UNIVERSIDAD NACIONAL DEL LITORAL

Facultad de Ingeniería y Ciencias Hídricas

PROYECTO FINAL DE CARRERA

INGENIERÍA INFORMÁTICA

DESARROLLO DE APLICACIÓN MÓVIL PARA RECONOCIMIENTO Y VALORACIÓN DE CERVEZAS ARTESANALES

Alumno: Manuel Paiva.

Director: Dr. Enzo Ferrante.

Co-Director: Dr. Enrique Marcelo Albornoz.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
M. Paiva, E. Ferrante & E. M. Albornoz; "Desarrollo de aplicación móvil para reconocimiento y valoración de cervezas artesanales (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2021.

Resumen

El rubro de la cerveza artesanal ha crecido notablemente durante los últimos años. Al ser un mercado nuevo existe una notable falta de información en los consumidores, lo que se evidencia al momento de elegir un producto.

En este proyecto se propuso el desarrollo de una aplicación para dispositivos móviles que brinde a los usuarios información acerca de qué tener en cuenta a la hora de catar una cerveza, información sobre distintas cervezas, estilos y productores. Una de las particularidades de dicha aplicación es que incorpora una herramienta para poder fotografiar una botella o lata de cerveza artesanal y, por medio de técnicas de procesamiento de imágenes, visión computacional e inteligencia computacional, detecta automáticamente la marca y tipo de cerveza, brindando información sobre la misma con el fin de ayudar al consumidor a elegir con criterio propio un producto.

Para esto se han estudiado tres métodos diferentes para obtener puntos característicos de una imagen y se ha seleccionado el que mejor se ajusta a los requerimientos del proyecto. También se estudió la posibilidad de incorporar una red neuronal convolucional con el propósito de detectar automáticamente la imagen de una botella, y descartarla en caso de que la misma no se encuentre presente en la fotografía. El prototipo de aplicación diseñado e implementado es compatible con el sistema operativo Android. Por último, la herramienta desarrollada en este proyecto final de carrera pretende informar de manera gratuita a consumidores de cerveza artesanal.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
M. Paiva, E. Ferrante & E. M. Albornoz; "Desarrollo de aplicación móvil para reconocimiento y valoración de cervezas artesanales (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2021.

Agradecimientos

Agradezco en primer lugar a mis padres, hermanos y a mi novia Carla por haberme acompañado y apoyado sin esperar nada a cambio en este camino que elegí.

A mis amigos, aquellos que están siempre.

A mis directores Enzo y Marcelo por su apoyo incondicional y por su predisposición para trabajar conmigo en este proyecto.

A los integrantes de la cátedra Proyecto Final de Carrera por su atención y acompañamiento en esta etapa.

Y por último a la Universidad Pública y sus docentes por haberme brindado educación.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
M. Paiva, E. Ferrante & E. M. Albornoz; "Desarrollo de aplicación móvil para reconocimiento y valoración de cervezas artesanales (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2021.

Índice general

Índice de figuras

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
M. Paiva, E. Ferrante & E. M. Albornoz; "Desarrollo de aplicación móvil para reconocimiento y valoración de cervezas artesanales (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2021.

1. Introducción

En este capítulo se presenta una introducción al proyecto final de carrera, incluyendo la justificación, objetivos y alcances del proyecto.

1.1 Justificación

En nuestro país el mercado de la cerveza artesanal como tal, comenzó con cervecería El Bolsón a mediados de la década del 80. Si bien siempre existieron cerveceros, este fue el primer emprendimiento a gran escala. Unos años más tarde se inaugura Patagonia y luego nace Antares, en Mar del Plata. Estos fueron los pioneros que comenzaron con la "revolución cervecera". Hoy en día son cada vez más los productores que se dedican al rubro, el cual ha crecido notablemente especialmente durante los últimos cinco años.

El mercado de la cerveza artesanal ha crecido de manera inesperada y la demanda es cada vez mayor. También surge una sobre oferta en el mercado, no solo de la mano de muchos productores nuevos, sino que cada vez son más los estilos y recetas nuevas que se ofrecen [1]. Al ser un mercado tan nuevo y con una oferta en crecimiento constante, existe una notable falta de información en los consumidores, lo que se evidencia en el desconocimiento al momento de elegir o evaluar una cerveza. Muchas personas no distinguen entre estilos (ni siquiera aquellos muy diferentes) ni conocen sus características básicas. Tampoco detectan cualidades notables en cervezas que sufren algún tipo de contaminación en sus aromas y sabores (lo que representa un problema bastante común y se encuentra presente en muchas ocasiones).

A diferencia del caso de la cerveza, las personas aficionadas al vino suelen contar con mayor información para elegir un producto. Un claro ejemplo es la aplicación móvil llamada VIVINO [2], que es una comunidad online centrada en la venta y recomendación de esta bebida alcohólica. Esta aplicación cuenta con un increíble catálogo y los usuarios pueden comprar, recomendar y calificar productos fácilmente. Además, brinda información de cada vino permitiendo a las personas aprender sobre la bebida, no solo para elegir un vino sino también para generar sus propias opiniones. Otra aplicación similar es Delectable [3], de las mismas características a la antes mencionada.

Para los amantes de la cerveza existen aplicaciones como Untappd [4] que permiten buscar bares y cervecerías en distintos lugares del mundo así como también brindan información acerca de distintas cervezas. Sin embargo, no se cuenta en la actualidad con una aplicación que permita escanear una cerveza y obtener su puntuación e información,

y como son cada vez más los productores y estilos de cerveza que aparecen en el mercado, será de gran interés tener una aplicación que oriente al usuario a elegir una de las tantas cervezas disponibles.

El presente trabajo realizará un aporte a los consumidores de cerveza, cuyo número viene creciendo considerablemente en la actualidad. A través del desarrollo de un software, se compartirá información relevante acerca de la bebida, con el objetivo de que las personas tengan a su alcance el conocimiento suficiente para elegir una cerveza con criterio propio.

Este software será una aplicación móvil que tendrá información sobre estilos, productores y

propiedades organolépticas, que son todas aquellas descripciones de las características físicas que tiene la cerveza en general, según las pueden percibir los sentidos, como por ejemplo su sabor, textura, aromas, color o temperatura. Además, los usuarios podrán puntuar la cerveza en una escala de uno a cinco y también podrán filtrar una búsqueda por puntuación. El sistema contará con un punto de acceso simple a la información basado en la búsqueda de la marca o estilo por medio de texto. Además permitirá escanear y reconocer la cerveza por medio de la cámara del teléfono móvil para luego mostrar información acerca de la misma. En este sentido, informar a los consumidores, ayudarlos a elegir qué cerveza comprar y brindarle información básica para poder catarla, es primordial para la aplicación.

La variedad disponible de cervezas es cada vez más amplia. El impacto de este proyecto será a favor de los usuarios, quienes contarán con la información para poder reconocer diferentes estilos, tendrán acceso inmediato a sus características y podrán filtrar cervezas por puntuación, lo que les será de gran ayuda para poder elegir un producto . Finalmente contarán con información para aprender sobre sabores y aromas no deseados, lo cual les será de gran ayuda para catar una cerveza.

Por otra parte, los productores también serán beneficiados, no solo por formar parte y exponer sus cervezas, sino también porque los usuarios podrán puntuar los productos dándoles la posibilidad de conseguir el máximo puntaje posible en el ranking.

La información utilizada será provista por el BJCP (Beer Judge Certification Program) [5], un reconocido proyecto que entre sus objetivos tiene fomentar el conocimiento, la comprensión y apreciación de diversos estilos de cerveza.

Motiva el desarrollo del proyecto compartir con las personas información relevante acerca de la cerveza artesanal para que puedan elegir un producto con mayor conocimiento.

1.2 Objetivos

Objetivo general

Desarrollar una aplicación para el reconocimiento y valoración de cervezas artesanales con el fin de orientar a las personas en la elección y compra de un producto.

Objetivos específicos

En particular, se persiguen los siguientes objetivos específicos:

- Evaluar y seleccionar los métodos basados en redes neuronales que mejor se adapten al proyecto para detectar botellas.
- Evaluar y seleccionar los métodos correspondientes al procesamiento digital de imágenes y visión computacional que mejor se adapten al proyecto para la detección y comparación de cervezas.
- Desarrollar un sistema de escaneo y reconocimiento de imágenes para que los usuarios tomen una fotografía de una botella y puedan obtener descripción organoléptica, valores principales del estilo de la cerveza escaneada y ver el puntaje

que le otorgan los usuarios a este producto en particular.

- Desarrollar un módulo de búsqueda por texto con el fin de poder encontrar una cerveza en particular o un estilo sin la necesidad de tomar una fotografía.
- Presentar la información de características no deseadas en cervezas con fin educativo para los consumidores.

1.3 Alcance

A continuación se presentan los alcances del proyecto.

Alcances no funcionales

- El software desarrollado será una aplicación compatible con dispositivos móviles cuyo sistema operativo sea Android:

  El prototipo que se desarrollará para este proyecto no será programado para múltiples sistemas operativos porque excede a los tiempos disponibles del proyecto. La aplicación se diseñará y programará para el sistema operativo Android. Esto se debe a que la mayoría de los usuarios de teléfonos móviles en Argentina lo utilizan

- El backend de este prototipo será escrito en el lenguaje Python, pues dicho lenguaje cuenta con reconocidas librerías de visión computacional y procesamiento de imágenes que tienen soportes de larga duración (LTS o long time support en inglés) , las cuales serán requeridas a la hora de analizar una captura.
- La aplicación debe funcionar para teléfonos móviles de gama media.
- La aplicación deberá pedir permiso para el uso de la cámara del dispositivo y deben funcionar todas sus características para teléfonos de gama media.

Alcances funcionales

La aplicación comprende los siguientes módulos:

- Reconocimiento de una cerveza a través de una fotografía tomada por el usuario:

  Se espera que el usuario pueda acceder a una pantalla dentro de la aplicación en la que se le pedirá acceso a la cámara y en caso de aceptar le permita tomar una fotografía de una botella de cerveza, apretar un botón de escaneo y la aplicación le devolverá información y puntuación acerca de dicha cerveza.

- Búsqueda por texto de cerveza:

  El software deberá contar con un buscador de cervezas. El usuario debe poder buscar una cerveza ingresando estilo, marca o productor y obtener información acerca de la búsqueda.

- Lista de características no deseadas en cervezas artesanales.

  Durante el proceso de cocción y almacenado de una cerveza se pueden generar

sabores y aromas no deseados ya sea por errores en la cocción, mala temperatura en el almacenamiento o herramientas no sanitizadas, entre otros. Es importante que la aplicación brinde al usuario información acerca de estas características no deseadas con el fin de informar al usuario de la existencia de estos defectos. Se espera que en la pantalla principal del prototipo aparezca esta información.

Para las dos primeras funcionalidades la aplicación devuelve la siguiente información:

- Descripción organoléptica del estilo de cerveza.
- Valores principales estándares del estilo de cerveza: amargor y alcohol.
- Puntuación por parte de los usuarios.
- Lista de estilos de cervezas relacionados.

Para la primera funcionalidad, si la base de datos no reconoce la cerveza se mostrará al usuario un mensaje diciendo que no se reconoce la cerveza y brindando la posibilidad de completar los siguientes campos:

- Nombre de la cerveza
- Estilo de cerveza.
- Valor de amargor y alcohol.

Esta información será enviada al administrador para ser verificada y luego se incorporará la cerveza a la base de datos.

2. Análisis del Estado del Arte

En este capítulo se expone en primera instancia la diferencia entre procesamiento de imágenes y visión computacional, diferenciación que permitirá al lector una mejor comprensión sobre el tema. Luego, se discuten diferentes técnicas existentes para la detección de botellas en una imagen y clasificación de objetos.

2.1 Introducción

En el capítulo 1 se han mencionado aplicaciones para la detección automática de botellas de vino las cuales son desarrollos privados y no se tiene acceso a las tecnologías que utilizan. Lo que sí se deduce es que utilizan una imagen como punto de entrada para el reconocimiento de la bebida. Además, se ha mencionado una aplicación que ayuda a los usuarios a encontrar bares y buscar cervezas existentes en su base de datos. Para esta aplicación tampoco se tiene acceso a las tecnologías con las que ha sido desarrollada.

Para este proyecto se opta por desarrollar una aplicación que tenga la funcionalidad de poder escanear una botella, por lo tanto en esta sección se expondrán las diferentes técnicas que se utilizarán, tanto para la detección de botellas, como para la extracción y comparación de características entre dos imágenes. En este apartado se evaluarán distintos métodos (y combinaciones entre ellos), con el fin de seleccionar los algoritmos que se implementarán en el proyecto. La aplicación que se desarrollará será utilizada en teléfonos móviles, por esta razón, el rendimiento en términos de costo computacional será crucial a la hora de seleccionar una combinación de métodos, sin dejar de lado la eficiencia de los resultados.

A continuación se presentan las diferencias entre los conceptos de visión computacional y procesamiento digital de imágenes. Esto será de gran importancia para una mejor comprensión del documento.

La visión es la percepción y comprensión de la realidad física a través de la vista. La visión computacional es el estudio de estos procesos. Su fin es entenderlos y construir máquinas con capacidades similares. Un área muy ligada a la visión computacional es el procesamiento de imágenes. Aunque ambos campos tienen mucho en común, el objetivo final es diferente. La finalidad del procesamiento de imágenes es mejorar la calidad de las mismas para su posterior utilización o interpretación, por ejemplo:

- Restaurar problemas por movimiento o desenfoque,
- Realzar ciertas propiedades como color, contraste, etc.

En la figura 2.1 se ilustra el enfoque del procesamiento de imágenes. Su función principal es presentar la misma imágen resaltando e ignorando ciertas características. Representa una transformación de una imagen en otra imagen [6].

Figura 2.1: Esquema general del procesamiento de imágenes.

El objetivo de la visión computacional, sin embargo, es extraer características de una imagen para su descripción e interpretación. Por ejemplo:

- determinar la localización y tipo de objetos presentes en una imagen,
- comparar objetos presentes en imágenes, etc.

En la figura 2.2 se ilustra el enfoque de visión computacional donde la imagen de entrada es procesada para extraer los atributos obteniendo como salida una descripción de la imagen analizada [6].

