucional y totalmente conectada llevan a cabo transformaciones que son función de la activación en la entrada y de los parámetros de estas capas: pesos y sesgos de las neuronas. Estos parámetros se entrenan con descenso de gradiente de modo que las puntuaciones de clase que la CNN calcula son consistentes con las etiquetas en el conjunto de entrenamiento para cada imagen [41].

Ahora, con las redes neuronales convolucionales hay varias neuronas de la misma capa convolucional que intercambian pesos, lo que significaría la evaluación del mismo filtro sobre varias subventanas de la imagen de entrada. De este modo la red neuronal convolucional podría entenderse como el aprendizaje más efectivo a un conjunto de filtros. Que la red pueda aprender una codificación general se podría conseguir o características de los datos de entrada se podría conseguir haciendo uso del mismo conjunto de filtros sobre la imagen entera. Con el objetivo de conseguir una generalización de la CNN se restringen los pesos a ser iguales a través de diferentes neuronas teniendo así un efecto de normalización sobre la CNN [41].

El uso de esta técnica en la visión artificial para la clasificación y reconocimiento de imágenes, así también como para la detección de anomalías es de suma importancia ya que puede resultar de gran ayuda en aplicaciones como sistemas de vigilancia, detección de anomalías en ciertos objetos, en aplicaciones médicas con el control se ciertos signos de saludo del paciente o inclusive en aplicaciones agrícolas con la detección de enfermedades o plagas en los cultivos.

1.3. Sistemas similares

Existen innumerables sistemas especializados y diseñados para la detección de anomalías utilizando la Visión Artificial.

La Visión Artificial la componen un conjunto de procesos destinados a realizar el análisis de imágenes. Estos procesos son: captación de imágenes, memorización de la información, procesado e interpretación de los resultados [4].

Con la visión artificial se pueden automatizar tareas repetitivas de inspección realizadas por operadores, realizar controles de calidad de productos que no era posible verificar por métodos tradicionales, hacer inspecciones de objetos sin contacto físico, reducir el tiempo de ciclo en procesos automatizados y realizar inspecciones en procesos donde existe diversidad de piezas con cambios frecuentes de producción [4].

Las principales aplicaciones de la visión artificial en la industria actual son: la identificación e inspección de objetos, la determinación de la posición de los objetos en el espacio, el establecimiento de relaciones espaciales entre varios objetos (guiado de robots), la determinación de las coordenadas importantes de un objeto, la realización de mediciones angulares y las mediciones tridimensionales [8].

Utilizando la visión artificial se puede detectar puntos anómalos, los cuales se pueden definir como aquellas muestras que caen en regiones de baja densidad o concentración dentro el dominio de los datos de entrada del sistema. El objetivo de la detección de anomalías es desarrollar sistemas que sean capaces de aprender cual es el comportamiento normal de unos determinados conjuntos de datos. Mediante modelos estadísticos que son construidos en base a la norma establecida, esta es capaz de generalizar dicho comportamiento de los datos.

Entre las principales aplicaciones que tienen como objetivos detectar anomalías están los sistemas de vigilancia en video y en imágenes; donde el análisis reside en las variables de espacio y tiempo [8]. A continuación, se describen los sistemas similares estudiados.

1.3.1. Aplicación Android que detecta objetos para una imagen tomada de la cámara

Esta es una aplicación de código abierto realizada por el desarrollador indio Amit Shekhar. Este es un proyecto para integrar la biblioteca Tensorflow Lite en una aplicación Android, capaz de detectar objetos para una imagen tomada de la cámara [9]. A continuación, se muestra un ejemplo del funcionamiento de esta aplicación:

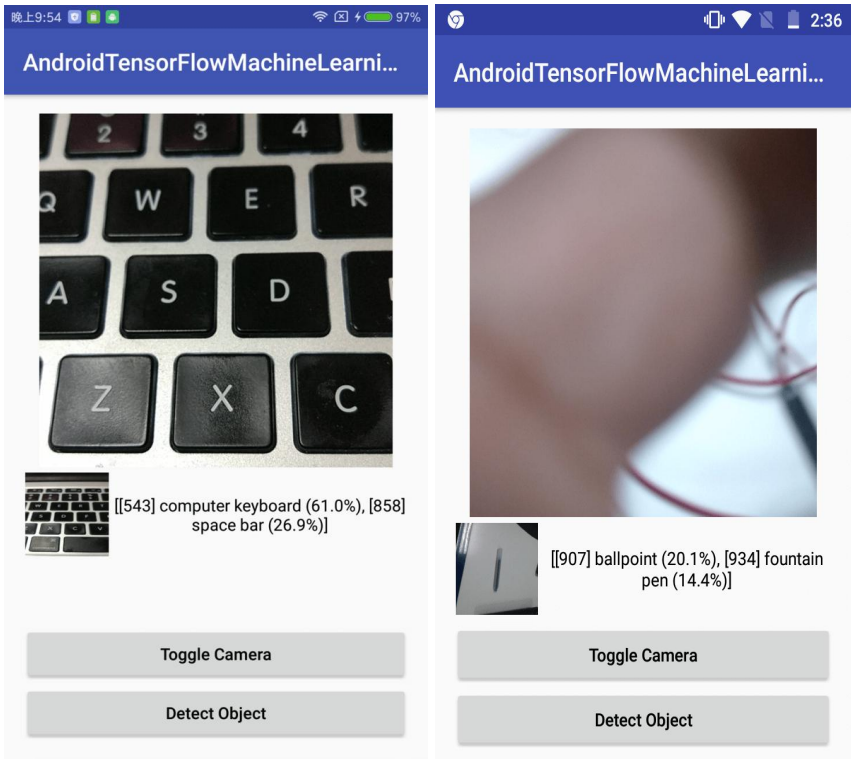


Figura 4: Funcionamiento de la aplicación AndroidTensorFlowLearning

1.3.2. Aplicación Android que detecta objetos en tiempo real utilizando la cámara del dispositivo

Esta es una aplicación realizada por el desarrollador indio Prabhakar Thota. Es la demostración de una aplicación de cámara que clasifica las imágenes continuamente utilizando un modelo cuantificado de Mobilenet o un modelo Inception-v3 de punto flotante. Para ejecutar la demostración, se requiere un dispositivo con Android 5.0 (API 21) o superior [9].

En esta demostración, la inferencia se realiza utilizando la API Java TensorFlow Lite, la cual clasifica los marcos en tiempo real, mostrando las clasificaciones más probables. También muestra el tiempo necesario para detectar el objeto [9]. A continuación, se muestra un ejemplo del funcionamiento de esta aplicación:

Figura 5: Funcionamiento de la aplicación desarrollada por Prabhakar Thota

1.3.3. Aplicación Android "Visión General"

Esta es una aplicación de cámara que detecta continuamente los objetos (cuadros delimitadores y clases) en los marcos vistos por la cámara trasera de su dispositivo, utilizando un modelo cuantificado SSD MobileNet entrenado en el conjunto de datos COCO. Estas instrucciones lo guían a través de la creación y ejecución de la demostración en un dispositivo Android.

Los archivos de modelo se descargan a través de scripts de Gradle cuando compila y ejecuta. La aplicación puede ejecutarse en el dispositivo o en el emulador. Para su desarrollo se requiere de un entorno de desarrollo de Android con un mínimo de API 21 y Android Studio 3.2 o posterior [9].

A continuación, se muestra un ejemplo del funcionamiento de esta aplicación:

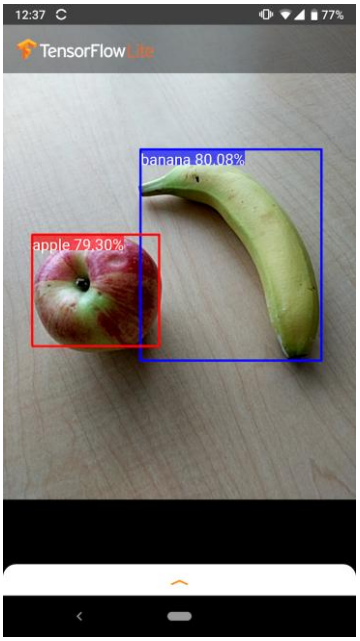


Figura 6: Funcionamiento de la aplicación Android "Visión General"

Resultado del análisis realizado a los sistemas existentes

Los sistemas anteriores no cumplen con el objetivo del problema planteado en la presente investigación, ya que no son capaces de identificar anomalías en una placa Arduino. Sin embargo, este estudio permitió determinar que poseen características comunes a la solución propuesta, ya que son aplicaciones que utilizan la visión artificial, mediante la librería Tensorflow, así como la cámara digital para la identificación de determinado punto. En el caso de la aplicación AndroidTensorFlowLearning se tuvo en cuenta su diseño, principalmente la organización de la información en las interfaces y los botones en general.

1.4. Metodología de desarrollo

Las metodologías de desarrollo de software son un conjunto de procedimientos, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un sistema informático [27]. Además, es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado [28].

1.4.1. Metodología AUP-UCI

Dado que no existe una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto exigiéndose así que el proceso sea configurable, nace en la Universidad de las Ciencias Informáticas (UCI) la metodología AUP-UCI como una variación de la metodología AUP (Proceso Unificado Ágil) para ser aplicada en los proyectos de desarrollo de software producidos en dicha institución.

Esta metodología está formada por tres fases, (Inicio, Ejecución y Cierre) para el ciclo de vida de los proyectos de la universidad, las cuales contienen las características de las cuatro fases (Inicio, Elaboración, Construcción y Transición) propuestas en AUP. Las características de las fases de la metodología de la universidad son:

Inicio: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el software es transferido al ambiente de los usuarios finales o entregado al cliente junto con la documentación. Además, en esta transición se capacita a los usuarios finales sobre la utilización de la aplicación.

Cierre: En el cierre se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto [17].

Las disciplinas definidas en la variación de AUP para la UCI se desarrollan en la Fase de Ejecución, de ahí que en la misma se realicen iteraciones y se obtengan resultados incrementales. En la siguiente figura se muestra como sería la ejecución de esta metodología según sus fases e iteraciones:

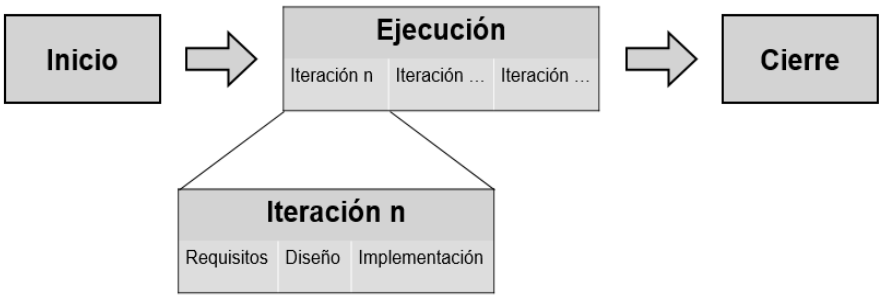


Figura 7: Fases de la Metodología AUP-UCI [28]

Esta metodología, además propone tres variantes a utilizar en los proyectos para el Modelo de Negocio: Caso de Uso de Negocio (CUN), Descripción de Proceso de Negocio (DPN) y Modelo Conceptual (MC) y tres formas de encapsular los requisitos: Caso de Uso del Sistema (CUS), Historias de Usuario (HU) y Descripción de Requisitos por Proceso (DRP). De igual forma,

establece cuatro escenarios para modelar el sistema en los proyectos manteniendo en dos de ellos el MC, como se describe a continuación:

Escenario No 1: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema.

