Desarrollo de una aplicación Android, asistida por visión artificial, para administrar imágenes almacenadas en un dispositivo móvil

PROYECTO FINAL DE CARRERA

Yackel, Francisco

Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral

Diciembre 2018

Documento maquetado con TEXIS v.1.0+.

Desarrollo de una aplicación Android, asistida por visión artificial, para administrar imágenes almacenadas en un dispositivo móvil

Memoria que presenta para optar al título de Ingeniero en Informática

Yackel, Francisco

Dirigida por el Doctor

Vignolo, Leandro

Codirigida por el Ingeniero

Ferrado, Leandro

Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral

Diciembre 2018

Resumen

No basta tener un buen ingenio, lo principal es aplicarlo bien.

René Descartes

En la actualidad, el constante avance de las tecnologías de comunicación permite que la transmisión de información multimedia sea un aspecto común y diario para las personas que utilizan dispositivos móviles. El fenómeno de las redes sociales, trae consigo un tráfico de datos que muchas veces se hace difícil de administrar y controlar por los usuarios, produciendo que se alcance el límite de almacenamiento en los dispositivos, principalmente en aquellos de gama baja o media, aunque ocasionalmente sucede también en los últimos modelos. Por otro lado, cuando se quieren administrar los archivos de un dispositivo móvil, o bien al realizar una copia de seguridad, las tareas llevadas a cabo para organizar las imágenes consumen mucho tiempo.

Gracias a los grandes avances que hoy en día provee la visión artificial, una disciplina que combina técnicas de procesamiento de imágenes y aprendizaje maquinal, se considera posible lograr un sistema que cuente con las propiedades adecuadas para lograr un agrupamiento o *clustering* de datos, basándose en una similitud computada sobre lo mismos. Con ello, se podría sugerir al usuario una estructura de directorios para organizar las imágenes de su dispositivo, y a partir de ahí proveer funcionalidades para agilizar y automatizar las tareas de administración involucradas, reduciendo el tiempo y esfuerzo requerido.

Por lo tanto, la propuesta de este proyecto es implementar una aplicación que agrupe, de manera automática, las imágenes de un directorio seleccionado por el usuario. Luego, una vez realizado el procesamiento, éste podrá tomar la decisión que desee interactuando con el resultado (e.g eliminar un grupo obtenido, fusionarlo con otros, renombrar carpetas, ingresar en un grupo y eliminar alguna imagen en particular, etc.). Dicha aplicación será implementada para la plataforma Android, dado que es un sistema operativo muy popular y además gratuito.

Palabras claves: imágenes, agrupamiento, administración y control, similitud computada.

Índice

Índice de figuras

Índice de Tablas

Parte I

Conceptos básicos

En esta primera parte del documento se presentan los conceptos manejados en este trabajo para definir el tratamiento realizado. Esto comprende los elementos que componen la especificación del proyecto y los fundamentos que sustentan las soluciones propuestas para alcanzar los objetivos planteados. Se realiza una breve aclaración de todos estos recursos, introducidos en un orden conveniente, para dar contexto a cómo se propone encarar el proyecto y cuál es la base identificada para poder realizar ello con éxito.

Capítulo 1

Introducción

La simplicidad es la máxima sofisticación.
Leonardo da Vinci

Resumen: En este capítulo se presenta la motivación del trabajo y los elementos que componen la especificación del proyecto. Se hace una reseña de la motivación del proyecto y cómo se justifica el trabajo realizado, aclarando los objetivos planteados y las restricciones tenidas en cuenta para el alcance y las tecnologías utilizadas.

1.1. Motivación

El aprendizaje maquinal (**ML**: del inglés, *"machine learning"*) es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de información suministrada en forma de ejemplos.

Hace ya unos años, la industria de la información y la tecnología (IT) está viviendo grandes logros y consiguiendo enormes inversiones monetarias en lo que refiere a ML[31]. La realidad es que este campo no siempre ha sido tan prolífico, alternando épocas de altas expectativas y avances con "inviernos" donde sufría severos estancamientos. A partir del 2006, gracias al aumento de la potencia de cálculo junto con la gran abundancia de datos disponibles, han vuelto a lanzar el campo del aprendizaje maquinal. Numerosas empresas están transformando sus negocios hacia el dato y están incorporando estas técnicas en sus procesos, productos y servicios para obtener ventajas competitivas sobre la competencia[22].