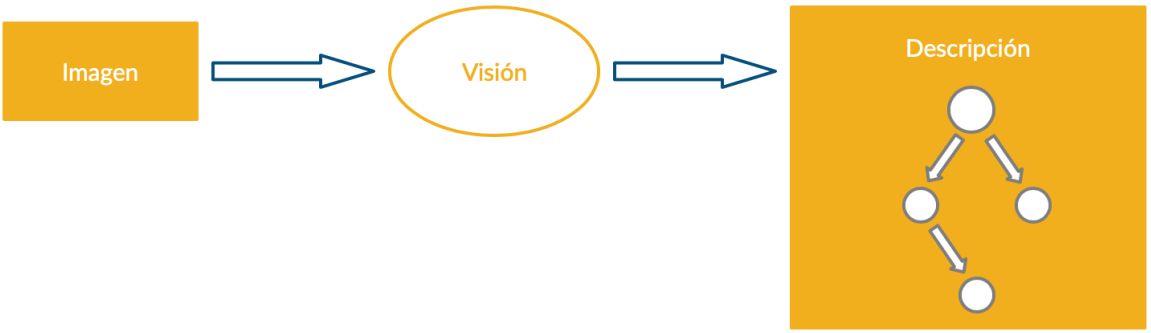


Figura 2.2: Esquema general de visión computacional.

Para mayor claridad se exponen a continuación dos ejemplos. En la figura 2.3 se muestra una operación correspondiente al procesamiento de imágenes, donde el objetivo es realzar el contraste de la imagen de entrada. La imagen de salida es esencialmente la misma pero ha sufrido una ecualización del histograma.

Figura 2.3: Realce de contraste: (a) imagen de entrada, (b) compresión del rango dinámico a través de una ecualización del histograma [6].

La figura 2.4 ilustra un caso de visión computacional; nótese que en esta imagen, el software ha detectado la patente del automóvil. La salida de este sistema de visión es la misma imagen con la placa enmarcada y los dígitos alfanuméricos que la componen.

Figura 2.4:Visión computacional aplicada al reconocimiento de matrículas [6]

Hasta aquí se han presentado algunas ideas acerca de la visión computacional y el

procesamiento digital de imágenes. Este trabajo comprende técnicas pertenecientes a ambas disciplinas, las cuales serán utilizadas junto a algoritmos de aprendizaje automático para llevar a cabo la detección de botellas. En particular, se explorará el uso de modelos de aprendizaje automático basados en redes neuronales profundas.

El objetivo final será detectar una botella presente en una filmación en tiempo real, tomar una captura de la misma, y luego extraer sus características. Dichas características serán comparadas con otras correspondientes a imágenes guardadas en una base de datos. Todo esto se realizará con el fin de encontrar similitudes entre las imágenes.

Las tareas a cumplir serán:

- Desde la cámara de un dispositivo móvil se tomará una captura de la botella utilizando un algoritmo de detección de objetos. Para una mejor captura, se utilizarán herramientas del procesamiento de imágenes con el fin de garantizar la calidad de la misma.
- Se utilizarán métodos de la visión computacional para comparar la captura obtenida anteriormente, con imágenes de cervezas preexistentes en una base de datos.

A continuación se detallan distintos métodos de visión por computadora que fueron explorados. También se detallarán dos redes neuronales, las cuales se utilizarán en conjunto para la detección de objetos en una imagen.

2.2 Image Matching

La detección de características y la coincidencia de imágenes han sido dos problemas importantes en visión artificial, y sus aplicaciones continúan creciendo en varios campos. Una técnica de detección de características ideal debería ser robusta ante las transformaciones de una imagen, como la iluminación, el ruido y las transformaciones afines. El ruido de una imagen se considera como un fenómeno aleatorio que contamina una imagen. Es típicamente modelado como un proceso aditivo en el procesamiento digital de imágenes. Una imagen contaminada puede ser descrita como g(x,y) = f(x,y) + n(x,y), donde g(x,y) es la imagen contaminada por algún tipo de ruido, f(x,y) es la imagen sin ruido y n(x,y) el ruido. Además, x e y es la posición de cada pixel de la imagen.

Scale Invariant Feature Transform (SIFT) [7] es un detector de características que ha demostrado ser muy eficiente en aplicaciones de reconocimiento de objetos aunque requiere una gran complejidad computacional, lo cual es inconveniente para aplicaciones en tiempo real. Existen variantes y extensiones de SIFT que han mejorado su complejidad de cómputo tales como Speeded-Up Robust Features (SURF) [8], que es una aproximación de SIFT, pero funciona más rápido y no reduce la calidad de los puntos detectados. Tanto SIFT como SURF se basan en un descriptor y un detector. Oriented FAST and Rotated BRIEF (ORB) es otra alternativa para SIFT y SURF que requiere menor capacidad de cómputo con un rendimiento de coincidencia similar. A continuación se describen estos tres métodos.

SIFT

La detección de características o puntos de interés hace referencia en visión computacional a la tarea de localizar puntos relevantes en cuanto a la información de su entorno, como por

ejemplo, la existencia de bordes en una imagen. Existen algoritmos abocados a esta tarea, los cuales funcionan bien ante rotaciones de la misma. Sin embargo, estos algoritmos no suelen ser invariantes frente al escalado. SIFT es un algoritmo de detección de características robusto ante transformaciones afines. Esto significa que se pueden detectar puntos característicos aún cuando la imagen ha sufrido cualquier tipo de transformación afín. A continuación se describe su funcionamiento.

Hay principalmente cinco pasos involucrados en el algoritmo SIFT. Los veremos uno por uno.

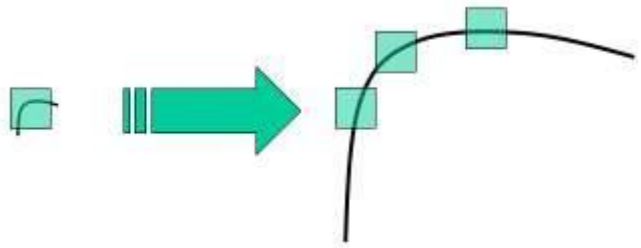


Figura 2 .5: Aumento de escala. Una esquina (izquierda) puede no detectarse como esquina si la imagen se escala (derecha).

Detección de extremo a escala de espacio

Como se puede notar en la figura 2.5, no se puede utilizar la misma ventana para detectar puntos claves ante distintas escalas de una imagen. Se necesitan diferentes ventanas para detectar esquinas en diferentes escalas. Para esto, SIFT utiliza filtros de espacio-escala. Generalmente estos filtros utilizan el Laplaciano del Gaussiano (LoG) que encuentra varios valores σ, los cuales actúan como parámetros de escala. Es decir, para una imagen pequeña el núcleo gaussiano tendrá un valor pequeño de σ y devolverá un valor alto donde exista una esquina, mientras que para una imagen más grande el filtro tendrá un valor mayor de σ y también devolverá un valor alto para una esquina. De esta manera se pueden encontrar los máximos locales a través de este tipo de filtros, los cuales retornan valores de (x,y,σ) que significa que hay un punto clave en las coordenadas x-y de la imagen a escala σ.

Lo que sucede con este concepto es que es costoso en término computacional, por lo que SIFT utiliza la diferencia de Gaussianos que es una aproximación al LoG. La Diferencia de Gaussianos (DoG) se calcula como la diferencia del desenfoque gaussiano de una imagen para escalas σ diferentes.

Luego de calcular la DoG, se comparan vecindades de diferentes escalas para encontrar máximos locales. Por ejemplo, para una cierta escala se compara un pixel de la imagen con sus 8 vecinos, y para la escala más pequeña anterior y más grande posterior se compara el mismo píxel con sus 8 vecinos. Si para las tres escalas el píxel en cuestión es un máximo local es porque ese punto es un punto de interés.

Localización de puntos claves

Una vez que se encuentran los puntos de interés, SIFT examina las intensidades en dichos puntos para obtener mejor precisión en los resultados. Si la intensidad en estos máximos

locales es inferior a un umbral, se descartan.

Otro problema es que DoG tiene una respuesta alta para los bordes, por lo que los bordes también deben descartarse. Para esto se detecta la curvatura, y nuevamente, si esta curvatura supera un cierto umbral, se descarta.

Asignación de la orientación

En esta etapa SIFT asigna una orientación a cada uno de los puntos claves obtenidos para lograr la invarianza respecto de la rotación de la imagen. Para esto, se evalúan las vecindades de los puntos calculando la magnitud y dirección del gradiente en cada punto correspondiente a la vecindad. Luego se crea un histograma de dichas orientaciones y se lo evalúa. Se toma el pico máximo del histograma para encontrar un punto de interés. También se considera cualquier pico que esté por encima del 80% del máximo para encontrar otros puntos de interés pero con diferente orientación.

Descriptor de puntos claves

En esta etapa se evalúa un vecindario de 16 x 16 para cada punto clave. Luego se divide en 16 subregiones de 4 pixeles y para cada subregión se crea un histograma de orientación. Los resultados se concatenan en un vector el cual es llamado descriptor del punto.

Correspondencia de puntos claves entre imágenes

Para comparar puntos claves entre dos imágenes se realiza una búsqueda del punto más próximo en el vector descriptor de cada punto. Lo que sucede es que en ocasiones el segundo punto más próximo puede estar muy cerca a causa del ruido presente en la imagen. Por esta razón se calcula la distancia a los primeros dos puntos más cercanos. Si para el segundo más cercano la distancia es menor que un umbral, entonces éste se descarta. Esto ayuda a eliminar coincidencias falsas entre puntos de dos imágenes.

SURF

El detector de características SURF [8] tiene como objetivo mejorar el rendimiento en términos de costo computacional respecto del SIFT a costa de la precisión. SURF aproxima el DoG (diferencia de gaussianos) con filtros de caja. En lugar de promediar de forma gaussiana la imagen, se usan cuadrados para la aproximación ya que la convolución con el cuadrado es mucho más rápida si se usa la imagen integral. También esto se puede hacer en paralelo para diferentes escalas.

Detección de puntos de interés y correspondencia

A diferencia del SIFT, este método utiliza un detector BLOB que se basa en la matriz de Hesse para encontrar los puntos de interés. Para la asignación de orientación, utiliza respuestas wavelet tanto en dirección horizontal como vertical mediante la aplicación de pesos gaussianos adecuados.

Para la descripción de la característica, SURF también utiliza las respuestas wavelet. Se selecciona un vecindario alrededor del punto clave y se divide en subregiones y luego, para cada subregión, las respuestas wavelet se toman y se representan para obtener el descriptor de característica SURF.

El signo de Laplaciano que ya se calcula en la detección se utiliza para los puntos de interés subyacentes. El signo del laplaciano distingue las manchas brillantes sobre fondos oscuros del caso inverso. En caso de coincidencia, las características se comparan solo si tienen el mismo tipo de contraste (basado en el signo) que permite una coincidencia más rápida.

ORB

Oriented FAST and Rotated BRIEF[9] propone un reemplazo computacionalmente eficiente de SIFT que tenga un rendimiento de coincidencia similar [10], se vea menos afectado por el ruido de la imagen y sea capaz de usarse en tiempo real por un dispositivo de poca memoria. Este descriptor funciona tan bien como SIFT en estas tareas (y mejor que SURF), mientras que es casi dos órdenes de magnitud más rápido y es un método utilizado para visión computacional en tiempo real en teléfonos inteligentes [11].

ORB se basa en el conocido detector de puntos clave FAST y el descriptor BREVE. Ambas técnicas son atractivas debido a su buen rendimiento y bajo costo. Las principales contribuciones de ORB son las siguientes:

- La adición de un componente de orientación rápido y preciso a FAST
- El cálculo eficiente de funciones BREVES orientadas
- Análisis de varianza y correlación de características BREVES orientadas
- Un método de aprendizaje para descifrar las características BREVES bajo invariancia rotacional, lo que lleva a un mejor rendimiento en aplicaciones vecinas más cercanas

Hasta aquí, se han presentado tres algoritmos conocidos para la detección de características en una imagen. A continuación se describen arquitecturas de redes neuronales utilizadas por dispositivos de bajos recursos computacionales, las cuales serán de interés para el reconocimiento de botellas. Finalmente se presentará una conclusión y se justificará la elección de los algoritmos elegidos.

2.3 Redes neuronales profundas

Una red neuronal profunda (DNN) [12] es una red neuronal artificial (ANN) con varias capas ocultas entre las capas de entrada y salida. El propósito principal de una red neuronal es recibir un conjunto de entradas, realizar cálculos y retornar una salida para resolver problemas del mundo real como por ejemplo, la clasificación de objetos.

Las redes neuronales convolucionales [13] se han vuelto omnipresentes en el área de visión computacional. La tendencia general ha sido crear redes más profundas y complejas centradas en lograr una mejor precisión para el reconocimiento de objetos. Sin embargo, estos avances no necesariamente hacen que las redes sean más eficientes con respecto al tamaño y la velocidad. En muchas aplicaciones del mundo real, como la robótica o la realidad aumentada, las tareas de reconocimiento deben llevarse a cabo de manera oportuna en una plataforma **computacionalmente limitada.**

En este trabajo se seleccionará y se utilizará un modelo ya entrenado para detección de objetos. Se comparan a continuación dos arquitecturas de redes pre seleccionadas teniendo

en cuenta el uso de los recursos computacionales.

MobileNet

Es una arquitectura de red neuronal presentada por Google Inc. desarrollada para su utilización en aplicaciones de dispositivos móviles [14]. Esta arquitectura utiliza convoluciones separables en profundidad con el fin de construir una red de gran rendimiento. La convolución separable en profundidad se utiliza para reducir el tamaño y la complejidad de este modelo [14]. MobileNet es particularmente útil para aplicaciones de visión móviles e integradas. Sus características son:

- Tamaño de modelo más pequeño que otros: menos número de parámetros.
- Menor complejidad: menos multiplicaciones y adiciones (adiciones múltiples).

MobileNet puede alcanzar alrededor de 25 fps con una gran cantidad de objetos detectados al mismo tiempo [15]. A continuación se expone una tabla comparativa entre la convolución estándar frente a la convolución separable en profundidad para el conjunto de datos de ImageNet:

Modelo	Tasa de acierto de ImageNet	Millón de adiciones y multiplicaciones	Millón de parámetros
Conv. MobileNet	71.7%	4866	29.3
MobileNet (Convolución separable en profundidad)	70.6%	569	4.2

Tabla 2.6: Convolución estándar vs Convolución separable en profundidad (conjunto de datos ImageNet) [14].