Escenario No 2: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio.

Escenario No 3: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad.

Escenario No 4: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos.

Siguiendo la política de desarrollo de software de la institución, se define como metodología a emplear la AUP-UCI en el escenario número 4, debido a la necesidad de una metodología que responda con facilidad a los cambios continuos, por estar el cliente siempre acompañando el desarrollo y por estar bien definido el negocio.

1.5. Herramientas y tecnologías utilizadas

Las herramientas informáticas, son programas, aplicaciones o simplemente instrucciones empleadas para efectuar otras tareas de modo más sencillo [34]. Seguidamente se describen las tecnologías y herramientas empleadas en la presente investigación, realizando un análisis de las principales características de cada una de ellas.

1.5.1. Herramienta de modelo

Visual Paradigm es una herramienta CASE: Ingeniería de Software Asistida por Computación. La misma propicia un conjunto de funcionalidades que apoyan al desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación [35].

Además, Visual Paradigm soporta los últimos estándares de la notación UML y todos los productos de éste están diseñados y desarrollados para eliminar la complejidad, mejorar la productividad y comprimir el desarrollo de software según plazos emitidos por los clientes. Permite representar gráficamente varios diagramas y facilita la generación de código. Es una herramienta multiplataforma, fácil de instalar y utilizar (Visual Paradigm, 2017).

1.5.2. Lenguajes de programación

Para programar existen varios lenguajes de programación como son Java, C#, C++, Python y Kotlin. Para la realización de la red neuronal se utilizó el lenguaje Python y para el desarrollo de Android el lenguaje Kotlin.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código fuente. Se trata de un lenguaje de programación multiparadigma, ya que soporta programación orientada a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico, multiplataforma y posee una licencia de código abierto, denominada Python Software Foundation License [10].

Desarrollar redes neuronales con Python permite reducir los tiempos de trabajo, lo que cual se traduce en una disminución de costes muy importante para las empresas. Esto es posible gracias a las diferentes ventajas que este lenguaje de programación aporta a nivel de desarrollo [10].

Entre las principales ventajas de este lenguaje para crear redes neuronales se tiene, el acceso a la comunidad abierta de desarrolladores, la gran disponibilidad de recursos online, gratuitos y muy potentes. Ejemplo de estos son los materiales escritos por el mismo Guido van Rossum y todos los contenidos publicados en las distintas comunidades [11].

Además, tiene la posibilidad de utilizar múltiples librerías gratuitas de Python que permiten acelerar los procesos de desarrollo tales como las librerías para el aprendizaje automático en TensorFlow de Google y librerías para el aprendizaje profundo como las de PyTorch de Facebook y Keras. Este lenguaje tiene la facilidad de aprenderse desde cero en tiempo récord debido a su gran similitud con el idioma inglés, brinda la posibilidad de elegir entre Programación Orientada a Objetos o secuencias de comandos y usa multiplataforma como Windows, MacOS, Linux, Unix, entre otras. Además, se puede acceder a paquetes como PyInstaller, los cuales permiten preparar el código para distintas plataformas [11].

Las Redes Neuronales Artificiales con Python para su funcionamiento intervienen tres tipos de nodos. El primero son los nodos de entrada que reciben los datos desde el exterior (input) y estos datos son los que la red debe procesar. Los segundos nodos llamados ocultos son los que están en el interior de la red y no tienen contacto con el exterior. Estos reciben los datos de los nodos de entrada y luego se encargan de transmitirlos a lo largo de toda la red para su procesamiento. Por

último, son los nodos de salida que reciben los datos de los nodos ocultos y son los encargados de enviarlos hacia el exterior de la red como información ya procesada (*output*) [11].

Kotlin es un lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript. Aunque no tiene una sintaxis compatible con Java, Kotlin está diseñado para interoperar con código Java y es dependiente del código Java de su biblioteca de clases, tal como pueda ser el entorno de colecciones de Java. Kotlin para Android presenta muchas ventajas como lenguaje moderno para el desarrollo de aplicaciones móviles siendo un lenguaje sin muchas restricciones. Entre sus principales características se tiene [12]:

- Compatibilidad: Kotlin es totalmente compatible con el JDK a partir de la versión 6, esto es importante ya que garantiza que Kotlin puede ejecutarse en dispositivos Android antiguos sin problemas. Además, es compatible con Android Studio y con las herramientas de compilación de Android.
- Performance: Una aplicación de Kotlin es tan rápida como una equivalente de Java, gracias a una estructura de bytecode (código intermedio más abstracto que el código máquina) muy similar. Con el soporte de Kotlin para funciones en línea, el código que usa lambdas (función anónima) y a menudo se ejecuta incluso más rápido que el mismo código escrito en Java.
- Interoperabilidad: Kotlin es cien por ciento interoperable con Java, permitiendo utilizar todas las bibliotecas existentes de Android en una aplicación de Kotlin. Esto incluye el procesamiento de anotaciones, por lo que el DataBinding y Dagger funcionan de igual forma.
- Tiempo de compilación: Kotlin es compatible con la compilación incremental eficiente, por lo que, aunque hay alguna sobrecarga adicional para las compilaciones limpias, las creaciones incrementales suelen ser tan rápidas o más rápidas que con Java.
- Curva de aprendizaje: Para un desarrollador de Java, empezar con Kotlin es muy fácil. El convertidor automatizado de Java a Kotlin incluido en el complemento Kotlin ayuda con los primeros pasos [12].

Todos los lenguajes tienes sus propias ventajas y desventajas. Sin embargo, en desarrollo de aplicaciones móviles, Python no tiene uso práctico. En Android, si es para desarrollo nativo, se utiliza Kotlin y Java. Python es utilizado para análisis de datos y desarrollo web.

Python es un lenguaje que dista mucho de Kotlin en varios aspectos. Kotlin generalmente es un lenguaje oficial orientado al desarrollo Android, que ofrece varias mejoras a Java, y Python es un lenguaje de propósito general, pero que poco tiene que ver con el desarrollo de aplicaciones móviles.

Python es más preferible para la inteligencia artificial que Java por varios factores: Python abarca una gran comunidad, para que un lenguaje sea preferible sobre otro debe tener una gran comunidad que lo respalde en dicha área, en esto Python en superior a Java. Python posee una mayor cantidad de bibliotecas, por eso es claro que hay una mayor cantidad de recursos open-source disponibles en Python que en Java para trabajar con inteligencia artificial. Además, al ser Python un lenguaje dinámico y con un tipado débil, los prototipos se pueden programar más rápido, esto permite desarrollar con mayor agilidad a diferencia de Java, donde se debe ser más explícito con los tipos de datos. Python posee mayores recursos para aprender inteligencia artificial, hay muchas plataformas de educación online, cursos, libros y artículos de blog que aparecen cada día para aprender inteligencia artificial con Python.

Comparativo general entre Kotlin y Java:

JetBrains decide crear un nuevo lenguaje que se ejecute sobre la máquina virtual de Java anteriormente nombrada y decide crearlo con el objetivo de que compile con la misma velocidad como el propio Java en su máquina virtual.

El líder de JetBrains en el año 2011 comentó que ningún lenguaje tenía las características que él buscaba y creó Kotlin basándose en algunos pilares clave como los son:

- Concisión frente a la verbosidad presente en un lenguaje con años como es Java. Un ejemplo claro es la creación de clases con los métodos habituales get (obtener), set (modificar), equals (iguales), hashcode (código hash) y copy(copiar) en una sola línea con la sentencia “data class” (clase dato).
- Mayor seguridad al intentar evitar errores en tiempo de compilación con su notada característica Null Safe, gestionando los valores nulos de manera segura garantizando así la no aparición de NullPointerException (NPE) de Java. La única forma de obtener NPE en Kotlin es: llamando la excepción explícitamente, usando la inconsistencia de datos o con la interoperabilidad con Java, otro pilar clave.
- Interoperabilidad, sin duda la más importante, la facilidad de transformación de proyectos Java a Kotlin y su posible convivencia en una misma aplicación. No tienen una sintaxis compatible, pero Kotlin sí está diseñado para interoperar con Java. Kotlin además depende de la biblioteca de clases de Java [13].

1.5.3. Bibliotecas para el desarrollo de la visión por computador

A continuación, se realiza una descripción de las bibliotecas a utilizar como base para el desarrollo de la solución propuesta, que a su vez utilizan técnicas de visión por computadora.

Tensorflow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de

construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento empleados por los humanos. Actualmente es utilizado tanto en la investigación como en los productos de Google frecuentemente reemplazando el rol de su predecesor de código cerrado, DistBelief. TensorFlow fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 [14].

TensorFlow es el sistema de aprendizaje automático de segunda generación de Google Brain, liberado como software de código abierto el 9 de noviembre de 2015. Mientras la implementación de referencia se ejecuta en dispositivos aislados, TensorFlow puede correr en múltiple CPUs[1] y GPUs[2] (con extensiones opcionales de CUDA (Arquitectura Unificada de Dispositivos de Cómputo) para informática de propósito general en unidades de procesamiento gráfico). TensorFlow está disponible en Linux de 64 bits, macOS, y plataformas móviles que incluyen Android e iOS [14].

Los cómputos de TensorFlow están expresados con estados, flujo de datos y gráficos. El nombre TensorFlow deriva de las operaciones que tales redes neuronales realizan sobre arreglos multidimensionales de datos. Estos arreglos multidimensionales son referidos como "tensores" [14].

Keras es una biblioteca de Redes Neuronales de código abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Además, contiene varias implementaciones de los bloques constructivos de las redes neuronales como por ejemplo los layers, funciones objetivo, funciones de activación y optimizadores matemáticos. Su código está alojado en GitHub y existen foros y un canal de Slack de soporte. Además del soporte para las redes neuronales estándar, Keras ofrece soporte para las Redes Neuronales Convolucionales y para las Redes Neuronales Recurrentes. De igual forma permite generar modelos de aprendizaje profundo en móviles tanto sobre iOS como sobre Android, sobre una máquina virtual de java o sobre web [15].

1.5.4. Entorno de desarrollo integrado

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android.

[1] Unidad Central de Procesamiento
[2] Unidad de Procesamiento Gráfico

Entre las principales ventajas de esta herramienta está la capacidad de ejecutar las compilaciones de forma muy rápida, el renderizado de layouts lo ejecuta en tiempo real y tiene la posibilidad de usar parámetros. Además, ejecuta en tiempo real el funcionamiento de la aplicación gracias al emulador y tiene la capacidad de asociar archivos y carpetas de forma automática en la aplicación, así como eliminar archivos y crear carpetas en valores [16].