Una disciplina que pertenece al aprendizaje maquinal, la cual incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador, se denomina visión artificial (**CV**: del inglés, *computer vision*). Tal y como los humanos usamos nuestros ojos y cerebros para comprender el mundo que nos rodea, la CV trata de producir el mismo efecto

para que los ordenadores puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación[21]. Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística, la física y otras disciplinas.

El auge de las aplicaciones para dispositivos móviles ha traído consigo la necesidad de incorporar este tipo de tecnologías. La evolución de los teléfonos inteligentes y su uso masivo, hicieron que empresas de desarrollo de software se acoplen a esta demanda, ofreciendo productos y servicios para que todas las personas puedan utilizarlos en sus propios dispositivos[32]. El área que comprende el procesamiento de imágenes y aprendizaje maquinal no escapa a esta ola tecnológica, por lo que un conjunto de bibliotecas que permiten el procesamiento de imágenes, empezaron a estar disponibles para su uso y adaptabilidad de una manera optimizada y preparada para que dispositivos de todo tipo de gama puedan aprovechar de sus beneficios. A continuación, se listan algunas aplicaciones destacadas que utilizan procesamiento de imágenes e inteligencia computacional:

- Google entrenó una red neuronal profunda que logró reconocer con buena precisión tanto rostros y cuerpos humanos como también gatos en videos de YouTube, lo cual se obtuvo de forma no supervisada sin utilizar información de clases (es decir, nunca se le daba información de lo que estaba aprendiendo) [29].

- El sistema DeepFace de Facebook alcanzó una precisión de 97.35 % con redes neuronales profundas sobre el conjunto de datos *Labeled Faces in the Wild* (LFW), reduciendo el error del actual estado del arte en más de un 27 % y acercándose a la precisión de un humano en reconocimiento de rostros (que es de aproximadamente un 97.53 %) [35].

- Google, a través de su producto *Google Photos*, ofrece una solución denominada *Organize*, la cual realiza tareas de organización sobre las imágenes de sus usuarios. Ésta proporciona distintos conjuntos de imágenes basándose en características u objetos que tengan en común, como pueden ser perros, autos, el cielo, etc[30].

- La aplicación fotos, perteneciente al sistema operativo Windows 10, permite buscar texto en las imágenes. La misma utiliza una tecnología denominada OCR[**?**] (reconocimiento óptico de caracteres), la cual permite detectar no sólo a la palabra, sino también la posición tiene en la imagen.

Todos los ejemplos enumerados y la gran mayoría de soluciones, realizan el procesamiento de la información (texto, imágenes o videos) aprovechando los recursos que ofrece la "nuble"(*cloud*). Para el usuario final, esto tiene un costo no menor, el cual significa aceptar que su información será compartida (y quizás utilizada) con la compañía que le esté brindando el servicio. Es por ello que una restricción principal para este proyecto fue decidir que la misma será *on premise*, lo que significa que el procesamiento de la información se realizará exclusivamente en el dispositivo, sin la necesidad de establecer una conexión a Internet para poder aprovechar alguna de sus funcionalidades.

1.2. Justificación

Lo que se busca en este proyecto es implementar una aplicación que asista al usuario en la tarea de organizar las imágenes de su dispositivo. Como etapa inicial, la aplicación provee una estructura de directorios mediante una metodología basada en algoritmos de visión artificial, formando grupos de imágenes según el grado de similitud que comparten. Luego, una vez realizado el procesamiento, el usuario podrá interactuar con el resultado (e.g eliminar un grupo obtenido, renombrar carpetas, ingresar en un grupo y eliminar alguna imagen en particular, etc.) para lograr la estructura/el agrupamiento que desee.

La aplicación es de código abierto, con el fin de poder hacer una contribución a la comunidad del software libre y que pueda ser continuamente mejorada o versionada y personalizada en base a distintos desarrolladores que presenten interés por la misma. Como se dijo en la sección anterior, se propone que sea *on premise* (i.e. el proceso se realiza únicamente en el dispositivo móvil) en lugar de ser un servicio *cloud*. De esta manera se puede asegurar la confidencialidad de los datos que se manejan, al mismo tiempo lograr que el despliegue y mantenimiento del sistema sean aspectos más sencillos y económicos.

Por último, será implementada para la plataforma Android, dado que es un sistema operativo muy popular y gratuito. Esta decisión se basa en que, a nivel mundial, Android es el líder indiscutido por cantidad de usuarios, y lo mismo sucede en la Argentina, ya que se encuentra en el 85 % de los equipos, según datos del mercado [33]. Además, es destacable que se trata de una plataforma de código abierto que brinda a los desarrolladores un amplio conjunto de herramientas de desarrollo y librerías a su disposición, las cuales a su vez cuentan con extensa documentación de respaldo.