Esto muestra que la convolución separable en profundidad pierde rendimiento pero es mejor en cuanto a velocidad en los cálculos.

Por otro lado, se utilizan dos parámetros de configuración para lograr una combinación entre precisión y velocidad.

- El multiplicador de ancho α es un parámetro de entrada para el algoritmo que se introduce para controlar el ancho de entrada de una capa. Este parámetro varía entre 0 y 1. Cuanto más cerca esté de 1 mayor precisión tendrá el algoritmo y menor velocidad. Por el contrario, cuanto menor sea, aumenta la velocidad y disminuye la precisión en los resultados.
- El multiplicador de resolución ρ es un parámetro que se introduce para controlar la resolución de la imagen de entrada de la red. ρ varía entre 0 y 1 y las resoluciones serán 224, 192, 160 y 128.

Los valores de la tabla 1.3.2 corresponden a α = ρ = 1.

Hasta aquí se ha discutido brevemente el modelo MobileNet. Si bien en este proyecto solo se utilizará la red ya entrenada, es de gran interés presentar información acerca de su desempeño. Utilizando MobileNet se logra un rendimiento similar en comparación con otros métodos, pero con una red mucho más pequeña y más rápida. Este modelo se ve favorecido por la convolución separable en profundidad.

Single Shot Detector (SSD)

El modelo SSD se basa en una red convolucional de avance que produce una colección de cuadros delimitadores (bounding boxes) y puntajes para la presencia de instancias de clase de objeto en esos cuadros, seguido de un paso de supresión no máxima para producir las detecciones finales [16]. Las primeras capas de la red se basan en una arquitectura estándar utilizada para la clasificación de imágenes de alta calidad (truncada antes de cualquier capa de clasificación), la cual es llamada red base. Luego se agrega una estructura auxiliar a la red para producir detecciones de interés.

Además del desempeño del SSD, se debe conocer otras opciones que afectan el rendimiento de una aplicación y que deben ser tenidas en cuenta:

- Resoluciones de imagen de entrada.
- El número de predicciones.
- Conjunto de datos de entrenamiento en la red utilizada.
- Uso de imágenes de múltiples escalas en entrenamiento o pruebas (con recorte).
- Plataforma de software de aprendizaje profundo utilizada.
- Configuraciones de entrenamiento que incluyen tamaño de lote, cambio de tamaño de imagen de entrada, tasa de aprendizaje y disminución de la tasa de aprendizaje.

Tiempos de inferencia

En el trabajo [17], publicado por investigadores de Google Research, se estudia la compensación entre velocidad y precisión para SSD frente a otros modelos conocidos que son F-R-CNN [18] y R-FCN [19], los cuales no se tendrán en cuenta por su bajo rendimiento en términos de velocidad.

Estos modelos se implementaron en TensorFlow utilizando el conjunto de datos MS COCO [20] (dataset de uso público utilizado para detección, segmentación y clasificación de objetos) para el entrenamiento. También presenta MobileNet, que logra una alta precisión con una complejidad mucho menor.

La pregunta más importante aquí no es qué detector presenta mayor precisión. La verdadera pregunta es qué detector y qué configuraciones brindan el mejor equilibrio de velocidad y precisión que este proyecto necesita.

A continuación se muestra la comparación de la precisión frente a la compensación de velocidad (tiempo medido en milisegundos). En esta imagen se distinguen las redes

mediante colores y sus distintas configuraciones de meta arquitectura con formas diferentes en los marcadores.

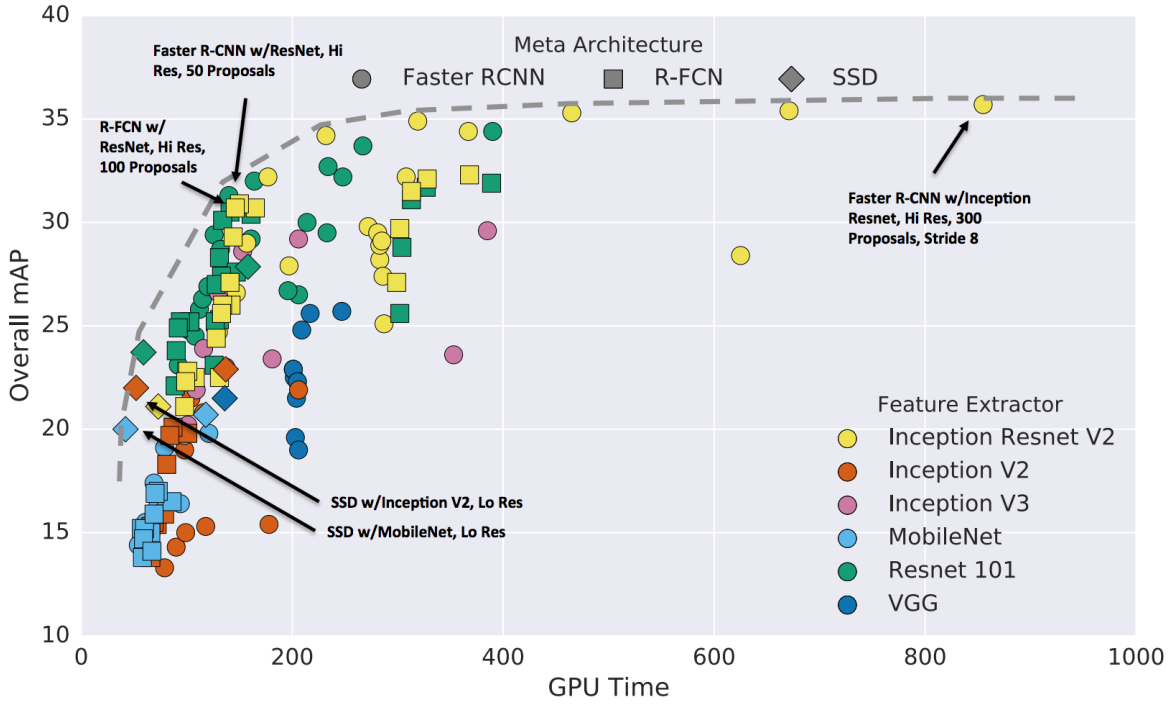


Figura 2 .7: Precisión de redes neuronales respecto del tiempo de cómputo [17].

En general, Faster R-CNN es más preciso, mientras que R-FCN y SSD son más rápidos. SSD en MobileNet tiene el mAP (mean average precision) más alto entre los modelos destinados al procesamiento en tiempo real.

Para objetos grandes, SSD funciona bien incluso con un extractor simple. SSD puede incluso igualar las precisiones de otros detectores utilizando un mejor extractor. Pero SSD no logra buena precisión en objetos pequeños en comparación con otros métodos. El presente trabajo considerará una botella como un objeto grande ya que las capturas en tiempo real se tomarán exclusivamente con dicho objeto en primer plano.

Hallazgos clave del documento de Google Research:

- Los modelos R-FCN y SSD son más rápidos en promedio, pero no pueden vencer al Faster R-CNN en precisión si la velocidad no es una preocupación.
- Faster R-CNN requiere al menos 100 ms por imagen.
- La resolución de la imagen de entrada afecta significativamente la precisión. Reducir el tamaño de la imagen a la mitad en ancho y alto reduce la precisión en un 15.88% en promedio, pero también reduce el tiempo de inferencia en un 27.4% en promedio.
- El procesamiento posterior incluye la supresión no máxima (que solo se ejecuta en la CPU) y ocupa la mayor parte del tiempo de ejecución para los modelos más

rápidos, aproximadamente 40 ms, lo que limita la velocidad a 25 FPS.

- Si mAP se calcula solo con una única IoU, use mAP@IoU=0.75.

Respecto de la velocidad:

- SSD con MobileNet proporciona la mejor compensación de precisión dentro de los detectores más rápidos.
- El SSD es rápido pero funciona peor para objetos pequeños en comparación con otros.
- Para objetos grandes, SSD puede superar a Faster R-CNN y Faster R-FCN en precisión con extractores más rápidos.

2.4 Elección del modelo

En este documento se presentaron dos arquitecturas de redes neuronales profundas: MobileNet (red base) y Single Shot Detector (red de detección). Se podría elegir una red base con su última capa totalmente conectada, sin embargo otra alternativa es reemplazar la última capa por algún tipo de red de detección con el fin de optimizar el tiempo de procesamiento.

El presente trabajo selecciona una combinación de SSD con MobileNet, un modelo de detección de objetos optimizado especialmente para visión computacional en tiempo real. Para comenzar, el usuario toma una fotografía de la cerveza y ésta se envía automáticamente al servidor donde será procesada. Una vez la imagen está en el servidor se evaluará si es borrosa o no. Si no lo es se utilizará la combinación de MobileNet y SSD para detectar si en la misma existe una botella y en caso de no detectarse una botella se le informará al usuario para que envíe otra fotografía. Finalmente, si la combinación de las redes antes mencionadas detectan una botella y la imagen no es borrosa, entonces se extraerán los puntos característicos de la imagen y se realizará una comparación de los mismos con los puntos característicos correspondientes a otras cervezas almacenadas en la base de datos. Para tal comparación se utilizará el método de macheo de imágenes ORB, el cual es elegido por su rapidez de procesamiento y por su capacidad de comparación ante imágenes que porten algún tipo de ruido.

3. Diseño y desarrollo de la aplicación

En la sección 1.3 se presentó MobileNet [14], SSD [16] y su combinación como método para la detección de botellas y captura automática de la imagen en tiempo real. Luego de considerar distintas opciones de diseño, se optó por que sea el consumidor quien tome una fotografía de manera manual y que ésta sea procesada ciento por ciento en el servidor quitando tareas de procesamiento al dispositivo móvil y mejorando la performance de la aplicación. Esta decisión permite además ejecutar el servidor en dos modos diferentes: 1) con la opción de detección de botellas, donde se advierte al usuario en el caso de que no exista una botella en la imagen, y se indica que debe tomar nuevamente la fotografía; 2) sin detección de botellas, lo que permite al consumidor consultar por cervezas enlatadas ya que la red implementada no está entrenada para la detección de latas. La opción 2 fue implementada dado que no se cuenta con una base de datos etiquetada con latas de cerveza para poder re-entrenar la red. Bajo esta variación en el diseño del software la imagen tomada por el usuario será procesada íntegramente en el servidor, permitiendo así ejecutar el servidor en dos modos de uso. Un modo utilizará la combinación de MobileNet con SSD para asegurarse que el usuario está tomando una imagen que contiene una botella. Por el contrario, el otro modo no aplica esta combinación de redes y permite que el consumidor pueda consultar tanto cervezas embotelladas como enlatadas.

En esta sección se detallan las tecnologías seleccionadas para el desarrollo, la arquitectura propuesta, el patrón de diseño elegido para la implementación y los esquemas (wireframes) de las vistas de la aplicación. Finalmente se detalla el desarrollo del prototipo.

La aplicación será desarrollada para el sistema operativo Android y contará con un backend escrito en el lenguaje de programación Python. A continuación se detalla la justificación de su elección.

3.1 Android como sistema operativo

Android es un sistema operativo móvil desarrollado por Google [21], basado en el Kernel de Linux y otros software de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas, automóviles (Android Auto) y televisores (Android TV). Para este proyecto se elige el sistema operativo antes mencionado porque según StatCounter, un reconocido sitio web de análisis de tráfico web, en Argentina el 91.68% de las personas lo utiliza en sus dispositivos [22]. Además se utilizará Jetpack que es un conjunto de bibliotecas, herramientas y guías para ayudar a los desarrolladores a escribir aplicaciones de alta calidad de forma más sencilla.

3.2 Python como lenguaje de backend

Se trata de un lenguaje de programación multiparadigma [23], ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Python además es compatible con OpenCV [24], una biblioteca libre de visión artificial originalmente desarrollada por Intel, la cual será utilizada en el desarrollo.

Se ha evaluado también desarrollar el backend del prototipo en NodeJS [25], que es un lenguaje basado en Javascript [26] muy utilizado para aplicaciones en tiempo real. Si bien NodeJS cuenta con un gestor de paquetes y librerías entre los cuales tiene disponibles librerías de procesamiento de imágenes y visión computacional, OpenCV es una librería nueva para NodeJS y es limitada en sus funcionalidades. Además el soporte no está garantizado para dicho lenguaje.

3.3 MongoDB como base de datos

Luego de un análisis respecto de qué tipo de base de datos se utilizaría en el proyecto se concluye que una base de datos no relacional será la más apropiada para el desarrollo del prototipo. Cabe aclarar que se desarrollará en este proyecto la primera versión de esta aplicación, así que será importante elegir una base de datos flexible para poder ejecutar cambios en las siguientes versiones de manera sencilla. Cuando se define una tabla en una base de datos relacional todas las ocurrencias existentes de dicha tabla deberán tener la misma estructura y si se quiere agregar, modificar o eliminar un campo se deberá modificar para todas las ocurrencias existentes en la tabla generando posiblemente redundancia en la base de datos. Si bien se puede normalizar la base de datos como posible solución a cualquier cambio, siempre existe redundancias en la realidad. Para esto, entre otros motivos, se han inventado las bases de datos no relacionales, las cuales son más flexibles a la hora de ejecutar cambios una vez la base de datos ya está cargada y esto permitirá ir modificando el prototipo en el futuro sin necesidad de estar preocupandose por los datos de cervezas ya existentes. Es decir que se podrá agregar un nuevo campo a las cervezas sin modificar los datos preexistentes. Dentro de las bases de datos no relacionales se han estudiado varias posibilidades. Todas las bases de datos no relacionales hoy en día presentan excelentes características, todas son escalables, flexibles, fáciles de mantener, pero se ha optado por MongoDB [27] porque una de sus características principales es que es de código abierto y tiene una gran comunidad a la cual se puede recurrir.