1.6. Conclusiones del capítulo

El establecimiento de los principales conceptos asociados a la detección de anomalías a través de la visión artificial permitió obtener un mayor conocimiento sobre los elementos que engloban el dominio del problema de la presente investigación. De igual forma, el estudio y análisis referente a la visión artificial y las redes neuronales, permitió la selección de la red neuronal convencional para darle respuesta al problema planteado. Por su parte, el análisis de los diferentes sistemas homólogos evidenció la necesidad de desarrollar una aplicación que permita la detección de anomalías en una placa Arduino con el uso de visión por computadoras. Además, la caracterización de la metodología de desarrollo AUP en su variación para la UCI fundamentó su selección para guiar el desarrollo de la solución propuesta, partiendo del principio de estar en relación con las políticas que sigue la universidad. Por último, el análisis de las principales características de las herramientas y tecnologías permitió seleccionar las que mejor se ajustan para el desarrollo de una aplicación que permita la detección de anomalías en una placa Arduino con el uso de visión por computadoras.

CAPÍTULO 2. Solución propuesta

En este capítulo se presentan el análisis y diseño de la solución propuesta. Esta incluye la descripción de las funcionalidades a través de los requisitos funcionales y no funcionales, las historias de usuario y aspectos de diseño (patrón arquitectónico, los patrones de diseño y diagramas).

2.1. Descripción de la solución propuesta

En el presente trabajo se propone desarrollar una aplicación destinada al Centro de Informática Industrial, que posibilite identificar las anomalías en una placa Arduino, a través de la captura de pantalla que se le realice a dicha placa desde un dispositivo móvil. Una vez capturada la imagen de la placa a través de la interfaz de la propuesta de solución, la misma detectará las anomalías que presente esta placa, notificando si se encuentra en buen o mal estado.

Para la realización de esta aplicación se tiene en cuenta el estudio realizado en el primer capítulo del presente documento sobre la red neuronal convolucional, con el objetivo de entrenar el modelo para diferenciar las imágenes de placas Arduinos, y así definir el estado de las mismas. Este entrenamiento se realizará con diferentes conjuntos de imágenes, con el fin de crear una aplicación móvil que permita la detección de las anomalías en estas placas. Para la creación de la red neuronal convolucional (CNN), se utilizarán las librerías TensorFlow y Keras las cuales permiten trabajar bajo el lenguaje de Python.

2.2. Captura de requisitos

“Los requisitos de un software son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información” [19]. Se pueden clasificar en funcionales y no funcionales. Los requisitos funcionales describen los servicios (funciones) que se esperan del sistema. En cambio, los requisitos no funcionales son restricciones sobre los requisitos funcionales identificados.

El levantamiento de requisitos es una etapa esencial en el inicio de todo proyecto de desarrollo de software y debe realizarse efectivamente para poder aumentar la posibilidad en el éxito de los proyectos. A continuación, se muestran los requisitos funcionales (RF) y no funcionales (RFN) obtenidos para la solución propuesta.

2.2.1. Requisitos funcionales

Los requisitos funcionales son “las declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a las entradas particulares y de cómo se debe

comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer" [19].

Tabla 1: Descripción de los requisitos identificados

No	Nombre	Descripción	Prioridad
RF.1	Entrenar modelo	Se entrena el modelo para identificar y clasificar imágenes de arduinos.	Alta
RF. 2	Exportar modelo al formato. Tflite	Se exporta el modelo a formato tfilite para utilizarlo en la aplicación.	Alta
RF.3	Acceder a la cámara del dispositivo móvil	La aplicación debe tener acceso a la cámara del dispositivo .	Media

RF.4	Establecer las capturas de pantalla en la interfaz gráfica de la aplicación	La interfaz gráfica de la aplicación debe tener permisos para establecer capturas de pantalla.	Alta
RF.5	Acceder al almacenamiento local	Al realizar la captura de pantalla la aplicación debe de tener permiso para guardar la imagen en el almacenamiento local.	Media
RF.6	Informar al usuario de los fallos ocurridos y debe tener en cuenta la recuperación frente a fallos	La aplicación debe de informar al usuario de los fallos ocurridos en el proceso.	Media

RF.7	Visualizar los elementos y las ayudas mediante la pantalla táctil del móvil	Permite mostrar el contenido de los elementos y las ayudas en la pantalla táctil.	Baja
RF.8	Proporcionar la interacción con el usuario por medio de interfaces táctiles	La interfaz táctil de la aplicación presenta los elementos necesarios para que le usuario pueda interactuar con la misma.	Media
RF.9	Identificar imagen	Una vez capturada la imagen el modelo identificara la imagen para saber si es una placa Arduino.	Alta

RF.10	Clasificar imagen	Una vez capturada la imagen el modelo clasificará la imagen en buen estado o en mal estado.	Alta

2.2.2. Requisitos no funcionales

Los requisitos no funcionales son “restricciones de los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema” [19].

RNF1. Requerimientos de software

- Para la utilización del sistema se requiere el SO Android en su versión 5.1 o superior.

RNF2. Requerimientos de hardware

- Capacidad disponible de 25 Mb en la memoria de teléfonos inteligentes para la instalación y ejecución de la aplicación.

RNF3. Requerimiento de apariencia o interfaz externa

- La aplicación debe poseer una interfaz sencilla y amigable, además de presentar información de forma clara y organizada.

Como se fundamentó en el Capítulo 1 de este documento, para el encapsulamiento de los requisitos funcionales se tendrá en cuenta el escenario 4 propuesto por la metodología que guía el proceso de desarrollo de la solución propuesta, el cual plantea que los mismos se describirán a través de las Historias de usuarios. A continuación, se describe dicha actividad.

2.3. Historias de usuario

Las historias de usuario son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento se

pueden modificar, reemplazar por otras más específicas o generales o añadirse nuevas. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla [20]. A continuación, se describen tres de las HU de prioridad alta para el cliente.

Tabla 2: HU_ Establecer las capturas de pantalla en la interfaz gráfica de la aplicación

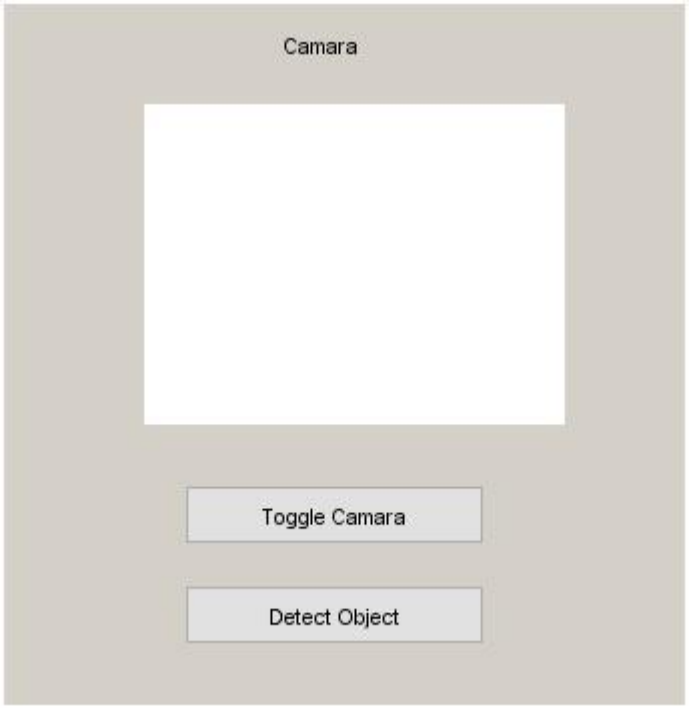
Número: 4	**Requisito**: Establecer las capturas de pantalla en la interfaz gráfica de la aplicación
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: -	**Tiempo Real:** 6 días
Descripción: Realizar captura de pantalla a travéz de la interfaz gráfica de la aplicación al objeto que se va a identificar. **Objetivo**: Establecer las capturas de pantalla en la interfaz gráfica de la aplicación	
Observaciones: **Prototipo de la interfaz:**  	

Tabla 3: HU_ Identificar imagen

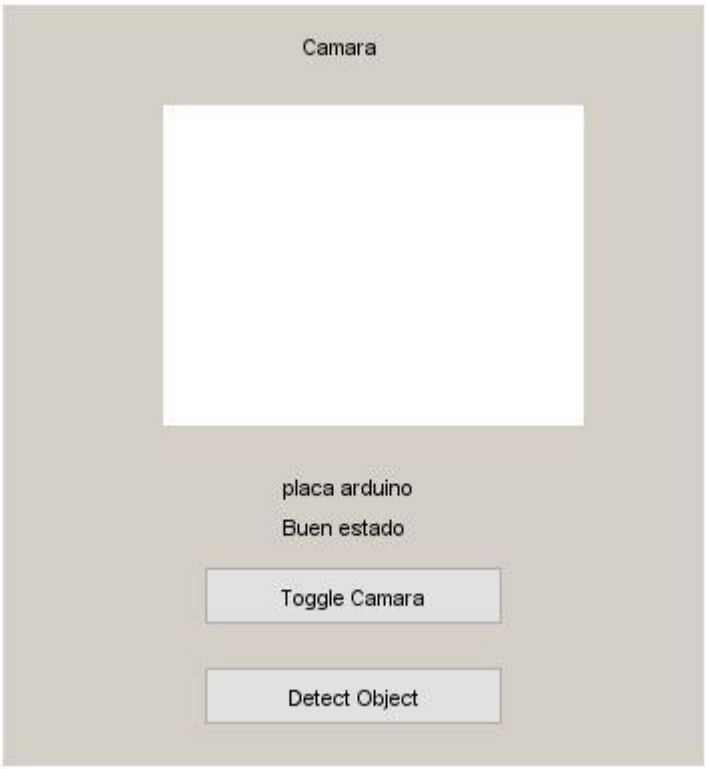
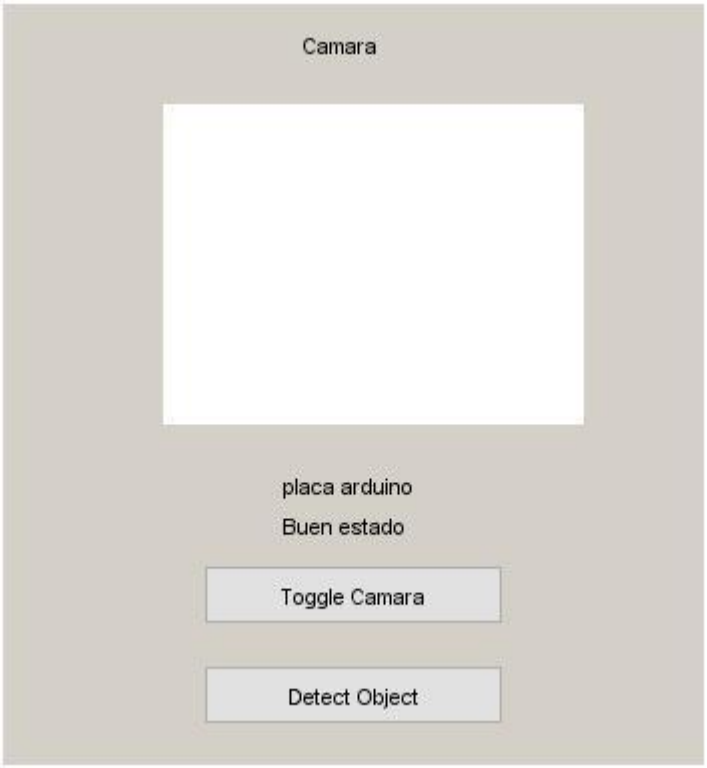
Número: 10	**Requisito**: Identificar imagen
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: -	**Tiempo Real:** 8 días
Descripción: Identificar la imagen una vez que halla sido capturada por la interfaz de la aplicación. **Objetivo**: Identificar imagen.	
AObservaciones: **Prototipo de la interfaz:** 	