1.3. Objetivos

Para desarrollar este proyecto, se plantearon los siguientes objetivos:

1.3.1. Objetivos generales

- Desarrollar una aplicación móvil sobre la plataforma Android que, a partir de imágenes almacenadas en el dispositivo, sugiera una estructura de directorios para organizar el conjunto, según una medida de similitud calculada entre los elementos.
- Proveer al sistema funcionalidades que le permitan al usuario la interacción con el resultado, para desempeñar distintas tareas de administración sobre el almacenamiento comprendido por las imágenes.

1.3.2. Objetivos específicos

- Analizar distintos algoritmos y métodos tanto para la extracción de características de las imágenes como para el clustering de las mismas.
- Definir la función o noción de similitud que se deba computar entre las imágenes a agrupar.

- Definir un conjunto de imágenes a utilizar en experimentos, que sea lo más fiel posible a la problemática tratada en el marco de dispositivos móviles.
- Desarrollar un protocolo de experimentación en base al conjunto de datos definido, para evaluar el desempeño de las técnicas investigadas.
- Llevar adelante las pruebas definidas en el protocolo de experimentación.
- Analizar y definir la metodología de visión artificial a implementar en el sistema, priorizando las técnicas según su desempeño en la experimentación así como la posibilidad de ajustarse a los recursos con los que cuentan la mayoría de los dispositivos.
- Lograr una interfaz de usuario sobre la cual podrá elegir el directorio/directorios a procesar, y que sea suficiente para la administración e interacción necesaria con el resultado del clustering.

1.4. Alcance

Se definen las siguientes cuestiones para el alcance de este proyecto:

- Se implementará el algoritmo de preprocesamiento y extracción de características de las imágenes, utilizando las ventajas y funcionalidades que provee la biblioteca TensorFlow para Android.
- La aplicación será desarrollada sobre la plataforma Android, contemplando en lo posible aquellas versiones que se encuentren instaladas en la mayoría de los dispositivos móviles actuales.
- La aplicación no realizará las tareas de administración de forma autónoma. Ésta asistirá al usuario brindando una estructura de directorios en la que las imágenes serán ordenadas por grado de similitud o contexto, para luego facilitar opciones de administración sobre los mismos.

1.5. Tecnologías utilizadas

1.5.1. Restricciones

Se realizó la documentación de la Especificación de Requisitos del Software (ERS) siguiendo el estándar IEEE 830-1998 (ver Anexo), y en la misma se encuentra la especificación completa de las tecnologías utilizadas. No obstante, se listan a continuación:

- **Sistema operativo**: Android
- **Versiones**: Android KitKat (4.4) en adelante
- **Lenguaje de programación**: JAVA 1.8.0
- **Paradigmas de programación**:
  - Orientado a objetos

    - Funcional
    - Imperativo
- **IDE**: Android Studio 3.0.1
- **Dependencias externas**:
    - TensorFlow Lite 0.1.7
    - Storage Chooser 2.0 ! [1]
    - Apache Commons[2]:
        - Maths 3.6.1
        - IO 2.4
    - Efficient Java Matrix Library[3] 0.33
    - Snatik Storage[4] 2.1.0
    - ZGallery[5] 0.3
    - NumberProgressBar[6] 1.4
- **Interfaz de usuario**: XML 1.1
- **Control de versiones**: Git v2.7.4

1.5.2. Evolución previsible

Se prevén las siguientes cuestiones tecnológicas a mejorar o integrar en el futuro del producto:

- Incorporar otras características que complementen la descripción de las imágenes para el procesamiento, afín de que ayuden a obtener mejores resultados de agrupamiento.
- Agilizar la reorganización de imágenes en los grupos obtenidos, agregando nuevas funcionalidades a la hora de tratar el resultado final las cuales mejoren la experiencia de usuario.
- Mejorar el proceso de la aplicación, con el objetivo de ahorrar espacio en disco mientras el usuario mantenga los resultados temporales sin confirmar (de momento, duplica las imágenes mientras no se confirmen o descarten los resultados temporales).
- Ofrecer al usuario la posibilidad de configurar ciertos parámetros que incidirán en el resultado del agrupamiento, según sus preferencias.
- Implementar un proceso de integración continua para mejorar los mecanismos de pruebas y despliegue de la aplicación.