3.4 Arquitectura de la aplicación

La arquitectura de una aplicación es una elección importante para garantizar el correcto funcionamiento de la misma. Jetpack recomienda a los programadores una arquitectura basada en la separación de problemas, es decir, en asegurarse de que cada parte del sistema tenga solamente lo indispensable para funcionar de manera óptima. Esta arquitectura está diseñada para considerar la mayoría de las situaciones posibles y los flujos de trabajo. El siguiente diagrama muestra cómo deberían interactuar los módulos entre sí, una vez diseñada la aplicación.

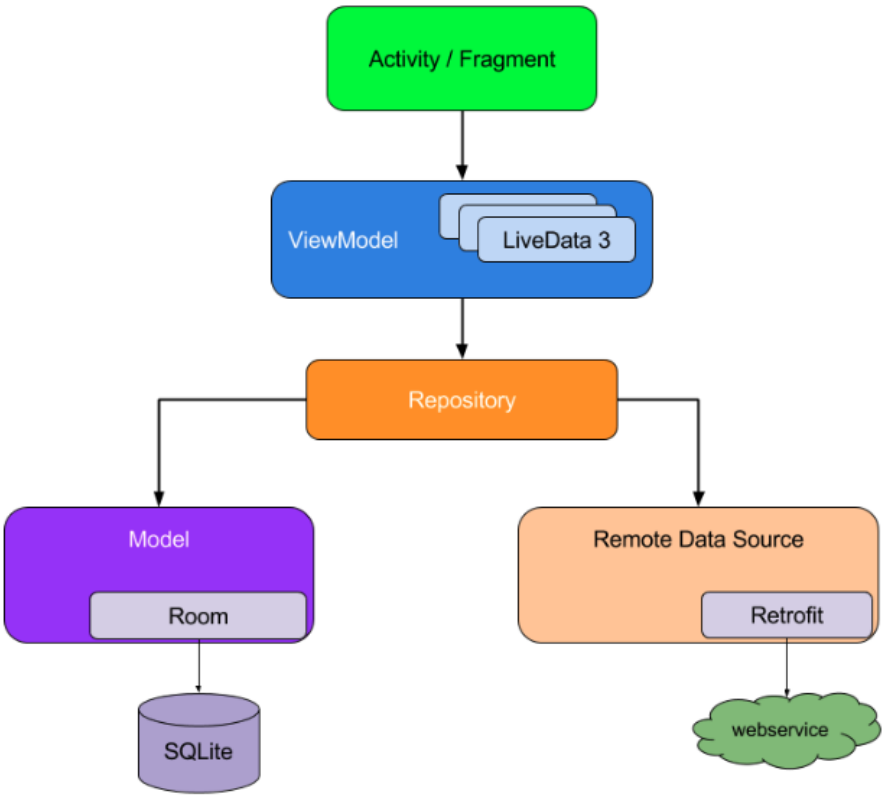


Figura 3.1: Arquitectura para aplicación Android recomendada por Jetpack [28]

En la figura se observa que cada componente sólo depende únicamente del componente inmediatamente inferior en términos de profundidad de la arquitectura hasta llegar a la base de datos o a un servicio web. Por ejemplo, las actividades y los fragmentos sólo dependen de un modelo de vista. El repositorio es el único componente que depende de más de un componente. En este ejemplo, el repositorio depende de un modelo de datos persistente y de una fuente de datos de backend remota.

Este diseño crea una experiencia del usuario consistente y agradable. Independientemente de que el usuario vuelva a la aplicación varios minutos después de cerrarla o varios días más tarde, verá al instante la información correspondiente al último estado actualizado. Si estos datos están inactivos, el módulo de repositorio comienza a actualizar los datos en segundo plano [28].

3.5 Modelo-Vista-Controlador

Si bien las aplicaciones escritas para el sistema operativo Android no requieren de un patrón de diseño [29], escoger uno es de vital importancia, no sólo para reducir costos de desarrollo y mantenimiento, sino también para garantizar la escalabilidad de la aplicación.

El objetivo de este modelo es separar la lógica de negocio de la lógica de presentación, y lograr que las modificaciones de cada una de las partes no afecten a la otra [29], mejorando así la mantenibilidad del sistema. El modelo consta de tres componentes: El modelo, la vista

y el controlador. El modelo representa los datos que serán consumidos por la aplicación y que serán mostrados en pantalla. La vista es el componente visual en pantalla; en nuestro caso, estará representado en los archivos xml del proyecto Android. Por último, el controlador (archivos .java asociados a los xml del proyecto) maneja los eventos desatados por las acciones del usuario y se comunica con el modelo. Además, el controlador se comunica con la vista de la aplicación si el modelo no requiere cambios (por ejemplo durante la acción de desplazamiento). En un patrón MVC, el modelo es totalmente independiente, es decir, no depende ni de la vista ni del controlador. Sin embargo, el controlador y la vista dependen del modelo. Este patrón de diseño establece que las vistas de la aplicación deben tener un solo controlador, mientras que un controlador puede ser compartido por más de una vista [29].

Si bien existen varias alternativas de uso para este patrón, aquí se utilizará Passive-MVC (Modelo-Vista-Controlador pasivo) que permite a la vista comunicarse con el modelo únicamente a través del controlador, a continuación una imagen representativa.

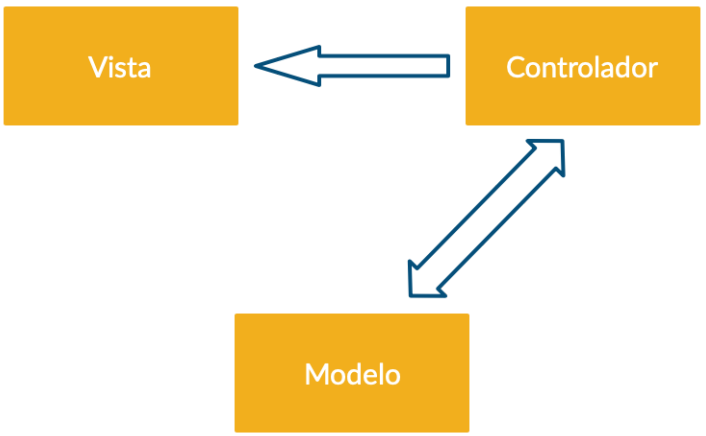


Figura 3.2: Modelo-Vista-Controlador Pasivo [29]

Para una mejor comprensión del funcionamiento de este patrón, se detalla a continuación un ejemplo de una petición real por parte de la vista hacia el modelo:

Una de las vistas principales de la aplicación permitirá al usuario buscar estilos de cervezas, productores o marcas a través de un componente *input* (componente de ingreso de texto en aplicaciones Android) en el cual el usuario ingresará la búsqueda por texto. Una vez que el usuario ingrese un caracter en el componente de la vista, el controlador tomará dicho caracter y enviará una consulta a la base de datos de cervezas y productores a través del controlador. Una vez que la base de datos (perteneciente al modelo) brinde una respuesta de las cervezas o productores que contienen dicho caracter, entonces el controlador tomará la información de este modelo y enviará una lista de todas las respuestas a la vista para que sean mostradas al usuario.

Se puede observar que, tanto en el ejemplo antes mencionado como en la figura 3.3 la única ruta de comunicación entre la vista y el modelo es a través del controlador de la

aplicación.

A continuación se adjunta el correspondiente diagrama de secuencias para una mejor comprensión.

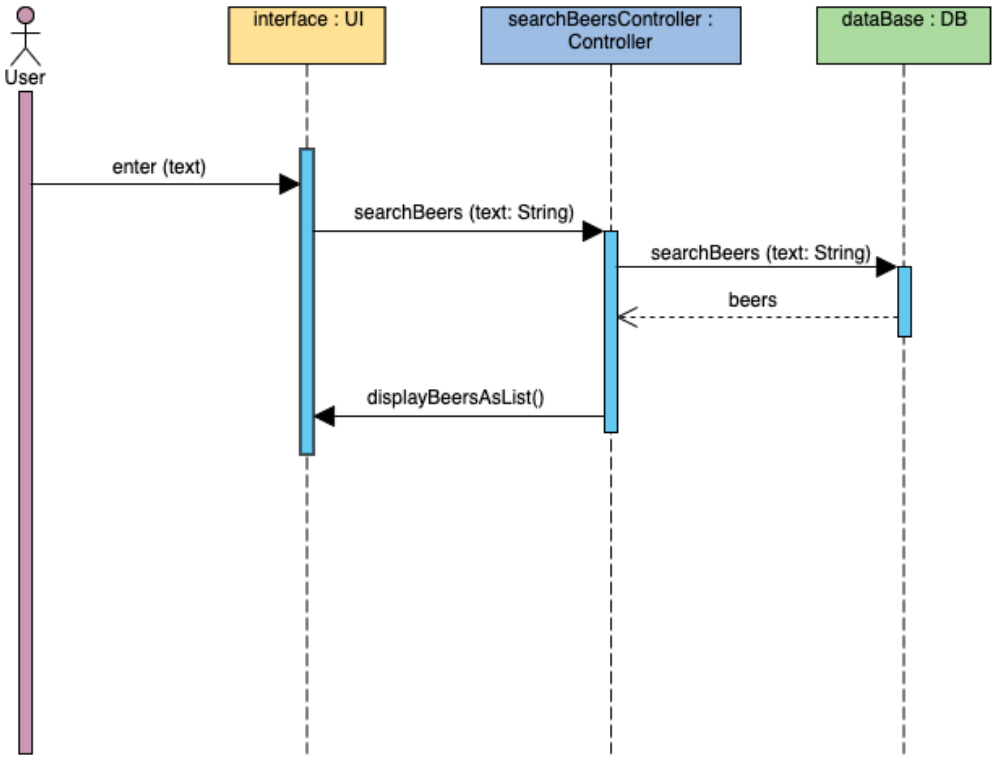


Figura 3.3: Diagrama de secuencias.

3.6 Casos de uso y wireframes de la aplicación

En este apartado se expondrá en primera instancia un diagrama de casos de uso de la aplicación, lo que ayudará a un mejor diseño del software. A continuación se presentarán los wireframes diseñados en base a dicho diagrama.

El diagrama de casos de uso correspondiente a los requerimientos planteados en la sección 1.3 es:

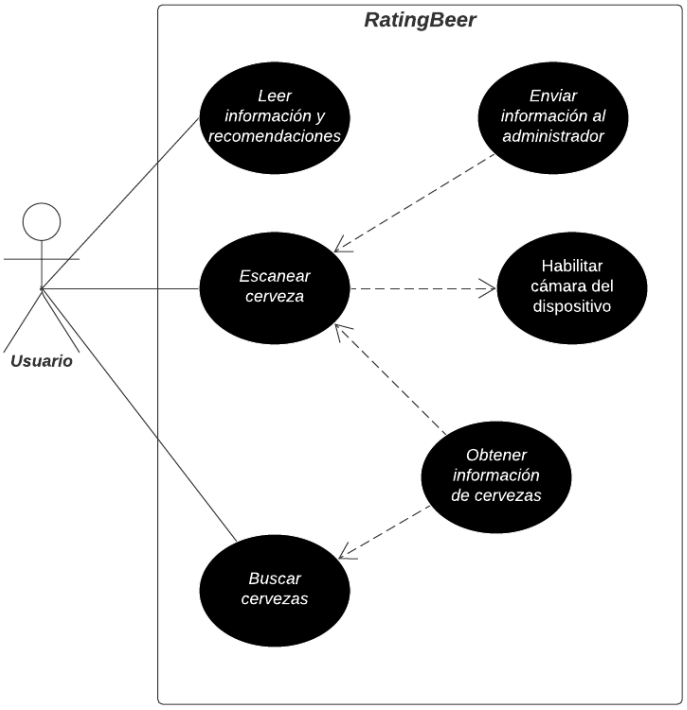


Figura 3.4: Diagrama de casos de uso.

En este diagrama se puede observar que el usuario puede leer recomendaciones de cómo catar una cerveza y obtener información acerca de los sabores y aromas no deseados que puede tener esta bebida. Además puede escanear una cerveza a través de una fotografía y en caso de que exista obtener información de la misma. Si la cerveza escaneada no existe, el usuario tendrá la posibilidad de enviar información sobre la misma al administrador del sistema para que sea evaluada. Finalmente también puede buscar una cerveza ingresando el estilo, el nombre o productor de la misma. Para esta opción, si la cerveza se encuentra en la base de datos, entonces el usuario podrá acceder a su información.

Este diagrama cumple con todos los requerimientos de diseño y será de suma importancia a la hora de cumplir con la totalidad de las funcionalidades requeridas.

A continuación se presentarán las pantallas que integrarán la aplicación. Cada una representa una actividad en la aplicación Android. Luego de analizar las aplicaciones existentes se ha optado por tomar una estética similar en el proyecto debido a que no tenemos conocimientos de experiencia de usuario para poder realizar un diseño de interfaces propio, y como estas aplicaciones están en el mercado y cuentan con muchos usuarios, suponemos que se ha pensado la experiencia del usuario y que el diseño lo ha realizado un profesional en el área. Por último, las pantalla ha sido pensada en base a los casos de uso planteados.

El prototipo ha sido realizado con la herramienta Pidoco [30].

Figura 3.5: (a) Diseño de actividad de bienvenida. (b) Diseño de actividad de pantalla principal.

La actividad de bienvenida se compone de:

- El logotipo de la aplicación.
- Nombre de la aplicación.

Esta actividad sirve como pantalla de inicio y se mostrará durante tres segundos. Además de presentar la aplicación móvil, también sirve para ocultar el proceso de peticiones y carga de la información de la actividad principal.

La actividad principal se compone de:

- Imagen de portada de la aplicación.
- Descripción del proyecto. Informará al usuario que la aplicación es producto de un proyecto final de carrera de la facultad de Ingeniería y Ciencias Hídricas de la

Universidad Nacional del Litoral.

- Valores no deseados en una cerveza y recomendaciones básicas acerca de cómo catar una cerveza artesanal.

Figura 3.6:(a) Diseño de actividad de detección de cerveza. (b) Diseño de actividad de listado y búsqueda de cervezas.

La actividad de detección de cerveza inicia al presionar sobre el botón de cámara del menú. En esta pantalla se activará la cámara del teléfono a través de la biblioteca OpenCV [24]. El usuario deberá tomar una fotografía a una botella de cerveza y, una vez guardada, la aplicación enviará la imagen al servidor para ser procesada.