Tabla 4: HU_ Clasificar imagen

Número: 11	**Requisito**: Clasificar imagen

Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: -	**Tiempo Real:** 8 días
Descripción: Clasificar la imagen una vez que halla sido capturada por la interfaz de la aplicación. **Objetivo**: Clasificar imagen.	
AObservaciones: **Prototipo de la interfaz:** 	

2.4. Diseño de la solución propuesta

"La esencia del diseño del software es la toma de decisiones sobre la organización lógica del software" [19]. Es por ello que este proceso de diseño tiene asociado la decisión del tipo arquitectura y los patrones de diseño que se deben utilizar en el sistema, así como la confección de distintos diagramas que favorezcan el trabajo en la fase de implementación. A continuación, se describe la arquitectura a utilizar en la solución propuesta.

2.4.1. Arquitectura de software

La arquitectura del software de un programa o sistema de cómputo es la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos. Es una representación que permite: analizar la efectividad del diseño para cumplir los requerimientos establecidos, considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y reducir los riesgos asociados con la construcción del software [21].

Arquitectura Clean

La Arquitectura Clean fue propuesta por el ingeniero y desarrollador Robert C. Martin. A través de esta arquitectura se busca crear sistemas que sean independientes de un marco de trabajo y que pueden ser probadas, independiente de la implementación. Esto busca crear una guía que permita a los desarrolladores poder generar aplicaciones que sean escalables y fáciles de mantener [22].

En la Arquitectura Clean los objetos encargados de la coordinación entre objetos de la capa de dominio y datos son los *Use Case* (Caso de Uso), también denominados *Interactors* (interactores). Estos se encuentran ubicados dentro de la capa de dominio y aquí si tienen una definición más concreta y clara. Por lo tanto, un caso de uso se necesita cuando se tiene que coordinar una acción requerida por el usuario [23].

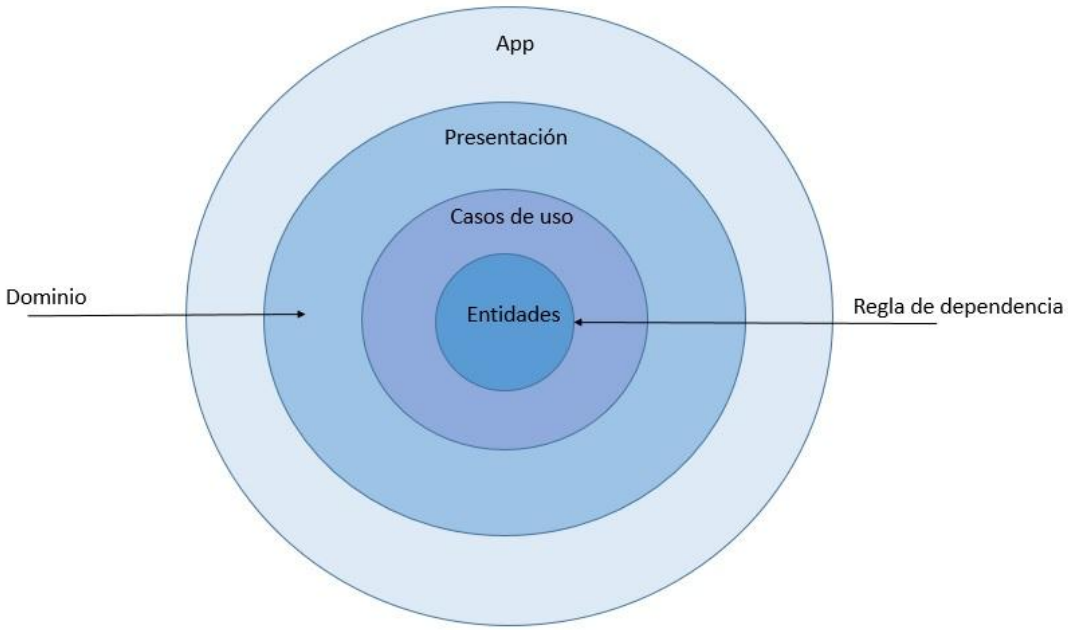


Figura 8: Estructura de la Arquitectura Clean

En la figura anterior se puede apreciar la capa Dominio, en esta se encuentran las Entidades, que encapsulan la lógica general, de alto nivel y los Casos de Uso, estos últimos se encargan de dirigir el flujo de datos desde y hacia las entidades. Por su parte la capa de Presentación y App,

representan el código más propenso al cambio y se mantienen alejadas para que su impacto sea menor.

Un punto muy importante en esta arquitectura son las reglas de dependencia. Estas definen que las dependencias a nivel de código fuente sólo pueden apuntar hacia dentro. En otras palabras, ninguna información que se encuentre en un círculo interior puede saber sobre otra información que se encuentre en un círculo exterior; el motivo es que los círculos exteriores son mecanismos mientras que los interiores son políticas [23].

Definición de las capas:

Capa dominio: Es la capa más interna de la arquitectura. No tendrá acceso directo al framework Android por lo que sólo tendrá código Java en su totalidad. El objetivo de esta capa es almacenar los casos de usos (interactors) y las entidades. Es la capa más importante de la aplicación pues es la que define el modelo de negocio [24].

Casos de uso: Son clases que dirigen el flujo de datos desde y hacia las entidades. Si las entidades encapsulaban la mayor parte de la lógica de negocio, serán los casos de uso los encargados de manejar la lógica propia de la aplicación [24].

Capa presentación: Es el anillo que envuelve el dominio y es la encargada de comunicar el dominio con la capa de la Interfaz de Usuario (UI) y no dependerá del framework Android. Además, contendrá los mappers (mapeo) que permitirán adaptar modelos de datos según la capa que los reciba. Ejemplo: cuando se comunique con la capa interna, mapeará un modelo de dato externo a una entidad, que es el único modelo de datos que entiende el modelo de negocio [24].

Capa 'Framework': Sería una capa dependiente del framework Android, que se divide en dos módulos dentro del proyecto:

- "Android": donde estará el código de las vistas: actividades, fragmentos, *custom* (personalizada), *views* (vistas) e inyección de dependencias o cualquier librería que dependa del framework.
- "data" (dato): donde estará la fuente de datos, de donde se obtendrá la información: base de datos, api, etc. [24].

2.4.2. Diagrama de clase del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y las interfaces que participan en el sistema con sus relaciones estructurales y de herencia. Los diagramas de este tipo contienen las definiciones de las entidades del software en vez de conceptos del mundo real y son utilizados durante el proceso de análisis y diseño de los

sistemas, donde se crea una vista lógica de la información que se maneja en el sistema, los componentes que se encargarán del funcionamiento y la relación entre uno y otro [29].

En la figura que se muestra a continuación se evidencia como quedarían organizadas las clases del diseño, teniendo como punto de partida la definición de la arquitectura descrita en el epígrafe anterior.

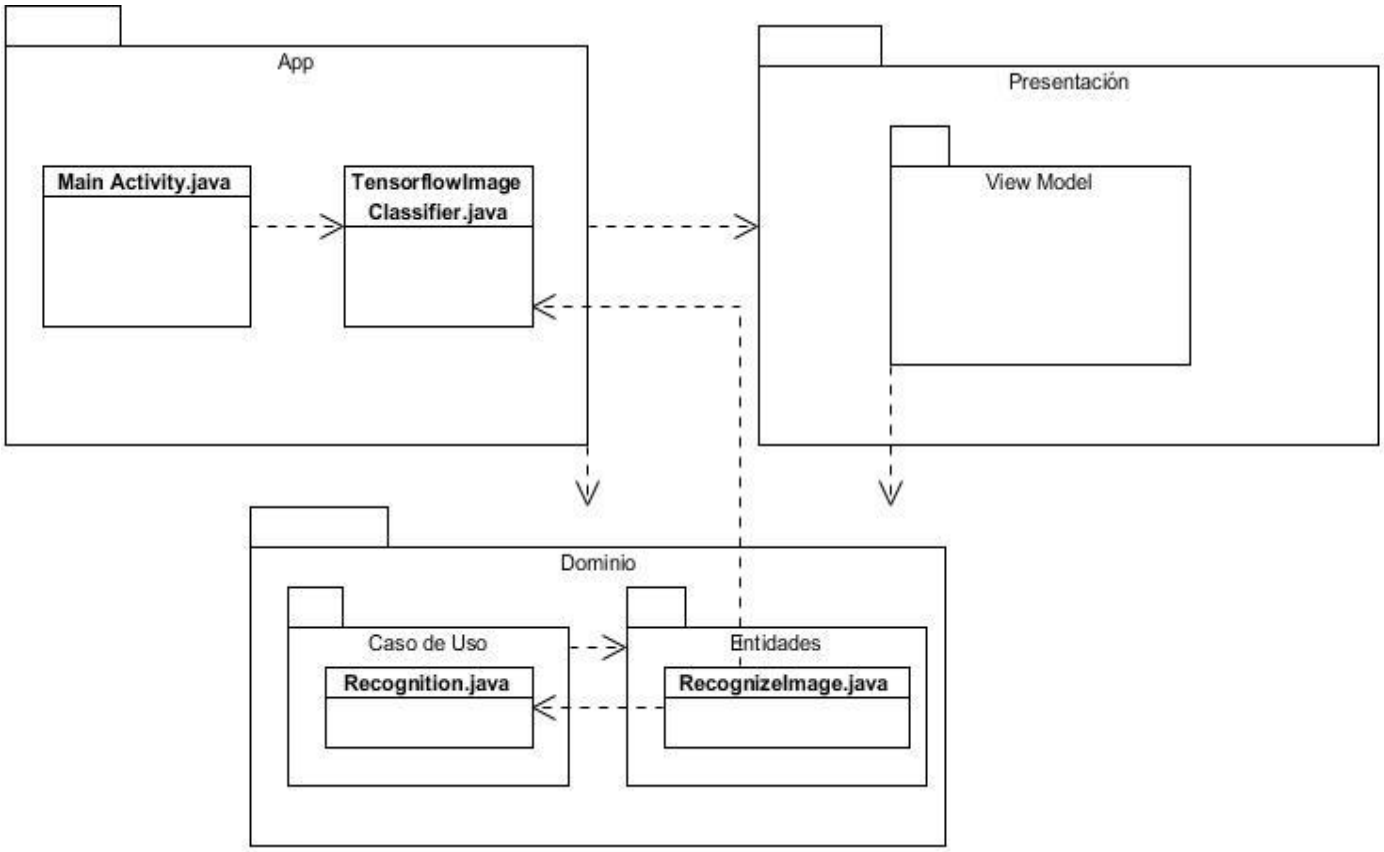


Figura 9: Diagrama clase de diseño. Elaboración propia.