[1] Repositorio oficial: `https://github.com/codekidX/storage-chooser`
[2] Repositorio oficial: `http://commons.apache.org/`
[3] Repositorio oficial: `http://ejml.org/wiki/index.php`
[4] Repositorio oficial: `https://github.com/sromku/android-storage`
[5] Repositorio oficial: `https://github.com/mzelzoghbi/ZGallery`
[6] Repositorio oficial: `https://github.com/daimajia/NumberProgressBar`

Capítulo 2

Fundamentos teóricos

Uno de los grandes descubrimientos que un hombre puede hacer, una de sus grandes sorpresas, es encontrar que puede hacer lo que temía que no podía hacer.

Henry Ford

Resumen: En este capítulo se presenta al sistema operativo Android, plataforma en la que se realizó el desarrollo y despliegue de la aplicación. También, se expondrán los fundamentos del aprendizaje profundo referido al enfoque utilizado en este proyecto respecto a visión computacional. Por último, se introducirán contenidos acerca de algoritmos de clustering, en especial conceptos que abarcan las características del algoritmo implementado para este trabajo.

2.1. Programación en dispositivos móviles

[DONE]Uno de los ámbitos multimedia con más crecimiento en los últimos años ha sido el de los dispositivos móviles. La llegada de los *smartphones* disparó en su momento la creación de aplicaciones móviles que aprovechan la capacidad multimedia de estos dispositivos. La aparición y el despegue poco después de las tabletas ha convertido el desarrollo de aplicaciones para dispositivos móviles en un pilar de la industria multimedia.

[DONE]Existen lenguajes o *frameworks* que nos permiten la creación de programas multidispositivo (se programa una vez y se ejecuta en cualquier dispositivo), pero para aplicaciones complejas que requieren un gran rendimiento, la programación suele hacerse en los lenguajes nativos de cada sistema operativo, con lo que debe reescribirse la aplicación para cada uno de ellos[15]. Como se ha dicho anteriormente, para la realización de este trabajo el desarrollo de la aplicación se hizo sobre el lenguaje nativo del sistema operativo Android.

2.1.1. Plataforma Android

[DONE]Es un sistema operativo diseñado para dispositivos móviles, desarrollado inicialmente por *Android Inc.*, una empresa comprada por Google en 2005. El lanzamiento se realizó el 5 de noviembre de 2007 y Google liberó la mayoría de código Android bajo licencia Apache, conocida por ser libre y de código abierto [18].

[DONE]Actualmente, se pueden encontrar más de 2.500.000 de aplicaciones desarrolladas para esta plataforma [9]. Además, cuenta con un entorno de desarrollo integrado oficial (IDE), denominado *Android Studio*, realizado por la empresa JetBrain. El mismo fue anunciado en mayo de 2013 y reemplazó a Eclipse como IDE oficial para el desarrollo de aplicaciones [12].

A continuación, se mencionan algunas de las características más destacables de la plataforma de desarrollo:

- Java es el lenguaje de programación en el que están escritas todas las aplicaciones Android nativas [24].
- Es Open Source, por lo que no es necesario pagar ninguna licencia para poder utilizar la plataforma.
- Existe una gran comunidad activa en comparación al desarrollo en otros sistemas operativos móviles, por lo que cuenta con mantenimiento y actualizaciones continuas.
- Las aplicaciones se proveen a través de los *Android Application Package* (APK), similar a los archivos con extensión .exe en Windows. Las mismas deben ser firmadas digitalmente antes de poder ser desplegadas pero no tienen ninguna restricción en comparación a las que tienen otros sistemas operativos, agilizando en términos de desarrollo y testing ya que una opción es firmar la aplicación en modo depuración[8].
- Provee de renderizado en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.

2.1.2. Entorno ejecución Android Runtime

[DONE]Android Runtime (ART) es el entorno actual de ejecución de aplicaciones utilizado por el sistema operativo móvil Android. ART reemplaza a Dalvik, que es la máquina virtual utilizada originalmente por Android, y lleva a cabo la transformación de la aplicación en instrucciones de máquina, que luego son ejecutadas por el entorno de ejecución nativo del dispositivo.