El servidor será el encargado de preprocesar la imagen y en caso de no presente borrosidad y se detecte una botella en la imagen se ocupará de comparar las características principales de la imagen con las características guardadas en la base de datos correspondientes a las cervezas existentes en la aplicación. En caso de que la cerveza sea detectada se devolverá la información correspondiente a la misma y se ejecutará la actividad “Descripción de cerveza”. Por el contrario, en el caso de que la botella no se encuentre se ejecutará “Enviar datos al administrador”. Si se decide ejecutar el servidor de la aplicación sin detección de botellas el preproceso de la imagen solo verificará

si la imagen es borrosa o es aprobada para la posterior extracción de características.

Para una correcta interpretación se expone a continuación el diagrama de actividades correspondiente al flujo de operaciones antes mencionadas.

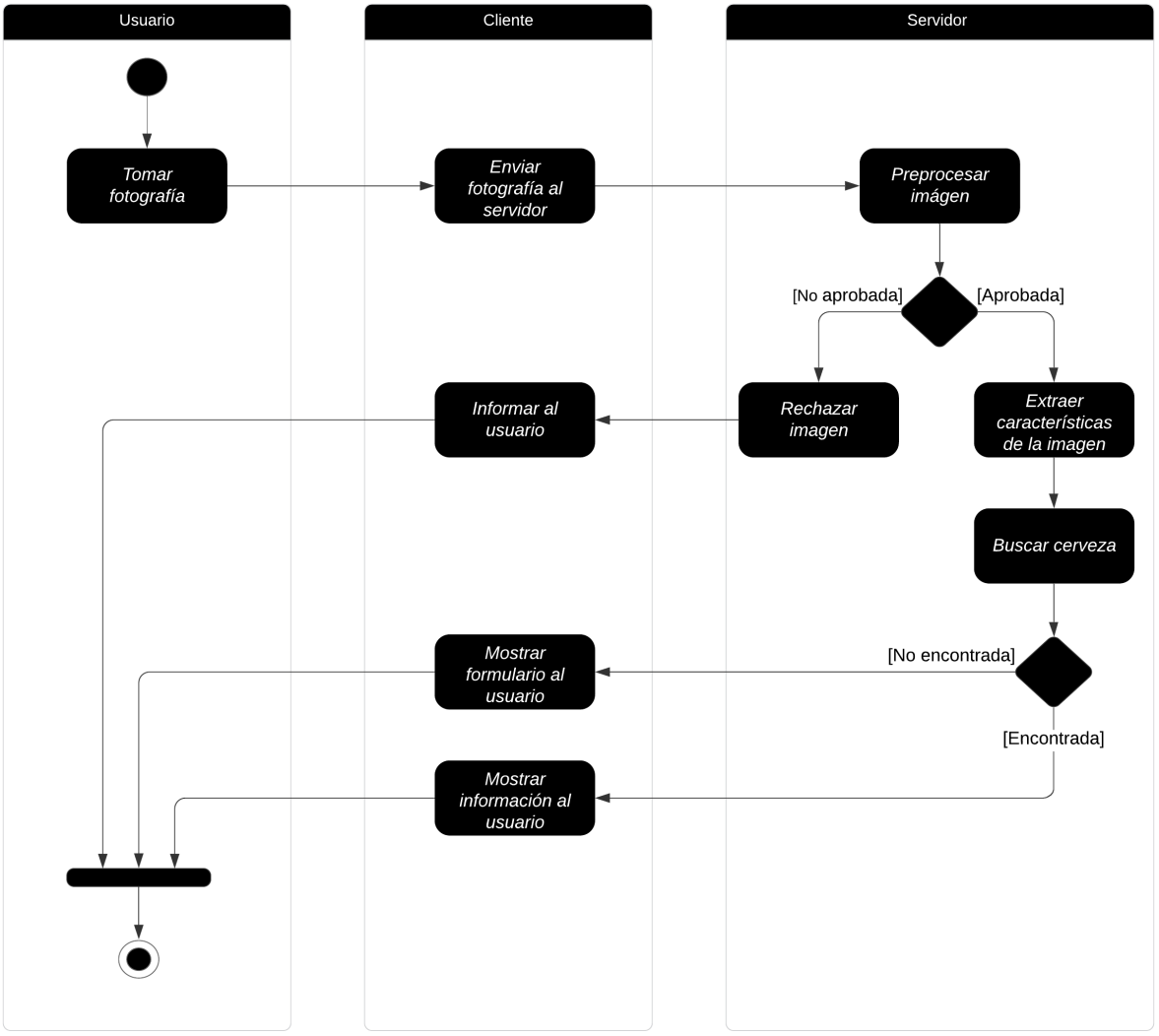


Figura 3.7: Diagrama de actividades del software.

Por otra parte, el diseño de la actividad de listado y búsqueda de cervezas se compone de:

- Imagen de portada de la actividad.
- Filtro de búsqueda. Este campo de ingreso de texto aceptará filtrar por cualquier valor de la cerveza:
    - Nombre,
    - Estilo,

- IBUs (amargor),
- ABV (alcohol),
- Descripción.

- Filtro por puntuación: Esta barra de puntuación permitirá filtrar cervezas por cantidad de estrellas.
- Listado principal de las cervezas existentes en la base de datos. Al seleccionar sobre una cerveza se iniciará la actividad “Descripción de cerveza” en la cual se desplegará toda la información acerca del ítem elegido.

A la presente actividad se ingresa presionando sobre el ícono de lupa “Buscar” del menú principal de la aplicación.

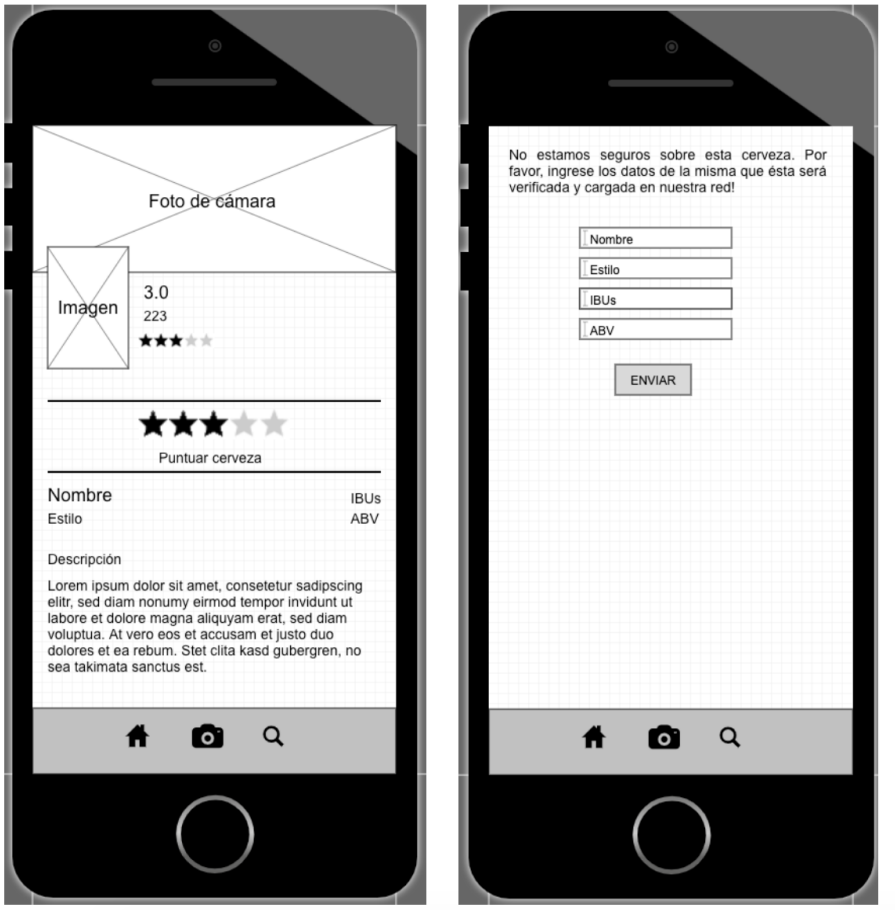


Figura 3.8: (a) Diseño de actividad de descripción de cerveza.(b) Diseño de actividad de enviar datos al administrador.

La actividad de descripción de cerveza se compone de:

- Imagen de portada de la actividad:
  - Si la actividad anterior es “Detección de cerveza” entonces esta imagen será

la foto tomada por la cámara.

    - Si la actividad anterior es “Listado y búsqueda de cervezas” esta imagen será tomada de la base de datos.

- Imagen principal de la cerveza. Será extraída de la base de datos y representa la botella o la etiqueta de la cerveza.
- Puntuación de los usuarios (3 puntos para este ejemplo).
- Cantidad de votaciones (223 para este ejemplo).
- Barra de Rating para que el usuario pueda puntuar la cerveza.
- Nombre de la cerveza.
- Estilo.
- Valores principales de alcohol (ABV) y amargor (IBUs).
- Descripción de la cerveza.

Además, por debajo aparecerá una lista de cervezas que pertenecen al mismo estilo.

La actividad para enviar datos al administrador de la aplicación se compone de:

- Texto explicativo acerca del desconocimiento de la cerveza por parte de la aplicación.
- Campos a completar por el usuario.
    - Nombre de la cerveza,
    - Estilo de la misma,
    - IBUs (escala de amargor),
    - ABV (grado de alcohol).
- Botón ENVIAR.

Una vez el usuario complete esta información acerca de una cerveza no encontrada, el software enviará los datos empaquetados en un formato JSON al servidor para que el administrador se encargue de verificar la información y luego cargar la nueva cerveza en la base de datos.

3.7 Desarrollo de la aplicación

El desarrollo de la aplicación fue llevado a cabo utilizando el lenguaje de programación Java para la parte cliente [31] y Python para el servidor [23]. Los entornos de trabajo elegidos fueron Android Studio [32] y Pycharm Edu [33].

La aplicación se ha desarrollado de manera tal que cuenta con el backend montado en un servidor remoto, el cual tiene conexión a una base de datos no relacional y un cliente implementado sobre el dispositivo móvil. Se muestra a continuación un diagrama de deployment del software desarrollado:

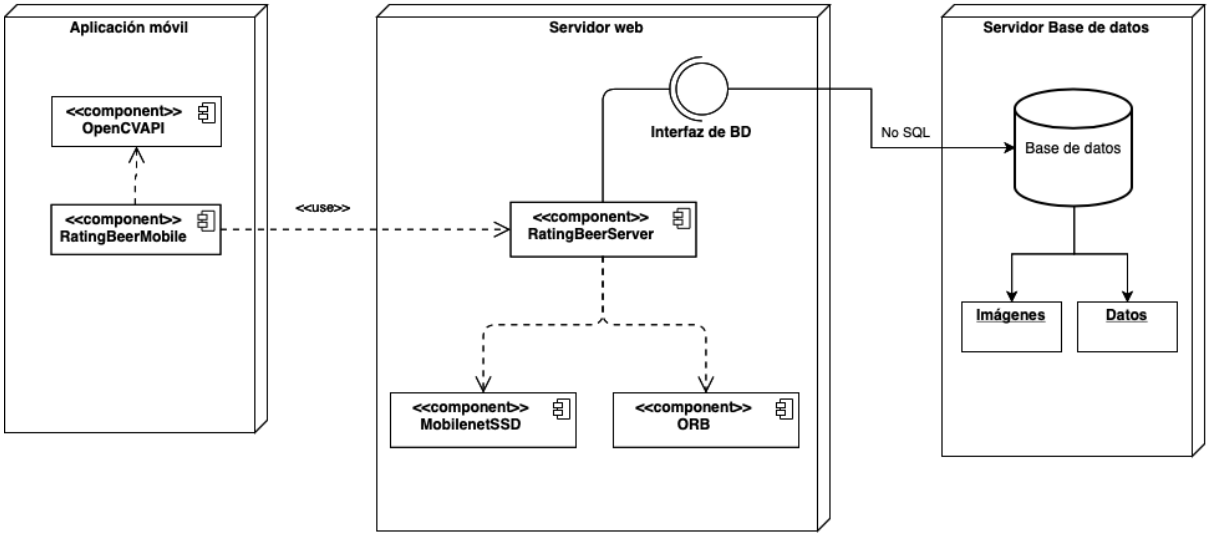


Figura 3.9: (a) Diagrama de deployment del software.

En la figura anterior se puede ver que en el cliente el componente RatingBeerMobile depende directamente de OpenCV, cuya api es integrada en este desarrollo. A su vez este componente puede realizar peticiones al servidor remoto. En el servidor, el componente principal RatingBeerServer depende estrictamente de los componentes MobileNetSSD y ORB para poder funcionar de manera correcta cuando quiere detectar una botella. Finalmente el componente principal puede hacer consultas a la base de datos no relacional MongoDB [27] a través de la interfaz que los conecta.

Se adjunta a continuación capturas de la aplicación desarrollada.

Figura 3.10: Vista de la pantalla de bienvenida.