2.4.3. Patrones de diseño

Los patrones de diseño describen una estructura que resuelve un problema de diseño en particular, dentro de un contexto específico y en medio de fuerzas que pueden tener un impacto en la manera en que se aplica y utiliza un determinado patrón. Estos son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces [30].

Se pueden agrupar en dos grandes grupos; los Patrones Generales de Software para Asignación de Responsabilidades (GRASP por sus siglas en ingles de General Responsibility Assignment Software Patterns) y los Patrones Banda de los Cuatro (GoF por sus siglas en ingles de Gang of Four), encargados de la inicialización, agrupación y comunicación de los objetos. A continuación,

se describen los patrones utilizados en el desarrollo de la solución propuesta como resultado de la presente investigación:

Patrones GRASP

Los Patrones para Asignar Responsabilidades (GRASP del inglés Responsability Assignment Patterns) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño del objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable [25]. En el diseño de la aplicación se emplearon los siguientes patrones de diseño GRASP:

Experto: mediante su uso, se asignan responsabilidades a la clase que cuenta con la información necesaria [26]. Este patrón se evidencia en la clase Main Activity, que a través del método initTensorFlowAndLoadModel se inicia el tensorflow, se crea el modelo y el clasificador.

Creador: permite crear objetos de una clase determinada [26]. Este patrón se evidencia en la clase Recognition, ya que es la encargada de crear nuevas instancias de la clase entidad RecognizeImage.

Polimorfismo: se emplea para asignar comportamientos distintos según el tipo de clase [26]. Este patrón se evidencia en la clase Main Activity que se extiende de la clase AppCompatActivity, de la cual hay métodos que redefinen su comportamiento como onCreate (iniciar actividad por primera vez), onResume (actividad comienza a interactuar), onPause (actividad no está en primer plano) y onDestroy (destruir actividad).

2.4.4. Diagrama de paquetes

Este diagrama es el encargado de representar las dependencias entre los paquetes, los cuales están compuestos por las clases o componentes de un subsistema en particular [40].

A continuación, se muestra el diagrama de paquetes que conforma la solución propuesta.

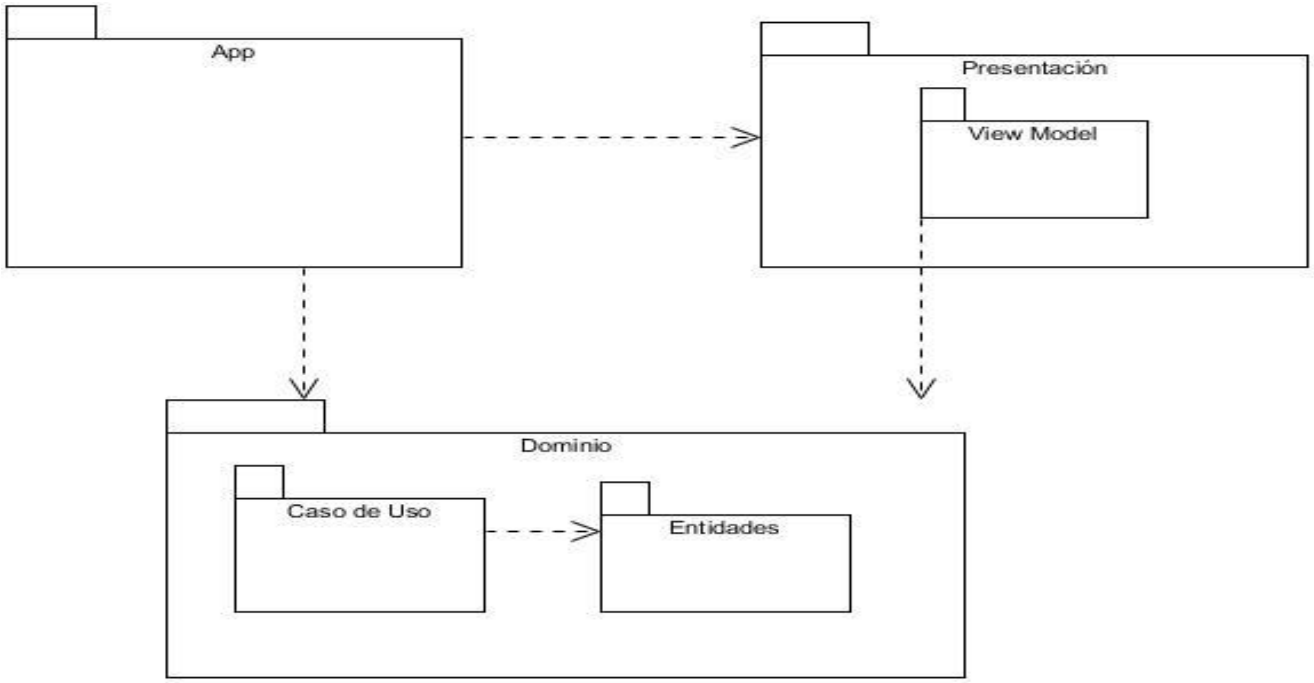


Figura 10: Diagrama de paquetes. Elaboración propia.

2.5. Conclusiones del capítulo

La descripción general de la solución propuesta favoreció adquirir una visión de los flujos de información y las actividades que engloban dicha propuesta. De igual forma, la captura de los requisitos de la aplicación propuesta permitió documentar toda la información relativa a los requisitos funcionales y no funcionales que se proponen para dar cumplimiento al objetivo trazado en la investigación. Por último, el establecimiento del diseño de la solución a través de la definición de la Arquitectura Clean, así como los patrones de diseño a utilizar y los diferentes diagramas generados, permitió sentar las bases para la implementación.

CAPÍTULO 3. Implementación y pruebas

En el presente capítulo se abordan los elementos que integran la etapa de implementación, donde se definen los estándares de codificación utilizados para el desarrollo de la solución propuesta, para garantizar el entendimiento, la legibilidad del código y los componentes necesarios para realizar la implementación. Además, se establece la estrategia de pruebas de software para verificar el correcto funcionamiento de la aplicación, así como los resultados obtenidos del proceso de verificación de la calidad.

3.1. Modelo de implementación

La implementación constituye una de las fases más importantes del desarrollo de software. En ella se toman como punto de partida los resultados obtenidos en el diseño, implementándose el sistema en términos de componentes como ficheros de código binario, código fuente, scripts y ejecutables. Su importancia reside en que se obtiene como consecuencia un sistema ejecutable, siendo esto uno de los principales objetivos en el desarrollo de software [31].

Para el caso de la presente investigación, el modelo de implementación se basa en el establecimiento de los estándares de codificación a emplear, así como la implementación según las descripciones de las historias de usuarios y, por último, la definición de los diagramas de componentes y despliegue a tener en cuenta en la solución propuesta.

3.1.1. Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código [32]. A continuación, se muestran los estándares de codificación definidos para la implementación de la solución propuesta:

- **Número de declaraciones por línea**: se debe declarar cada variable en una línea distinta, de esta forma cada variable se puede comentar por separado.
  Ejemplo:

```
private static final String MODEL_PATH = "mobilnet_float_v1_224.tflite";
private static final boolean QUANT = true;
private static final String LABEL_PATH = "labels.txt";
private static final int INPUT_SIZE = 224;
```

- **Espacio en blanco**: se utilizan líneas en blanco para separar segmentos de código que pueden corresponder a clases, funciones, declaraciones, implementaciones, comentarios y bloques. Se colocaron espacios para separar cada operador de su respectivo operando. Ejemplo:

```
private void initTensorFlowAndLoadModel() {
 executor.execute(new Runnable() {
 @Override
 public void run() {
 try {
 classifier = TensorFlowImageClassifier.create(
 getAssets(),

 MODEL_PATH,
 LABEL_PATH,
 INPUT_SIZE,
 QUANT);
 makeButtonVisible();
 } catch (final Exception e) {
 throw new RuntimeException("Error initializing TensorFlow!", e);
 }
 }
 });
}
```

- **Asignación de nombres:** cada tipo de elemento debe nombrarse con una serie de reglas determinadas.

  Clases e interfaces: la inicial en mayúscula ya sea simple o compuesta su nombre. Ejemplo:

```
public class MainActivity extends AppCompatActivity
public interface Classifier {
```

- **Métodos o funciones:** La declaración de funciones o métodos siempre comenzarán con letra inicial minúsculas. En caso de ser un nombre compuesto se regirá por la normativa. Ejemplo:

```
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 cameraView = findViewById(R.id.cameraView);
```

- **parámetros**: Los identificadores de los parámetros dentro de los constructores de las clases deben estar escritos con minúsculas separadas a un espacio cada uno. Ejemplo:

```
public Recognition(
 final String id, final String title, final Float confidence, final boolean
quant) {
 this.id = id;
 this.title = title;
 this.confidence = confidence;
 this.quant = quant;
```

3.1.2. Diagrama de componente

Un componente es una parte física de un sistema (módulo, base de datos, programa ejecutable). Se puede decir que un componente es la materialización de una o más clases, porque una abstracción con atributos y métodos pueden ser implementados en los componentes (29).