[DONE]A diferencia de Dalvik, que utiliza *just- in-time* (JIT) para compilar el código cada vez que se inicia una aplicación, ART introduce el uso de *ahead-of-time* (AOT), que crea un archivo de compilación posterior a la instalación de la aplicación. Este archivo es utilizado al abrir la aplicación, por lo que se evita que la aplicación se compile continuamente, cada vez que ésta es ejecutada. Al

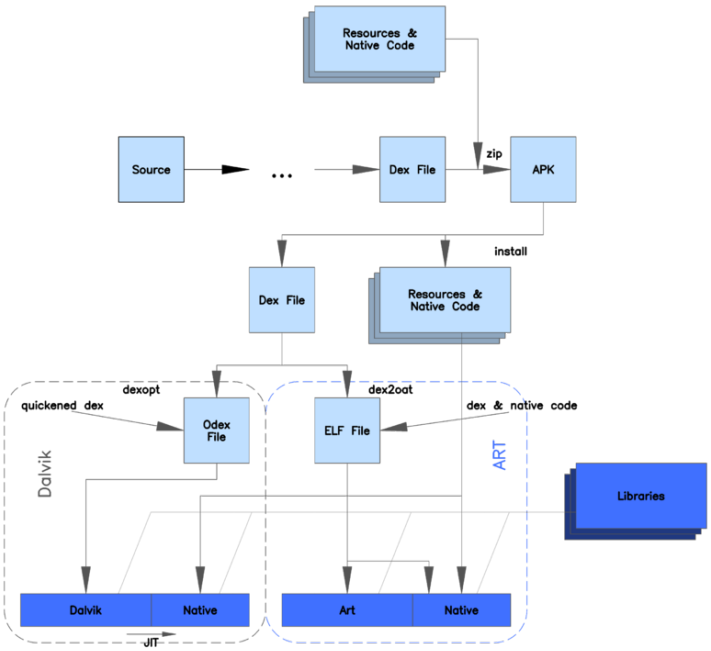


Figura 2.1: Compilación con Dalvik-VM y compilación con ART-VM. Recuperado de: `https://es.wikipedia.org/wiki/Android_Runtime`

reducir la cantidad global de compilaciones realizadas por cada aplicación, el uso del procesador del dispositivo móvil se reduce y aumenta la duración de la batería. Al mismo tiempo, ART trae mejoras en el rendimiento, tales como la recolección de basura, aplicaciones de depuración y perfilado[5].

[DONE]Para mantener la compatibilidad con versiones anteriores, ART utiliza el mismo código de bytes de entrada que Dalvik. En la Figura 2.1 se pueden apreciar las diferencias que existe entre la compilación que se hacía con la maquina virtual *Dalvik* y como se hace hoy en día con *Android Runtime*

2.1.3. [DONE]Componentes básicos de una aplicación Android

Existen cinco componentes con los que se puede crear una aplicación Android, y ellos son: Activities, Services, Intents, Content Provider y Broadcast Receivers[6]. Android facilita la creación de aplicaciones gracias al uso de un conjunto de componentes de software reutilizables. A continuación, se detallará que función cumple cada uno de ellos:

- **Activity:** componente principal de la interfaz gráfica de una aplicación. A cada Activity se le asigna una ventana en la cual se dibuja la interfaz de usuario, con la cual éste podrá interaccionar para realizar las diversas acciones que hayamos contemplado en la aplicación.

  Para construir la interfaz gráfica, existen componentes denominados Views

(vistas) con los que se dispone de numerosos controles básicos, como por ejemplo, botones, listas desplegables o cuadros de texto, pudiendo extender la funcionalidad de estos controles o crear otros personalizados.

Por lo general, una aplicación está formada por diferentes *activities*, que están más o menos relacionadas entre sí. Siempre existe la Actividad Principal, quien es la encargada de iniciar la aplicación. Para crear una nueva actividad, se debe crear una nueva clase la cual heredará de la clase Activity, definiendo su propia interfaz de usuario y las funcionalidades que esta aportará a la aplicación.

Por último, es importante mencionar que existen componentes que funcionan dentro del ámbito de una Activity, denominados *Fragments*, los cuales amplían y enriquecen las posibilidades de interacción con el usuario.

- **Service:** componente sin interfaz gráfica que se ejecuta en segundo plano. Son llamados a través de otro componente, como puede ser una Activity, y seguirán ejecutándose en segundo plano aunque la Activity haya finalizado o, incluso, aunque se haya salido de la aplicación. Un ejemplo es una aplicación para reproducir música, la cual sigue funcionando mientras el usuario navega por otras aplicaciones.

- **Intent:** Un intent es el elemento básico de comunicación entre los componentes que estamos describiendo, es decir, mediante este objeto se puede llamar a un Activity, iniciar un servicio, enviar un mensaje broadcast, incluso se podría iniciar otra aplicación si así se necesitara. El uso más importante de este componente es el de iniciar Activities, por lo que puede considerarse como la *unión* entre ellas.