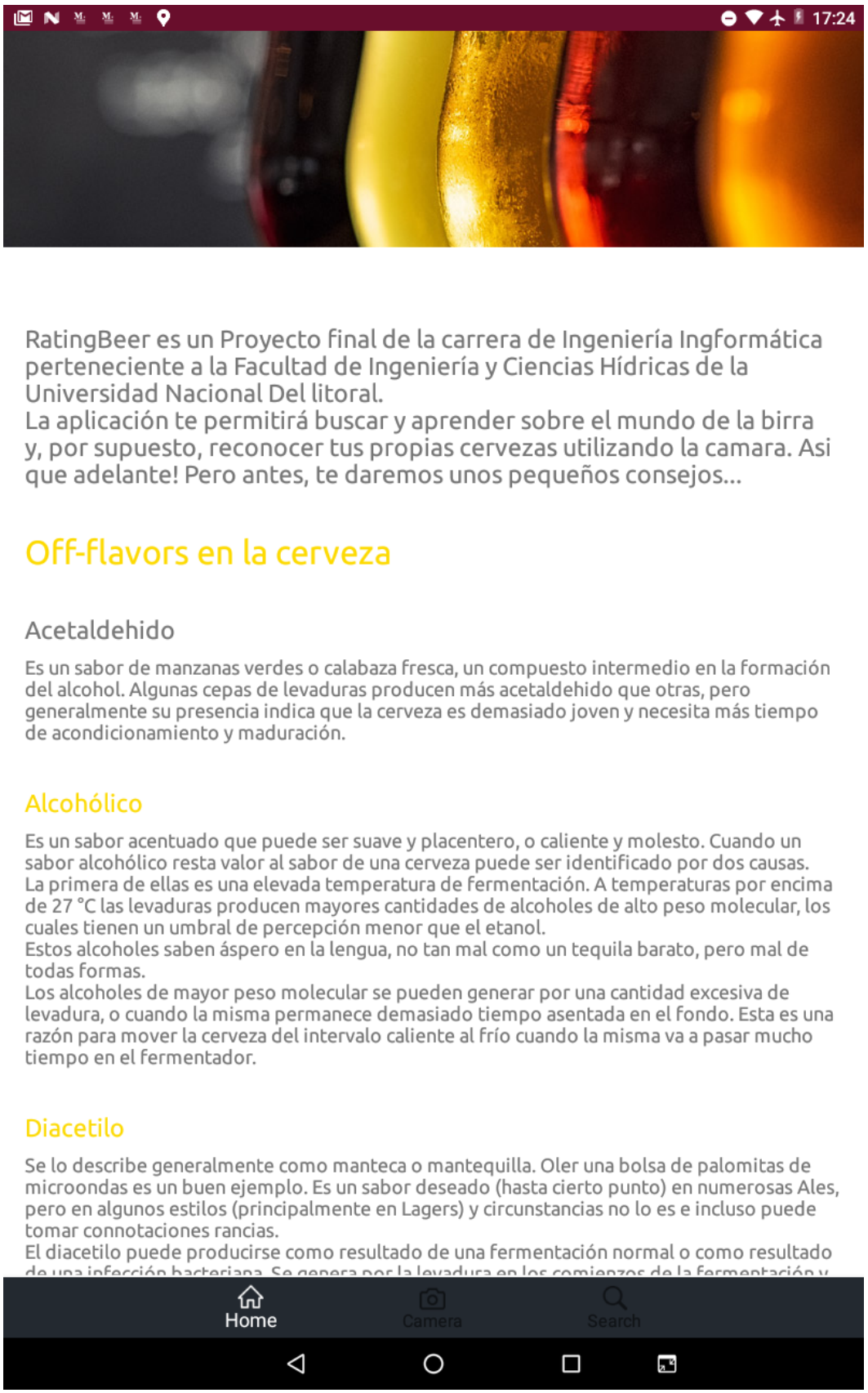


Figura 3.11: Vista de la pantalla principal.

Figura 3.12: Vista de la detección de cerveza.

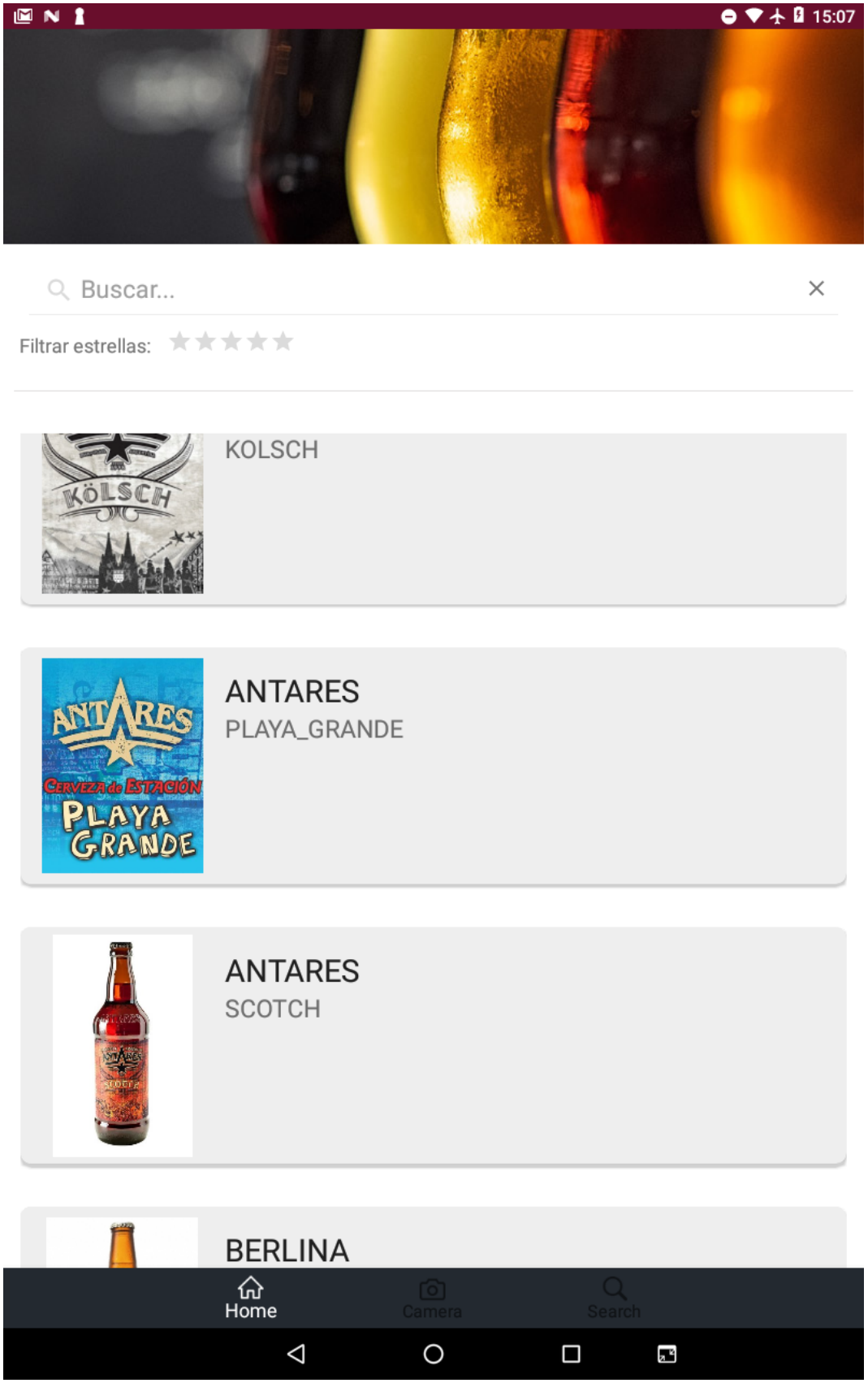


Figura 3.13:Vista de listado y búsqueda de cerveza.

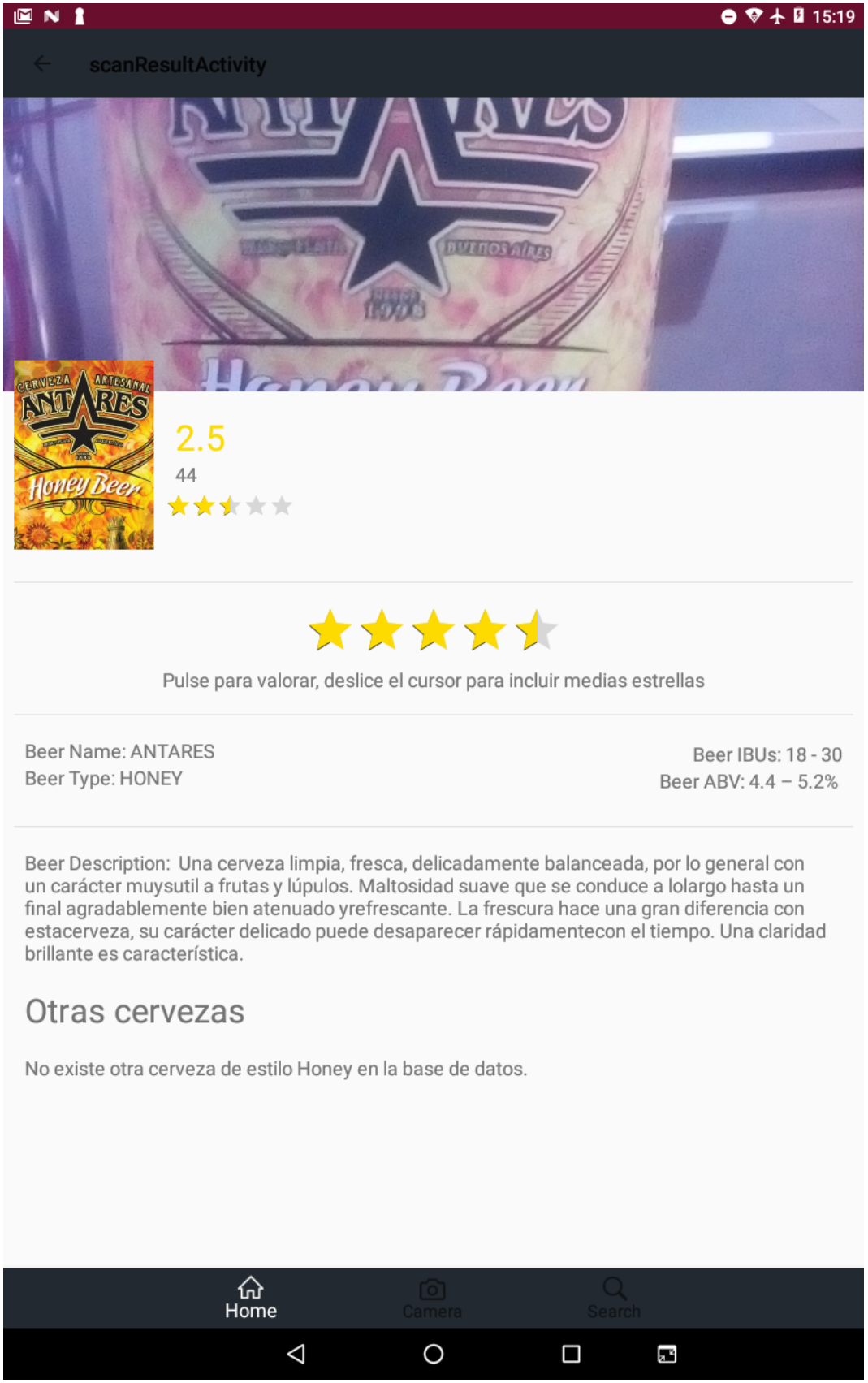


Figura 3.14: Vista de la descripción de una cerveza.

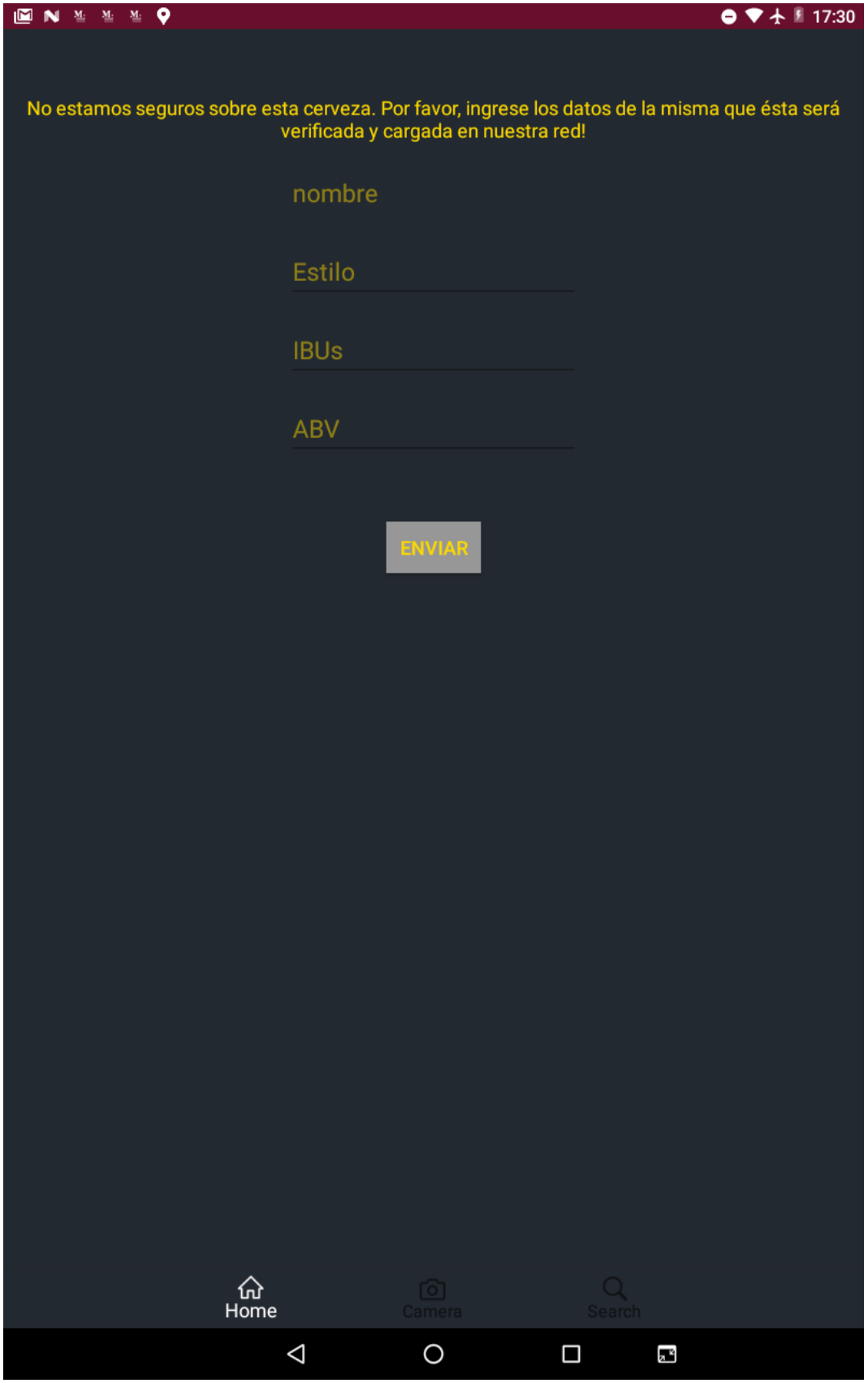


Figura 3.15: Vista de enviar datos al administrador.