Un diagrama de componentes representa la separación de un sistema de software en componentes físicos (por ejemplo: archivos, módulos, paquetes, base de datos, código fuente, binario o ejecutable) y muestra las dependencias y organización existente entre estos componentes [33]. Teniendo como base la arquitectura definida en la presente investigación, a continuación, se muestra el diagrama de componentes de la solución propuesta:

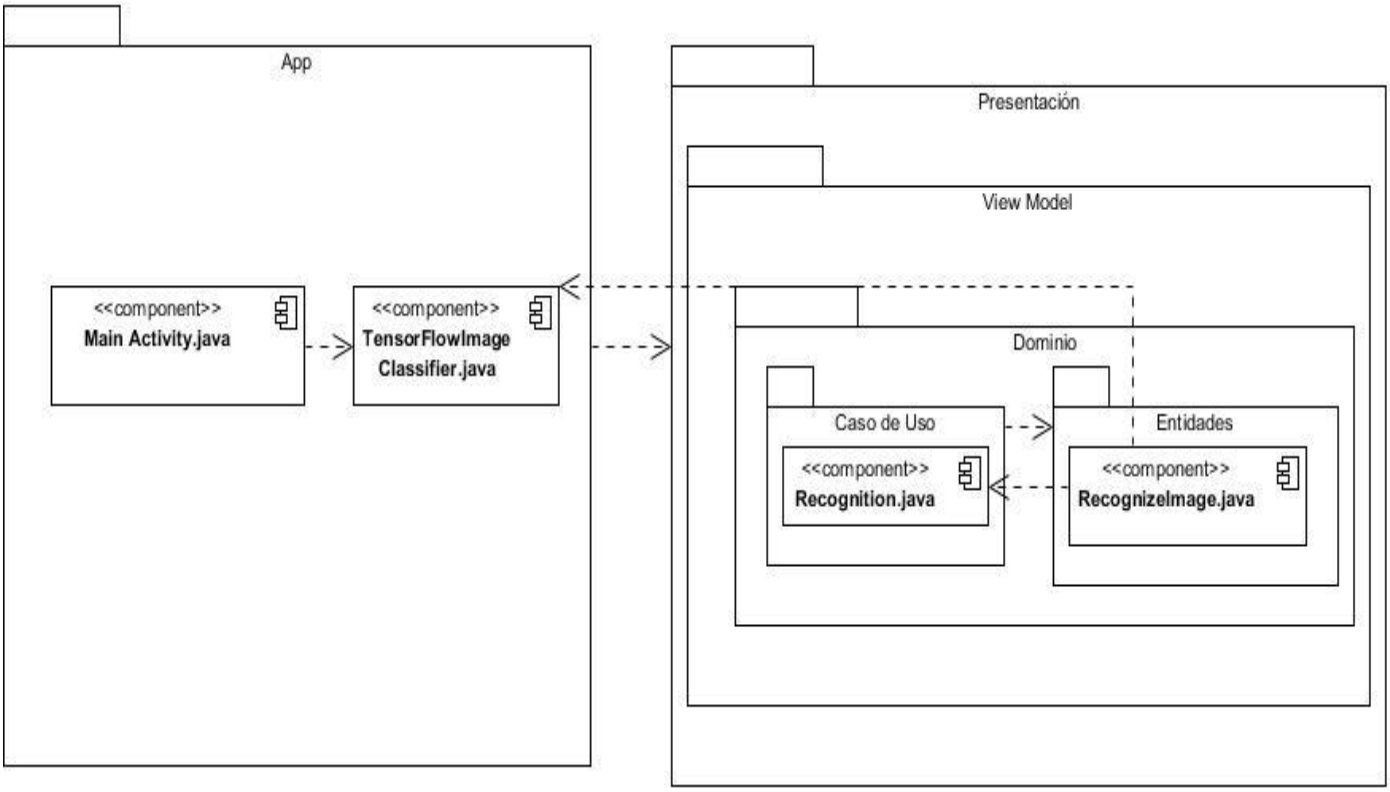


Figura 11: Diagrama de componentes. Elaboración propia.

3.1.3. Diagrama de despliegue

Los diagramas de despliegue muestran cómo los componentes de software se despliegan físicamente en los procesadores; así como el hardware y el software en el sistema, y el software de enlace para conectar los diferentes componentes en el sistema. En esencia, los diagramas de despliegue se pueden considerar como una forma de definir y documentar el entorno objetivo [39].

El diagrama que se presenta en la figura siguiente muestra la distribución física necesaria para el despliegue de la aplicación. Partiendo del principio que para utilizar esta aplicación en la detección de anomalías de una placa Arduino, se necesita de un dispositivo móvil con dicha aplicación instalada.

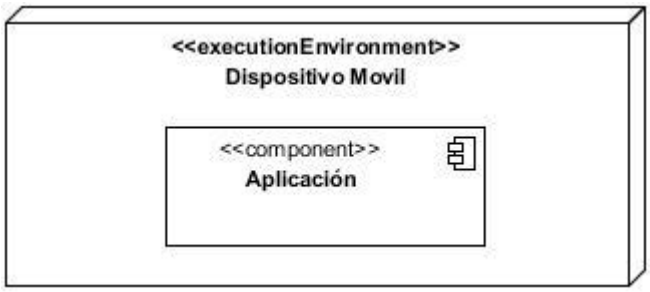


Figura 12: Diagrama de despliegue. Elaboración propia.

3.2. Pruebas del software

Las pruebas de software son un elemento crítico para la garantía de la calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación. El objetivo fundamental de las pruebas es descubrir diferentes clases de errores con la menor cantidad de tiempo y de esfuerzo. Aunque las pruebas no pueden asegurar la ausencia de defectos; sí pueden demostrar que existen defectos en el software [36].

3.2.1. Estrategia de prueba

Una estrategia de prueba del software integra los métodos de diseño de casos de prueba en una serie bien planteada de pasos que va a converger en la eficaz construcción del software. Dicha estrategia debe ser lo suficientemente flexible como para promover un enfoque personalizado, y al mismo tiempo lo adecuadamente rígido como para originar una planeación razonable y un seguimiento administrativo del avance del producto [37].

Pruebas unitarias

Se puede definir el concepto de prueba de unidad o unitaria como la actividad de probar el funcionamiento de un módulo software (código) con el objetivo de lograr su correcto funcionamiento [38]. Para aplicar las pruebas unitarias, el autor de la presente investigación define utilizar el método de caja blanca que a continuación se describe.

Método de caja blanca

La prueba de caja blanca, en ocasiones llamada prueba de caja de vidrio, es una filosofía de diseño de casos de prueba, que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que [38]:

1.1. Garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez.
1.2. Revisen todas las decisiones lógicas en sus lados verdadero y falso.
1.3. Ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas.
1.4. Revisen estructuras de datos internas para garantizar su validez. [38]

Para comprobar que cada sentencia de código se ejecuta al menos una vez, se realizaron pruebas al código de las funcionalidades más complejas desde el punto de vista de la programación en cada uno de los paquetes. Para esto se aplicó la técnica ruta básica, con el objetivo de comprobar que cada ruta se ejecute independiente de un componente o programa, obteniéndose una medida de la complejidad lógica del diseño.

Esta técnica de prueba debe ser utilizada para evaluar la efectividad de los métodos asociados a una clase, con el objetivo de asegurar que cada ruta independiente sea ejecutada por lo menos una vez en el sistema [38].

Pressman propone como estrategia para aplicar la ruta básica, realizar un análisis de la complejidad ciclomática de cada procedimiento que componen las clases del sistema; una vez concluido este paso se selecciona el método con valor de contener errores, además de que ofrece una medida del número de pruebas que deben diseñarse para validar la correcta implementación de una determinada función. Esta métrica se calcula sobre un grafo y se puede realizar mediante tres formas distintas:

1. V(G) = R.
2. V(G) = E – N + 2
3. V(G) = P + 1

Conociendo que:

- G: Grafo de flujo (grafo).
- R: El número de regiones contribuye a estimar el valor de la complejidad ciclomática.
- E: Número de aristas.
- V(G): Complejidad ciclomática.
- N: Número de nodos del grafo.
- P: Número de nodos predicados incluidos en el grafo.

Una vez calculada la complejidad ciclomática, el valor obtenido representa el límite superior de pruebas que deberán aplicarse [38].

Aplicación del método caja blanca

Como parte de la aplicación de la técnica ruta básica se presentan los resultados de cada uno de los pasos según Pressman, tomando como ejemplo el método recognizeImage(), siendo este uno

de los métodos que corresponde al requisito funcional “Clasificar imagen”, de prioridad Alta para el cliente.

```
public List<Recognition> recognizeImage(Bitmap bitmap) {
1   ByteBuffer byteBuffer = convertBitmapToByteBuffer(bitmap);
2   if(quant){
3     byte[][] result = new byte[1][labelList.size()];
       interpreter.run(byteBuffer, result);
      return getSortedResultByte(result);
4     } else {
5     float [][] result = new float[1][labelList.size()];
       interpreter.run(byteBuffer, result);
      return getSortedResultFloat(result);
    }
6   }
```

Figura 13: Método recognizeImage()

A continuación, se detallan los pasos que se realizaron para aplicar la técnica ruta básica:

1. **Confeccionar el grafo de flujo**: usando el código de la figura anterior, se realizó la representación del grafo de flujo, el cual describe un flujo de control lógico y está compuesto por los siguientes elementos [38]:
   - **Nodos de gráfica de flujo:** son círculos que representan una o más instrucciones procedimentales.
   - **Aristas o enlaces:** son flechas que representan el flujo de control y son análogas a las flechas del diagrama de flujo.
   - **Regiones:** son las áreas delimitadas por aristas y nodos.

En la siguiente figura se presenta el grafo de flujo obtenido:

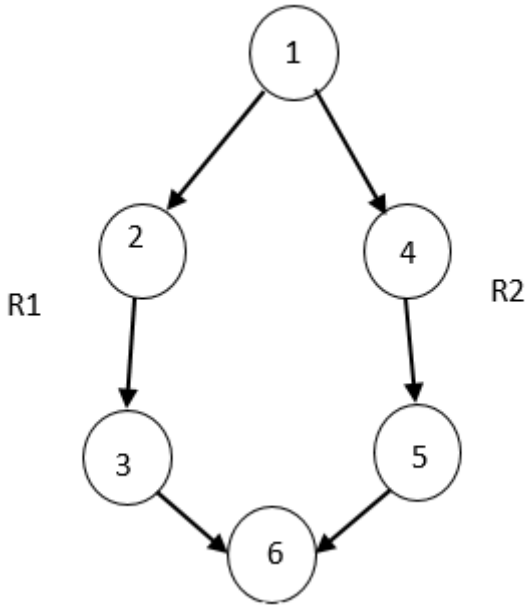


Figura 13: Grafo de flujo para el método recognizeImage()

2. **Calcular la complejidad ciclomática:** proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado define el número de rutas independientes del conjunto básico de un programa, y proporciona un límite superior para el número de pruebas que deben aplicarse para asegurar que todas las instrucciones se hayan ejecutado al menos una vez. La complejidad ciclomática se calcula mediante las tres formas siguientes:

   - **V (G) =** E – N + 2

     El gráfico de flujo tiene **2** regiones, representadas en la figura anterior como R1 y R2.

   - **V (G) =** E – N + 2

     V (G) = E – N + 2 = (6 – 6) + 2 = 2

   - **V (G) =** P + 1

     V (G) = 1 + 1 = 2

3. **Determinar un conjunto básico de rutas linealmente independientes:** el valor de V(G) indica el número de rutas linealmente independientes de la estructura de control del programa, por lo que a continuación se definen las 2 rutas independientes obtenidas:

   Ruta básica 1: 1-2-3-6

   Ruta básica 2: 1-4-5-6

4. **Obtención de casos de prueba:** cada ruta independiente es un caso de prueba a realizar, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino [38]. En este caso se obtuvieron 2 rutas independientes, que dan lugar a la confección de igual número de casos de pruebas. A continuación, se muestran los casos de pruebas para el método que se toma de ejemplo en el presente trabajo:

Tabla 5: Caso de prueba. Ruta independiente #1

Caso de prueba: Ruta independiente #1	
Descripción: en este método se retorna una lista de imágenes de tipo reconocimiento. En este caso se convierte un arreglo de imágenes bit a formato bytebuffer.	
Entrada	Imagen en mapa de bit (Bitmap)
Resultados esperados	Se retorna una lista de imágenes de tipo reconocimiento.
Condiciones	Imágenes bit.