  Las intent están formados por un paquete de información. Contienen detalle de interés para el componente que las recibe, como la acción que será ejecutada y los datos necesarios, más información de interés para el sistema Android, como la categoría del componente que maneja el intent y las intrucciones de cómo lanzar la Activity.

- **Content Provider:** es un objeto destinado a compartir datos entre aplicaciones. Dichos datos pueden ser almacenados en el sistema de archivos, en una base de datos o en cualquier otro lugar que sea accesible desde la aplicación.

  Un ejemplo de Content Provider es el que utiliza Android para gestionar la información de un contacto, mediante el cual cualquier otra aplicación podrá acceder a ellos haciendo una petición sobre la interfaz que gestiona dicha herramienta (*ContentResolver*)

- **Broadcast Receiver:** es un componente que detecta y reacciona frente a mensajes globales del sistema, como puede ser batería baja, SMS recibido, llamada perdida, etc. Además de esto, una aplicación puede iniciar un Broadcast Receiver (por ejemplo, para saber si se han descargado datos al dispositivo y poder ser usados por esa aplicación). Al igual que ocurre con los Services, un Broadcast Receiver tampoco muestra ninguna interfaz gráfica.

2.1.4. [DONE]Aplicaciones: actividades y ciclo de vida

En Android, el tiempo de vida de las aplicaciones está generalmente controlado por el sistema operativo y no por el usuario como es usual. Cada aplicación se ejecuta en un proceso y si el sistema operativo lo cree necesario lo elimina, como por ejemplo si necesita liberar memoria[23].

Como se dijo en la sección anterior, una *activity* es un componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar. A cada actividad se le asigna una ventana en la que puede "dibujar"su interfaz de usuario.

Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. Normalmente, una actividad en una aplicación se especifica como la "principal", lo que significa que se presentará al usuario cuando éste inicia la aplicación por primera vez. Cada una de ellas puede a su vez iniciar otra actividad para poder realizar diferentes acciones. Cada vez que se inicia una nueva, se detiene la actividad anterior, pero el sistema las conserva en una pila (la "pila de actividades"). Cuando se inicia una actividad nueva, se la incluye en la pila y capta el foco del usuario. La pila de actividades cumple con el mecanismo *LIFO*: el último en entrar es el primero en salir, por lo que, cuando el usuario termina de interactuar con la actividad actual y presiona el botón Atrás, se quita de la pila (se destruye) y se reanuda la actividad anterior.

Administrar el ciclo de vida de las actividades mediante la implementación de métodos *callbacks* es fundamental para desarrollar aplicaciones potentes y flexibles. El ciclo de vida de una actividad se ve directamente afectado por su asociación con otras actividades, con sus tareas y con la pila de actividades[4].

Una actividad puede existir básicamente en tres estados:

- **Reanudada:** La actividad se encuentra en el primer plano de la pantalla y tiene la atención del usuario. (A veces, este estado también se denomina en ejecución).
- **Pausada:** Otra actividad se encuentra en el primer plano y tiene la atención del usuario, pero ésta todavía está visible. Es decir, otra actividad está por encima de ésta y esa actividad es parcialmente transparente o no cubre toda la pantalla. Una actividad pausada está completamente "viva"(el objeto *activity* se conserva en la memoria, mantiene toda la información de estado y miembro y continúa anexado al administrador de ventanas), pero el sistema puede eliminarla en situaciones en que la memoria sea extremadamente baja.
- **Detenida:** La actividad está completamente opacada por otra (se encuentra en "segundo plano"). Una actividad detenida también permanece "viva"(el objeto se conserva en memoria, mantiene toda la información de estado y miembro, pero no está anexado al administrador de ventanas). Sin embargo, ya no está visible para el usuario y el sistema puede eliminarla.

Cuando una actividad entra y sale de los diferentes estados que se describieron arriba, esto se notifica a través de diferentes métodos *callback*. En conjunto, estos métodos definen el ciclo de vida completo de una actividad, como se puede observar en la Figura 2.2.