4. Rendimiento de la aplicación

En este apartado se exponen pruebas que se han realizado y cambios sobre el diseño que han sido necesarios para el correcto funcionamiento del prototipo. Al ser un requisito que la aplicación funcione en teléfonos móviles de gama media se necesita un tiempo de respuesta eficiente para optimizar lo más posible la performance del software.

4.1 Eficiencia del algoritmo de comparación de imágenes

En un principio se probó el algoritmo de comparación de imágenes calculando los puntos claves de las imágenes almacenadas en la base de datos cada vez que un usuario tomaba una fotografía de una cerveza y hacía una consulta sobre esa cerveza. Sin embargo, rápidamente se llegó a la conclusión de que la forma de operar era ineficiente porque se perdía mucho tiempo de cómputo al realizar estas operaciones de manera repetitiva y esto hacía más lento el tiempo de respuesta de la aplicación. Entonces, se optó por calcular por única vez los puntos claves de cada imagen correspondiente a cervezas que son guardadas en la base de datos y guardar los vectores en un archivo. De esta manera, los puntos claves correspondientes a cada cerveza quedan a disposición y están listos para ser comparados con los puntos claves de la imagen que envía el usuario para una consulta.

Está claro que implementar la detección de puntos para imágenes almacenadas cada vez que el usuario realiza una consulta es un error de diseño, pero a fin de documentar la diferencia entre los tiempos de cómputo se muestra a continuación un gráfico que expone resultados de las pruebas realizadas sobre una notebook Macbook Pro con un procesador Intel Core I5 de 2,6Ghz, 8Gb de memoria RAM de 1600Mhz DDR3 y con un disco de estado sólido.

Figura 4.1: Comparación entre cálculo de puntos claves de imágenes obtenidos por única vez contra cálculos obtenidos en cada consulta. Cantidad de cervezas analizadas respecto del tiempo de ejecución de la petición. Tiempo medido en segundos.

El gráfico anterior expresa la diferencia entre calcular por única vez los puntos claves de las cervezas pre existentes en la base de datos y almacenados en un objeto contra el cálculo de dichos puntos cada vez que un usuario envía una consulta sobre una cerveza. La serie azul representa el software guardando los puntos claves de todas las cervezas de la base de datos por única vez y luego comparando dichos números con puntos claves solo de la captura tomada por el usuario. Por otra parte la serie roja representa el software calculando los puntos claves de todas las cervezas de la base de datos cada vez que se escanea una botella.

El cálculo de puntos claves por única vez ha reducido en gran medida el tiempo de respuesta de la aplicación, lo cual también ayuda la performance del software.

4.2 Preproceso de la imagen de entrada

Al momento de capturar una fotografía de una botella o lata de cerveza para luego verificar su existencia en la base de datos, es posible que el usuario tome una foto desenfocada o con movimientos. Una imagen con estas características no es válida como entrada al algoritmo ORB ya que se detectarán muy pocos puntos claves en la imagen. Además de esto se aceptarán imágenes de entrada que tengan una resolución mínima de 800x600 ya que el ORB ante una imagen de de menor resolución podría devolver menor cantidad de puntos claves que una imagen de mayor resolución y esto podría afectar los resultados de comparación de la imagen contra imágenes almacenadas en la base de datos.

Tanto las imágenes desenfocadas como imágenes de resoluciones menores a 800x600 no sólo aumentan el tiempo de cómputo de forma improductiva sino que proporcionarán datos erróneos al usuario, lo que puede generarle fastidio con la app siendo que el error está originado en la captura. Es por ésto, que antes de ejecutar el algoritmo y la comparación de puntos, el software realiza un preproceso de la imagen en el cual estudia primero si la resolución supera los 800x600 píxeles. En caso de que la resolución sea menor, entonces se informa al usuario que la aplicación no es compatible con dicho dispositivo. Por el contrario, si la imagen obtenida supera la resolución mínima se verifica si es borrosa o es apta para la comparación. Esto se implementa utilizando el laplaciano provisto por OpenCV [15]:

```
cv2.Laplacian(image, cv2.CV_64F).var()
```

Se evalúa la escala de grises de la imagen y se convoluciona con un kernel de 3x3 Laplaciano:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figura 4.2: Ejemplo de Kernel Laplaciano

Luego, se calcula la varianza de la respuesta. Si ésta es menor que un umbral previamente definido heurísticamente, entonces la imagen se considera borrosa y se devuelve un mensaje al usuario informando de que la foto no es apropiada. Por el contrario, si la imagen es aprobada continua la ejecución. Por experimentación el umbral se ha configurado en 200, lo que quiere decir que si una imagen tiene una varianza mayor a éste será aprobada y analizada por ORB [11].

A continuación se adjuntan dos imágenes. La imagen de la Figura 4.2.2 es evaluada en la etapa de preproceso por el laplaciano y arroja una varianza del laplaciano de 785.7639. Mientras que la imagen de la Figura 4.2.3 es preprocesada y la varianza del laplaciano es 31.9397. Esta última es una imagen que ha sido tomada en movimiento y no es aprobada como entrada a ORB. La razón por la cual no se aprueba una imagen es que el algoritmo ORB detecta la mayoría de puntos claves en bordes existentes en la imagen y si la imagen sufre borrosidad o movimiento estos bordes se pierden y ORB no devolverá un resultado eficaz.

Figura 4.3: Fotografía de una botella sin movimiento y bordes bien definidos. La misma es aprobada por el algoritmo de preproceso de imágenes.

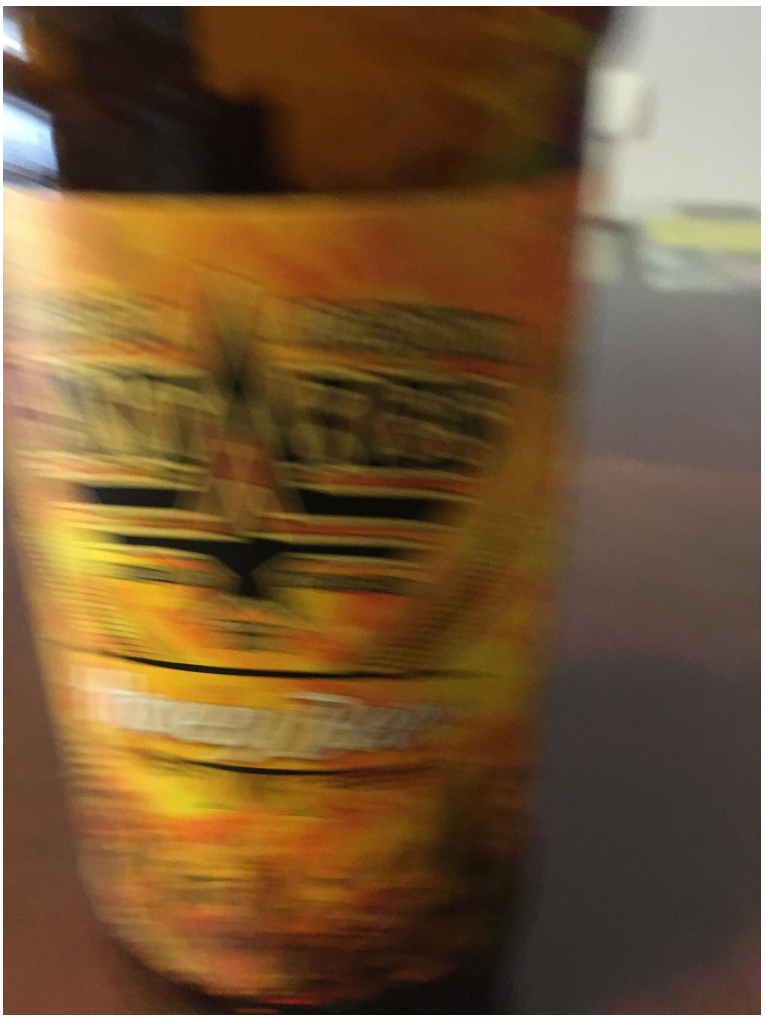


Figura 4.4:Fotografía de una botella con movimiento y bordes no definidos. La misma es rechazada por el algoritmo de preproceso de imágenes.

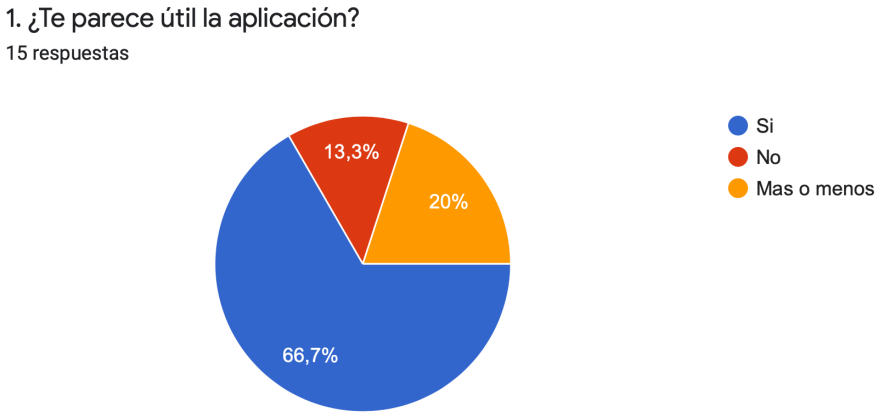
Para verificar que la configuración del umbral es aceptable se ha realizado un experimento con treinta imágenes de cervezas para ver cual es el porcentaje de imágenes que pasan la etapa de preproceso. Se ha detectado que el 86.66% (26 fotografías) han pasado el preproceso de imágenes lo cual quiere decir que son imágenes de cervezas aptas para buscar en la base de datos de RatingBeer. Por el contrario, cuatro imágenes no han superado la prueba por no tener sus bordes bien definidos y han sido descartadas por el preproceso de imágenes de la aplicación.

Por último, obtener por única vez los puntos característicos correspondientes a las imágenes y realizar un preproceso de la imagen de entrada ha permitido que la aplicación tenga un tiempo de respuesta razonable y aceptable. Además se ha realizado una prueba de principio a fin en la que se ha comprobado que todas las funcionalidades desarrolladas en la aplicación cumplen con los alcances del proyecto y funcionan de manera correcta.

5. Evaluación de la aplicación basada en la experiencia del usuario

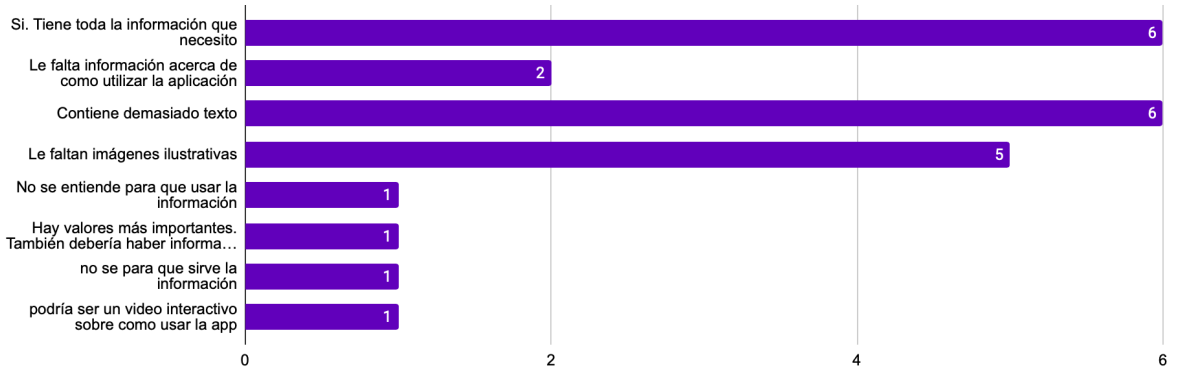
Con el fin de obtener una evaluación por parte de potenciales usuarios acerca de la aplicación realizada se ha realizado una encuesta a quince personas, la cual se expone en esta sección. Se eligieron las personas con el objetivo de obtener la devolución de individuos que estén informados sobre cerveza artesanal e individuos que sean consumidores casuales.

La estructura de la encuesta se compone de siete preguntas. A continuación se expone cada pregunta y sus resultados



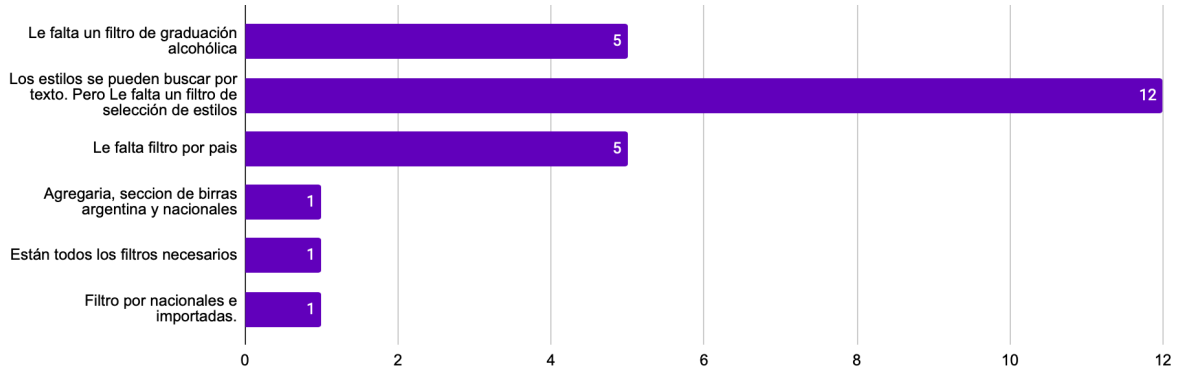
3. ¿Está completa la información de la pantalla inicial? (marcar todas las que considere)