Tabla 6: Caso de prueba. Ruta independiente #2

Caso de prueba: Ruta independiente #2	
Descripción: en este método se retorna una lista de imágenes de tipo reconocimiento. En este caso se convierte un arreglo de imágenes float a bytebuffer.	
Entrada	Imagen en mapa de bit(Bitmap)
Resultados esperados	Se retorna una lista de imágenes de tipo reconocimiento.
Condiciones	Imágenes float.

Una vez ejecutados todos los casos de pruebas obtenidos a través de la técnica de Ruta básica, se concluye que desde la primera iteración estos fueron probados satisfactoriamente, esto provocó que no se realizara una segunda iteración. Al concluir la prueba se demuestra que todas las funcionalidades internas del módulo se ejecutan satisfactoriamente, quedando libres de código repetido o innecesario.

Método de caja negra

Las pruebas de caja negra, también denominadas, pruebas de comportamiento, se concentran en los requisitos funcionales del software. Es decir, permiten al ingeniero de software derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa. La prueba de caja negra no es una opción frente a las técnicas de caja blanca. Es, en cambio, un enfoque complementario que tiene probabilidades de describir una clase diferente de errores de los que se descubrirán con los métodos de caja blanca [38].

Estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación [38]

La técnica de prueba que se utilizó es la técnica de Partición Equivalencia. Esta técnica divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El método se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de prueba que deben desarrollarse. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada [38].

Para aplicar esta técnica, se deben primeramente realizar el Diseños de casos prueba (DCP) con el objetivo de obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software. Un DCP es, en ingeniería del software, un conjunto de condiciones o variables bajo las cuales un analista determinará si una aplicación o una característica de éstos es parcial o completamente satisfactoria [38].

A continuación, se muestra el Diseño de caso de pruebas para el requisito “Acceder a la cámara del dispositivo móvil”:

Tabla 7: Escenarios del Diseño de Caso de Prueba

Escenario	Descripción	Ejemplo inicial	Respuesta del sistema	Flujo central
EC 1.1 Acceder a la cámara del dispositivo móvil con datos	El usuario debe acceder a la cámara del dispositivo	V True	Se accede a la cámara del dispositivo móvil.	1.1 El usuario pulsa el botón detectar objeto. 1.2 Se muestra la información del objeto

correctos.	móvil.			detectado, su identificación y clasificación
EC 1.2 Acceder a la cámara del dispositivo con datos incorrectos.	El usuario no debe acceder a la cámara del dispositivo móvil.	I False	La aplicación cancela el proceso.	1.1 El usuario pulsa el botón detectar objeto. 1.2 No se muestra la información del objeto detectado, su identificación y clasificación

Para un mejor entendimiento del DCP mostrado anteriormente, se muestra a continuación la descripción de las variables o de los campos que intervienen en dicho DCP.

Tabla 8: Descripción de la variable del Diseño de caso de prueba

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Ejemplo inicial	Objeto de la clase MAinActivity	No	Componente de la aplicación

Resultados de las pruebas de caja negra

Los problemas detectados en el período de pruebas se clasificaron en: No conformidades significativas (NCS) y en No conformidades no significativas (NCNS). A continuación, se describe en que consiste cada clasificación.

- **NCS:** son las no conformidades relacionadas a las funcionalidades de la aplicación. Son respuestas de la aplicación diferentes a lo descrito en las historias de usuario o validaciones incorrectas.
- **NNCS**: son las no conformidades relacionadas con errores ortográficos y al diseño de la propuesta de solución.

A la aplicación se le realizaron 3 iteraciones de pruebas. En la primera iteración se detectaron 2 NCS y 1 NCNS. Las mismas fueron resueltas satisfactoriamente en la misma iteración.

No Conformidades Significativas:

- La aplicación se detuvo al iniciarse.
- Al presionar el botón Detect Object (Detectar Objeto) la aplicación no captura la imagen.

No Conformidades No Significativas:

- La aplicación tiene faltas de ortografías en la descripción de uno de los botones.

En la segunda iteración se detectaron 1 NCS y 1 NCNS, siendo solucionadas de la misma forma que las anteriores.

No Conformidades Significativas:

- Al presionar el botón Toggle Camera (Alternar Cámara) no funciona.

No Conformidades No Significativas:

- La aplicación muestra incorrectamente los textos de la identificación y de la clasificación del objeto.

En la tercera iteración no se detectó ninguna NC, por lo que la aplicación fue considerada concluida, cumpliendo así con los requisitos funcionales establecidos con el cliente. A continuación, se muestra un gráfico donde se representa el total de no conformidades identificadas:

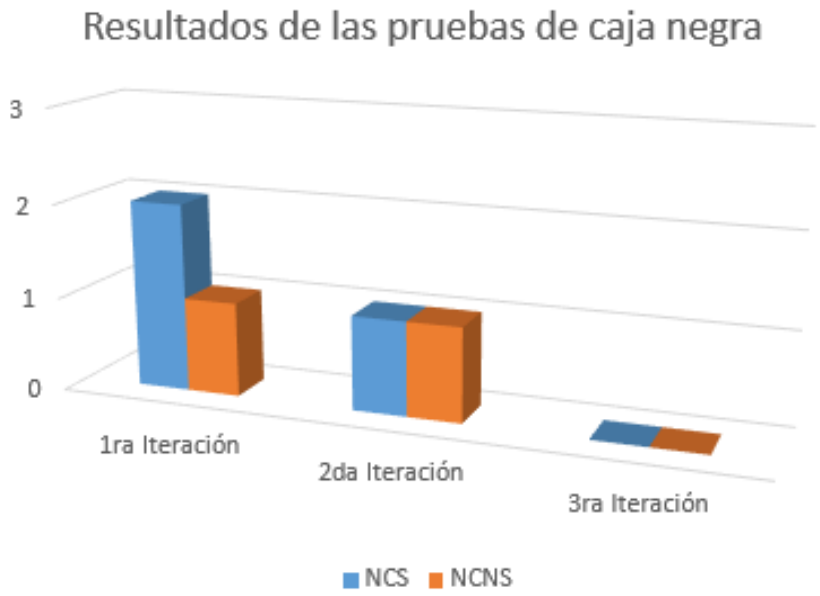


Figura 14: Resultados de las pruebas de caja negra

Para una mejor comprensión, en la Tabla 9 se presenta un resumen del total de las NC por cada Iteración:

Tabla 9: Resumen de las pruebas de caja negra

Iteraciones	Total de NC
1ra Iteración	3
2da Iteración	2
3ra Iteración	0

3.3. Conclusiones del capítulo

La elaboración del modelo de implementación, a través del establecimiento de los estándares de codificación, posibilitó una mayor comprensión del código, favoreciendo así la reutilización del mismo. De igual forma la realización de los diagramas de componentes y de despliegue permitió obtener una mejor visión de la distribución física y lógica de la solución propuesta. Por último, la definición de estrategia de pruebas, a través de las pruebas de caja blanca y caja negra permitió comprobar el correcto funcionamiento del sistema y el cumplimiento de los requisitos establecidos, detectándose un conjunto de no conformidades que fueron corregidas en su totalidad, garantizando así la calidad del software desarrollado.

Conclusiones generales

Luego de finalizada la investigación, se pudo arribar a las siguientes conclusiones:

- La elaboración del marco teórico de la investigación mediante el estudio y el análisis de los principales referentes teóricos en los que se sustenta la investigación, facilitó la adquisición de los conocimientos necesarios para desarrollar una aplicación con el fin de detectar anomalías en una placa Arduino a través del uso de la visión por computadora.
- El análisis de las soluciones existentes permitió determinar las características que constituyen la base para el diseño de las funcionalidades que se definen en la propuesta de solución.
- La elaboración de los artefactos propuestos por la metodología de desarrollo y la especificación de requisitos permitió un mejor entendimiento de la aplicación que se propone desarrollar, así como las características de la misma.
- La validación de los resultados alcanzados a través de las pruebas de software permitió comprobar de forma cuantitativa la calidad de los artefactos obtenidos durante el desarrollo de la solución propuesta.

Recomendaciones

Una vez cumplidos los objetivos de la investigación y teniendo en cuenta las experiencias obtenidas en la misma, se recomienda:

- El desarrollo de la aplicación para otros sistemas operativos compatibles con los dispositivos móviles.
- Desarrollar funcionalidades de realidad aumentada para ver la información en tiempo real .

Referencias bibliográficas

[1] «Visión artificial industrial como ejemplo evolutivo de la Industria 4.0», *[R]evolución artificial*, 25-nov-2019.

[2] «Visión artificial industrial: aplicaciones y sectores», *[R]evolución artificial*, 20-sep-2017.

[3] «Lección 1». [En línea]. Disponible en: http://datateca.unad.edu.co/contenidos/90169/90169_exe/leccin_1.html. [Accedido: 19-abr-2020].

[4] J. P. FERNÁNDEZ, «SISTEMA COLABORATIVO DE DETECCIÓN Y SEGUIMIENTO DE ANOMALÍAS MEDIANTE VISIÓN POR COMPUTADOR», 2016.

[5] «ENERO - ROBÓTICA CLAUDIA». [En línea]. Disponible en: https://sites.google.com/site/claudiarobot1719/home/enero. [Accedido: 19-abr-2020].

[6] «Redes Neuronales: una visión superficial - Fernando Sancho Caparrini». [En línea]. Disponible en: http://www.cs.us.es/~fsancho/?e=72. [Accedido: 19-abr-2020].

[7] «(DOC) Las Redes Neuronales Artificiales | Natan A Finol Bencomo - Academia.edu». [En línea]. Disponible en: https://www.academia.edu/7855311/Las_Redes_Neuronales_Artificiales. [Accedido: 19-abr-2020].

[8] «VISIÓN ARTIFICIAL - Buscar con Google». [En línea]. Disponible en: https://www.google.com/search?ei=oRRzXqiUBYOxggeLpLKoDg&q=+VISI%C3%93N+ARTIFICIAL&oq=+VISI%C3%93N+ARTIFICIAL&gs_l=psy-ab.3..0i7i30l10.104393.526256..526535...0.1..2.139.703.0j6.....3....1j2..gws-wiz.....0..0i71.wraRRbT7wVo&ved=0ahUKEwjo6Jvs96XoAhWDmOAKHQuSDOUQ4dUDCAs&uact=5. [Accedido: 19-abr-2020].

[9] «Build software better, together», *GitHub*. [En línea]. Disponible en: https://github.com. [Accedido: 19-abr-2020].