A continuación, se explicarán brevemente cada uno de estos métodos:

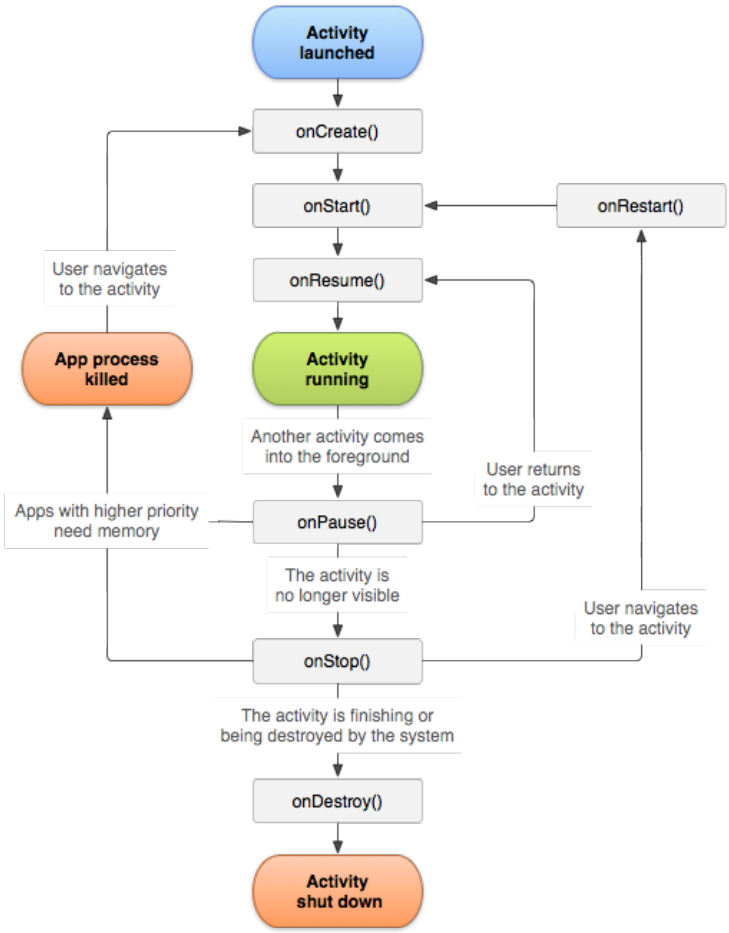


Figura 2.2: Ciclo de vida de una actividad. Recuperado de: `https://developer.android.com/guide/components/activities`

- **onCreate:** se llama cuando se crea la actividad por primera vez. El siguiente método es onStart.
- **onRestart:** se llama después de que ha parado la actividad y antes de volver a comenzarla. El próximo método es onStart.
- **onStart:** se llama cuando la actividad debe ser visible para el usuario. El próximo método será onResume u onStop.
- **onResume:** se llama antes de que la actividad comience a interactuar con el usuario. El método que le sigue es onPause.
- **onPause:** se llama cuando el sistema quiere ejecutar otra actividad como principal. El siguiente método será onResume u onStop.
- **onStop:** este método es llamdo cuando la actividad deja de estar visible para el usuario. El siguiente método es onStart u onDestroy.

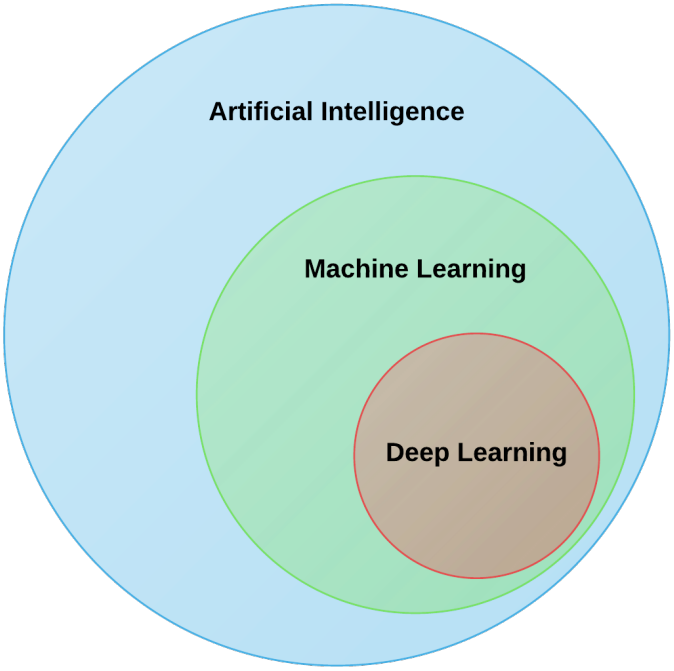


Figura 2.3: Inteligencia artificial, aprendizaje maquinal, y aprendizaje profundo

- **onDestroy:** se llama cuando se desea destruir la actividad. Es el último método que recibe la actividad.