15 respuestas



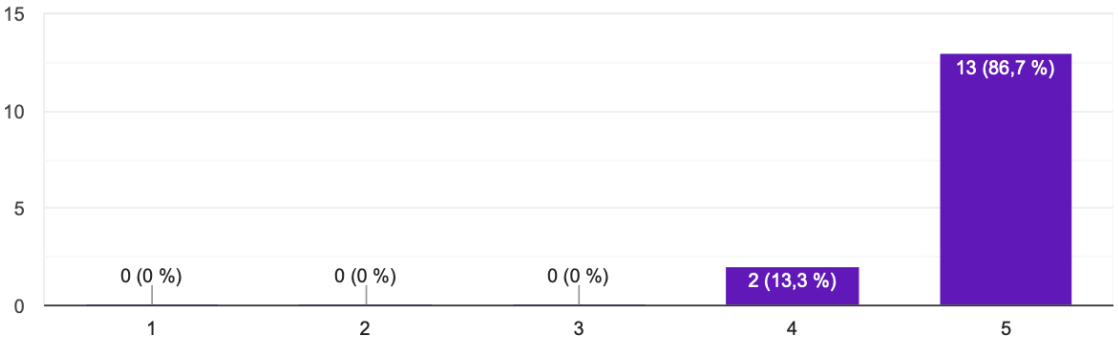
4. ¿Agregarías o quitarías algún filtro de búsqueda? (marcar todas las que considere)

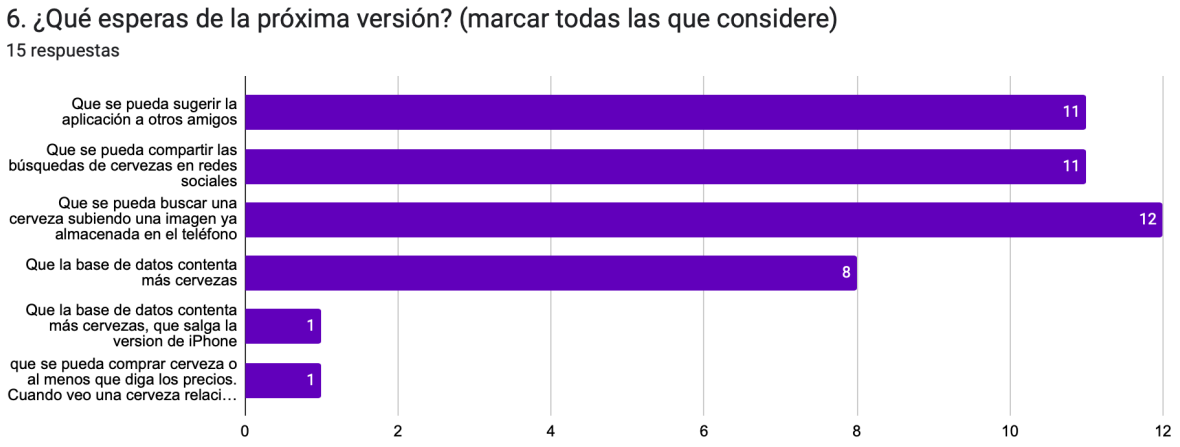
15 respuestas

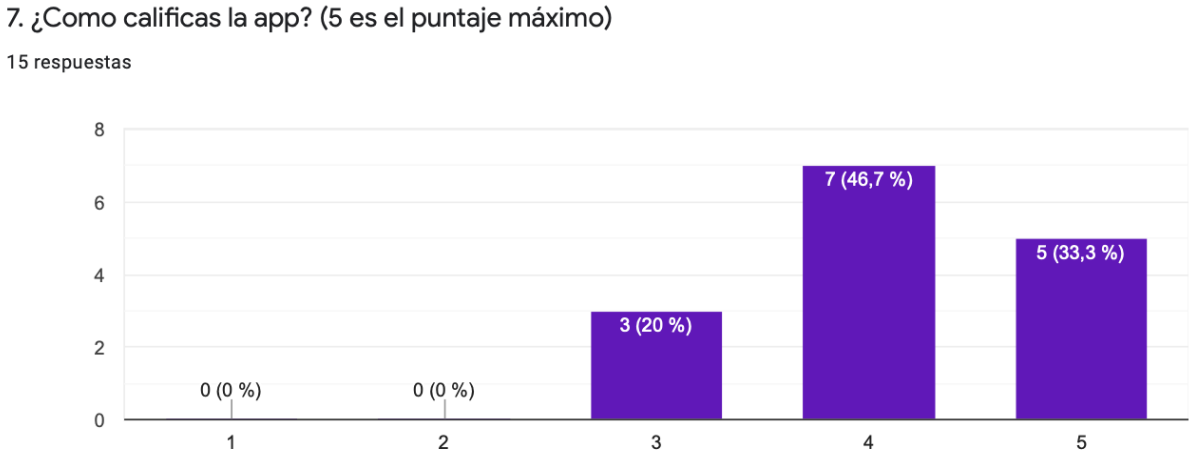


5. ¿Qué puntaje le das a la velocidad con la que responde la app? (5 es el puntaje máximo)

15 respuestas







Los resultados de la encuesta revelan que al 66,7% de los encuestados les resulta útil la aplicación mientras que el resto la consideran parcialmente útil o inutil. Sin embargo, el 73,3% de las personas (un porcentaje mayor a los que contestaron que les parece útil) instalarían la aplicación en su dispositivo móvil. En cuanto a la pantalla inicial, la cual busca orientar al consumidor sobre cómo catar una cerveza, ha recibido críticas por parte del 40% de los encuestados de que contiene demasiado texto, mientras que otro 40% coincide en que la información es necesaria. El resto de las personas reparten sus opiniones. Respecto de los filtros de búsqueda de cervezas ha quedado en evidencia que no son suficientes para una aplicación que busca atraer consumidores inexpertos. Por último, las personas coinciden en que la velocidad de la aplicación es muy buena, sin embargo, el puntaje final de la misma se reparte entre tres, cuatro y cinco estrellas (siendo cuatro sobre cinco el puntaje más votado). Esto se debe a que las personas encuestadas afirman que deberían poder compartir sus búsquedas en redes sociales, compartir la aplicación con amigos y cargar fotos que ya tienen en su dispositivo.

Como conclusión, se observa una calificación general positiva de la experiencia de usuario

con la aplicación. Sin embargo le faltan características y modificaciones para salir al mercado y ponerla a disposición de la sociedad, que serán abordadas como trabajos futuros.

6. Conclusiones y trabajos futuros

Para poder llevar a cabo este proyecto se ha realizado un profundo análisis del estado del arte teniendo en cuenta que el software ha sido pensado para el uso en dispositivos móviles desde un principio. En este sentido, el rendimiento en términos de costo computacional ha sido crucial para seleccionar los algoritmos, en especial aquel que se utiliza para la detección de puntos claves de una imagen. De este punto se concluye que ha sido fundamental para la eficiencia del software que los puntos claves obtenidos de las imágenes se almacenen en la base de datos y no se calculen siempre que un usuario escanee una cerveza. También ha sido de gran importancia para el desempeño de la aplicación el diseño de la arquitectura guiado por un patrón de diseño recomendado durante el desarrollo. Siguiendo este patrón se ha logrado un un código mantenible y escalable, lo cual no hubiera podido ser posible en caso de no seguir un patrón.

Se han evaluado tres algoritmos del campo de la visión computacional para la detección de características de una imagen. Además, se ha priorizado que el método seleccionado debe ser robusto ante alteraciones de la imagen, como la iluminación, el ruido y las transformaciones afines. La elección del método ha sido muy importante para lograr un resultado aceptable ante cambios como los antes mencionados.

Scale Invariant Feature Transform (SIFT) [7] es un método muy eficiente pero requiere de un gran costo computacional por lo cual se ha descartado desde un principio ya que no cumple con que la aplicación funcione en dispositivos de media gama. Mientras que SURF [8] mejora el rendimiento en términos de costo computacional respecto del SIFT a costa de la precisión, motivo por el cual no se ha elegido. Finalmente, se seleccionó ORB [11] porque tiene un rendimiento similar a SIFT pero se ve menos afectado por el ruido y es capaz de usarse en tiempo real por un dispositivo de poca memoria.

Además se ha optado por presentar dos versiones del prototipo. Una que implementa en el lado del servidor la combinación de Mobilenet [14] y SSD [16] y que detecta de manera automática si la fotografía tomada por el usuario contiene una botella de cerveza, advirtiéndole si no detecta botella alguna que debe tomar nuevamente la fotografía de la cerveza y otra que no implementa la red con el fin de brindar la posibilidad de buscar cervezas enlatadas. Se pretende para versiones futuras reentrenar la red para poder unificar estas dos versiones en una sola que pueda detectar correctamente latas de cerveza.

Finalmente, se logró desarrollar el servidor de la aplicación en el lenguaje Python, que realiza las tareas de preproceso de imágenes, extracción de puntos característicos en una imagen y comparación entre imágenes, además de contar con métodos para consultas a la base de datos e implementar la red neuronal para la detección de botellas. El cliente de la aplicación se desarrolló en Android respetando la arquitectura Jetpack, la cual presenta una muy buena performance, y un patrón de diseño Modelo-Vista-Controlador pasivo que permite una comunicación bidireccional entre los componentes del sistema a través de una única ruta. En cuanto a la combinación de estos lenguajes no se ha presentado problema alguno a la hora de utilizarlos.

En lo que respecta de la captura de una cerveza que el usuario envía al servidor para ser procesada y comparada con las imágenes preexistentes en la base de datos se ha

desarrollado un método de preproceso de la misma que verifica que la imagen tomada no presente movimientos o efecto de borrosidad, ya que esto perjudica la detección de características de la misma. En caso de aprobar este preproceso, entonces se extraen sus características matemáticas y se comparan con las características guardadas correspondientes a las cervezas almacenadas en la base de datos. En caso de coincidencias, la aplicación retornará al usuario información sobre la cerveza fotografiada y en el caso contrario le pedirá al usuario que envíe información sobre la cerveza para ser evaluada por los administradores y cargada en la base de datos.

Como trabajos futuros se proponen las siguientes tareas:

- Incorporación de un sistema de logueo de usuario para así poder resolver de manera eficiente la puntuación de cervezas no permitiendo que el mismo usuario valore reiteradas veces un mismo producto.
- Permitir a los usuarios guardar productos favoritos y poder compartir su afinidad de un producto en redes sociales.
- Permitir a los usuarios compartir la aplicación con otras personas a través de distintos medios con el fin de publicitar la misma.
- Desarrollar la versión compatible para iOS.
- Entrenar un modelo de red neuronal que sirva tanto para la detección de botellas de cerveza como de latas y modificar la aplicación para que la misma evalúe automáticamente cuando obtener la fotografía de una cerveza a partir de la detección automática de objetos.
- Ejecutar pruebas de escalabilidad con una base de datos mucho mayor a la provista para el desarrollo del prototipo.
- Reevaluar e implementar un sistema de filtros que contengan mayor cantidad de opciones.
- Incorporar recomendaciones sobre cervezas para atraer a consumidores inexpertos.

Referencias

[1] La cerveza artesanal gana terreno: Proyecciones del negocio de moda. (2018, August 04). Retrieved March 27, 2021, from https://www.cronista.com/pyme/negocios-pyme/Cerveza-artesanal-un-mercado-efervescente-que-crecio-40-20180801-0003.html

[2] VIVINO. (n.d.). VIVINO. Retrieved August 17, 2020, from https://www.vivino.com

[3] Delectable - Scan & Rate Wine. *App Store*, 5 Apr. 2012, apps.apple.com/us/app/delectable-scan-rate-wine/id512106648.

[4] Untappd - discover beer. (2011, September 29). Retrieved March 30, 2021, from https://apps.apple.com/us/app/untappd/id449141888

[5] BJCP. (n.d.). Beer Judge Certification Program. Retrieved August 17, 2020, from https://www.bjcp.org

[6] Sucar, L. E., & Gómez, G. (2011). Visión computacional. Instituto Nacional de Astrofísica, Óptica y Electrónica. Puebla, México.

[7]Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

[8] Bay, H., Tuytelaars, T., & Van Gool, L. (2006, May). Surf: Speeded up robust features. In European conference on computer vision (pp. 404-417). Springer, Berlin, Heidelberg.

[9] Introduction to ORB (Oriented FAST and Rotated BRIEF). (2019). Retrieved 4 December 2019, from https://medium.com/software-incubator/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[10] Karami, E., Prasad, S., & Shehata, M. (2017). Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images. arXiv preprint arXiv:1710.02726.

[11] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. R. (2011, November). ORB: An efficient alternative to SIFT or SURF. In ICCV (Vol. 11, No. 1, p. 2).

[12] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

[13] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

[14] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[15] Laplace Operator. (n.d.). Retrieved March 23, 2020, from https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html

[16] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016,

October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[17] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7310-7311).

[18] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems(pp. 91-99).

[19] Dai, J., Li, Y., He, K., & Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. In Advances in neural information processing systems (pp. 379-387).

[20] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[21] Android. (2019). Retrieved 4 December 2019, from http://www.android.com/

[22] Mobile Operating System Market Share Argentina | StatCounter Global Stats. (2019). Retrieved 4 December 2019, from https://gs.statcounter.com/os-market-share/mobile/argentina

[23] Python.org. (2019). Retrieved 4 December 2019, from https://www.python.org/

[24] OpenCV. (2019). Retrieved 4 December 2019, from https://opencv.org/

[25] Node.js. (n.d.). Retrieved March 27, 2021, from https://nodejs.org/es/

[26] Javascript. (n.d.). Retrieved March 27, 2021, from https://developer.mozilla.org/es/docs/Web/JavaScript

[27] MongoDB. (n.d.). Mongo DB. Retrieved August 28, 2020, from https://www.mongodb.com

[28] Guía de arquitectura de apps | Desarrolladores de Android. (2019). Retrieved 4 December 2019, from https://developer.android.com/jetpack/docs/guide?hl=es-419

[29] Sokolova, K., Lemercier, M., & Garcia, L. (2013, May). Android passive MVC: a novel architecture model for the android application development. In International Conference on Pervasive Patterns and Applications.

[30] Pidoco - The Rapid Prototyping Tool. (2020). Retrieved 16 March 2020, from https://pidoco.com/

[31] Java | Oracle. (2020). Retrieved 16 March 2020, from https://www.java.com/es/

[32] Download Android Studio and SDK tools | Android Developers. (2020). Retrieved 16 March 2020, from https://developer.android.com/studio

[33] Pycharm Edu. (n.d.). Pycharm Edu. Retrieved September 4, 2020, from

https://www.jetbrains.com/es-es/pycharm-edu/

[34] Review: MobileNetV1—Depthwise Separable Convolution (Light Weight Model). (2019). Retrieved 4 December 2019, from https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69