[10] «¿Qué es Python? - Curso de iniciación a la programación con Python en Raspberry Pi», *https://www.programoergosum.com*. [En línea]. Disponible en: https://www.programoergosum.com/cursos-online/raspberry-pi/244-iniciacion-a-python-en-raspberry-pi/que-es-python. [Accedido: 19-abr-2020].

[11] «El ecosistema de TensorFlow para programadores principiantes y expertos en Machine Learning: cursos, lenguajes y Edge Computing». [En línea]. Disponible en: https://www.genbeta.com/desarrollo/ecosistema-tensorflow-para-programadores-principiantes-expertos-machine-learning-cursos-lenguajes-edge-computing. [Accedido: 19-abr-2020].

[12] «Kotlin para Android, principales características - Código OnClick». [En línea]. Disponible en: https://codigoonclick.com/kotlin-para-android/. [Accedido: 19-abr-2020].

[13] «Comparativo general entre Kotlin y Java: JetBrains. - Buscar con Google». [En línea]. Disponible en:

https://www.google.com/search?source=hp&ei=dCBzXqv7CKeIggetgqagBg&q=Comparativo+general+entre+Kotlin+y+Java%3A+JetBrains+decide+crear+un+nuevo+lenguaje+que+se+ejecute+sobre+la+m%C3%A1quina+virtual+de+Java+anteriormente+nombrada+y+decide+crearlo+con+el+objetivo+de+que+compile+tan+deprisa+como+el+propio+Java+en+su+m%C3%A1quina+virtual.+&oq=Comparativo+general+entre+Kotlin+y+Java%3A+JetBrains+decide+crear+un+nuevo+lenguaje+que+se+ejecute+sobre+la+m%C3%A1quina+virtual+de+Java+anteriormente+nombrada+y+decide+crearlo+con+el+objetivo+de+que+compile+tan+deprisa+como+el+propio+Java+en+su+m%C3%A1quina+virtual.+&gs_l=psy-ab.12...2015.2015..2847...0.0..0.0.0.....0....2j1..gws-wiz.oTY6agWCTZI&ved=0ahUKEwirhdGPg6boAhUnhOAKHS2BCWQQ4dUDCAo. [Accedido: 19-abr-2020].

[14] «¿Qué es Tensorflow? | OpenWebinars». [En línea]. Disponible en: https://openwebinars.net/blog/que-es-tensorflow/. [Accedido: 19-abr-2020].

[15] «Keras backends», *keras.io*.

[16] «Android Studio - Wikiwand». [En línea]. Disponible en: https://www.wikiwand.com/es/Android_Studio. [Accedido: 19-abr-2020].

[17] «Metodología de Desarrollo de Software Variación de AUP para la UCI - EcuRed». [En línea]. Disponible en: https://www.ecured.cu/Metodolog%C3%ADa_de_Desarrollo_de_Software_Variaci%C3%B3n_de_AUP_para_la_UCI. [Accedido: 19-abr-2020].

[18] F. J. García-Peñalvo y A. García-Holgado, "Análisis orientado a objetos," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: https://goo.gl/LZ3K6z. doi: 10.5281/zenodo.1181820. (pp. 17-47)

[19] «Ingenieria-de-Software-Ian-Somerville-9-edicion-español[1] | Jamie Patrick - Academia.edu». [En línea]. Disponible en: https://www.academia.edu/15366832/Ingenieria-de-Software-Ian-Somerville-9-edicion-espa%C3%B1ol_1_. [Accedido: 19-abr-2020].

[20] R. Jeffries, A. Anderson, y C. Hendrickson, *Extreme Programming Installed*. Addison-Wesley, 2001.

[21] PRESSMAN., R.S., 2005. Ingeniería del software. Un enfoque práctico. 6ta Edición. S.l.: s.n.

[22] «La Arquitectura Clean, - Buscar con Google». [En línea]. Disponible en: https://www.google.com/search?ei=OypzXqLADsfj_Ab5rp7oDg&q=La+Arquitectura+Clean+++fue+propuesta+por+el+ingeniero+y+desarrollador+Robert+C.+Martin.+A+trav%C3%A9s+de+esta+arquitectura+se+busca+crear+sistemas+que+sean+independientes+de+un+framework%2C+que+pueden+ser+probadas%2C+&oq=La+Arquitectura+Clean+++fue+propuesta+por+el+ingeniero+y+desarrollador+Robert+C.+Martin.+A+trav%C3%A9s+de+esta+arquitectura+se+busca+crear+sistema

s+que+sean+independientes+de+un+framework%2C+que+pueden+ser+probadas%2C+&gs_l=psy-ab.12...308970.333640..334814...0.0..0.276.681.2j2j1.....2....1j2..gws-wiz.....0..0i7i30j0i67j0i30j0j0i131.W9ds8S_tjxs&ved=0ahUKEwjiyZm5jKboAhXHMd8KHXmXB-0Q4dUDCAs. [Accedido: 19-abr-2020].

[23] «¿Por qué utilizo clean Architecture?» [En línea]. Disponible en: http://xurxodev.com/por-que-utilizo-clean-architecture-en-mis-proyectos/. [Accedido: 19-abr-2020].

[24] «Vísteme con Clean Architecture que tengo prisas», *Chema Pramos*, 25-feb-2016. .

[25] SHIH, R., 2014. AutoCAD 2015 Tutorial - Second Level: 3D Modeling. S.l.: SDC Publications. ISBN 978-1-58503-865-7.

[26] LARMAN, C., 2003. UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado. S.l.: Pearson Educación. ISBN 978-84-205-3438-1.

[27] Enríquez Ruiz, José Luis, Farías Palacín, Elías y Flores Flores, Eder. 2017. Metodología de desarrollo de software. Rectorado, Universidad Católica Los Ángeles - Chimbote. Chimbote - Perú : s.n., 2017. pág. 39.

[28] RODRÍGUEZ SÁNCHEZ, T., 2015. Metodología de desarrollo para la actividad productiva de la UCI. S.l.: s.n.

[29] Larman, Craig. 2016. UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. 3ra . s.l. : Prentice-Hall, 2016.

[30] Rojas, MC Juan Carlos Olivares. 2014. Patrones de Diseño. 2014.

[31] Colombia.com. Colombia.com. [En línea] 2017.

[32] Alfonso, Damián Pérez. 2008. Normas y estándares de codificación. Universidad de las Ciencias Informáticas. Normas y estándares de Codificación del ERP. 2008.

[33] Somerville, Ian. 2005. Ingeniería de Software. 7ma. 2005.

[34] Sistema-Master-Magazine. 2018. Sistema-Master-Magazine. [En línea] 2018. https://sistemas.com/herramienta.php#ixzz2qwSpRx1E

[35] S. PRESSMAN, R., 2003. Ingeniería de Software. Un enfoque práctico. 5ta Edición. S.l.: s.n.

[36] Información, Oficina Nacional de Estadística e Información. Anuario Estadístico de Cuba 2015. Cuba : s.n., 2015.

[37] ANDROIDPIT. [En línea] [Citado el: 22 de 2 de 2017.] http://www.androidpit.es/sdk-android.

[38] Pressman, Roger S. 2010. Ingeniería de Software. Un enfoque práctico. Septima . Mexico DF : McGraw-Hill INTERAMERICA EDITORES, 2010.

[39] I. Sommerville. Software Engineering. Ed. por Pearson Education Limited. E7. Pearson Education Limited, 2006. isbn: 84-7829-074-5 (vid. págs. 30, 33, 42, 45).
[40] http://www.geocitrus.com/adotsaha/index.html

http://www.alyuda.com/neural-networks-software.htm

Asuncion A, Newman DJ. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.

[41] Belén Fos-Guarinos, Ángel Alberich-Bayarri, Ignacio Bosch-Roig, Amadeo Ten-Esteve, Luis Martí-Bonmatí

Herrera A. (2015). Detección de texto utilizando redes neuronales convolucionales. Barcelona: Universitat Politècnica de Catalunya.

Krizhevsky A., Sutskever I., Hinton G.E. (2012). Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 25 (NIPS 2012).

Anexos

Anexo 1. Historias de usuario de la solución propuesta

Número: 1	**Requisito**: Entrenar modelo para clasificar e identificar imágenes de arduinos
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 12 días
Riesgo en Desarrollo: Que se presente errores en el código.	**Tiempo Real:** 13 días
Descripción: Para realizar este modelo se debe crear una red neuronal, la cual se entrena con ciertas imágenes de placas Arduinos en buen estados y otros con defectos para que sepa identificar y clasificar las anomalias existentes en ellas **Objetivo**: Permite clasificar e identificar la imagen de la placa Arduino.	
Observaciones:	

Número: 2	**Requisito**: Exportar el modelo a tflite
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 15 días
Riesgo en Desarrollo: Que no se pueda exportar el modelo.	**Tiempo Real:** 16 días
Descripción: Una vez creado la neurona y de haber creado el modelo en el formato h5 se convierte el formato a tflite. **Objetivo**: Exportar el modelo a tfile convirtiendo de formato h5 a tflite.	
Observaciones:	

Número: 3	**Requisito**: Tener acceso a la camara del dispositivo

Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: -	**Tiempo Real:** 6 días
Descripción: Se le da acceso a la cámara del dispositivo para que la aplicación pueda realizar capturas de pantalla desde su interfaz. **Objetivo**: Permite el acceso a la cámara del dispositivo.	
Observaciones:	

Número: 5	**Requisito**: Tener acceso al almacenamiento local
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 12 días
Riesgo en Desarrollo: -	**Tiempo Real:** 13 días
Descripción: Se le da acceso al dispositivo para que guarde las imágenes capturadas en el almacenamiento local del dispositivo. **Objetivo**: Tener acceso al almacenamiento local para que se guarden las imágenes capturadas.	
Observaciones:	

Número: 6	**Requisito**: Informar al usuario de los fallos ocurridos y debe tener en cuenta la recuperación frente a fallos
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 días
Riesgo en Desarrollo: -	**Tiempo Real:** 9 días

Descripción: Despues de capturar la imagen con la camara del dispositivo, se muestra la anomália ocurrida. **Objetivo**: Informar al usuario de la anomalía detectada.
Observaciones:

Número: 8	**Requisito**: Visualizar los elementos y las ayudas mediante la pantalla táctil del móvil
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 días
Riesgo en Desarrollo: -	**Tiempo Real:** 9 días
Descripción: Se visualiza los elementos de la aplicación con las que el usuario interactua. **Objetivo**: Visualizar los elementos y las ayudas mediante la pantalla táctil del móvil	
Observaciones: :	

Número: 9	**Requisito:** Proporcionar la interacción con el usuario por medio de interfaces táctiles
Programador: Raudel Hernández González	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: -	**Tiempo Real:** 6 días
Descripción: El usuario puede interactuar con los elementos de la interfaz a través de la pantalla táctil. **Objetivo**: Proporcionar la interacción con el usuario por medio de interfaces táctiles	
Observaciones:	