2.2. [WIP]Fundamentos del aprendizaje profundo

En la siguiente sección, se realizará una introducción sobre conceptos relacionados a redes neuronales, aprendizaje maquinal y aprendizaje profundo, conceptos que pertenecen a la inteligencia artificial. El último de ellos, en particular, fue fundamental en el desarrollo de este proyecto, ya que a partir de uno de sus conceptos se logra la caracterización las imágenes en la aplicación.

2.2.1. Ingeligencia Artificial, aprendizaje maquinal y aprendizaje profundo

Para comenzar, se debe definir claramente de que se está hablando cuando se menciona a la inteligencia artificial. ¿De que se tratan los conceptos de inteligencia artificial (**AI**, del inglés *"Artificial Intelligence"*), aprendizaje maquinal (**ML**) y aprendizaje profundo (**DL**, del inglés *"Deep Learning"*) (ver Figura 2.3)? ¿Cómo están relacionados entre ellos?

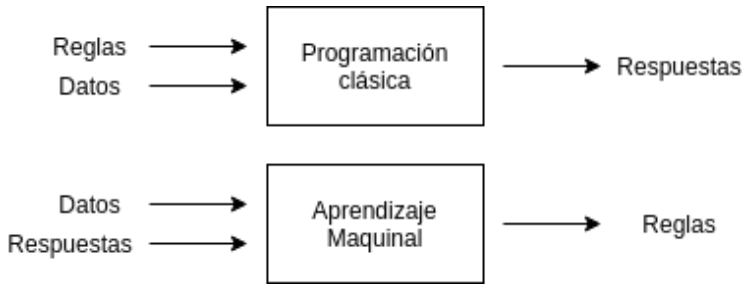


Figura 2.4: Aprendizaje maquinal: nuevo paradigma de programación

2.2.1.1. Ingeligencia Artificial

La inteligencia artificial nació en los años 1950s, cuando un conjunto de pioneros dedicados al campo de las ciencias de la computación se comenzaron a preguntar como las computadoras podrían ser hechas para "pensar". Una concisa definición del campo podría ser la que sigue: *el esfuerzo de automatizar tareas intelectuales, normalmente realizadas por humanos*. Como tal, la AI es un campo general que incluye al aprendizaje maquinal y al profundo, pero que también incluye muchos más conceptos que no involucran ningún *aprendizaje*. Los primeros juegos de ajedrez por computadora, por ejemplo, sólo envolvían un montón de reglas preestablecidas escritas por programadores con un alto nivel de conocimiento y de ayuda proveniente de expertos en la materia. Este concepto es conocido como "inteligencia artificial simbólica", y fue muy popular entre los años 50 y 80 gracias al boom de los sistemas expertos creados en esos años.[16]

Aunque la AI simbólica provee buenas resoluciones para problemas bien definidos, lógicos, tales como el ajedrez, estos no sirven para obtener buenas aproximaciones a problemas relacionados a la clasificación de imágenes, reconocimiento del habla y traducción de lenguaje, entre otros. Por lo que una nueva aproximación tomó el lugar de la AI simbólica: el aprendizaje maquinal (en ingles: *Machine Learning*).

2.2.1.2. Aprendizaje Maquinal

El aprendizaje maquinal surge a partir de esta pregunta: ¿puede una computadora ir mas allá "de lo que los humanos saben" con el objetivo de conseguir mejores resultados y aprender por ella misma como mejorar tareas concretas, sin que se le especifique cómo hacerlo?

Esta pregunta abrió la puerta a un nuevo paradigma de programación. En la programación clásica, como por ejemplo la inteligencia artificial simbólica, el humano escribe todas las reglas (a través de sus programas) y los datos se procesan acorde a lo que ellas dicen, y luego se obtiene una posible respuesta (ver Figura 2.4). En cambio, con el *machine learning*, los datos de entrada por parte de los humanos son tan bien conocidos como la respuesta esperada por ellos, por lo que verdaderamente importa son las reglas que se obtienen, ya que estas serán aplicadas a nuevos datos de entrada para producir respuestas originales.[16]

Un sistema realizado bajo el paradigma del aprendizaje maquinal es *entrenado* más que explícitamente programado. Es decir, se le presentan un montón de ejemplos relevantes a una tarea, y él por sus propios medios encontrará la estructura estadística más acorde para obtener la respuesta esperada a ellos. Por lo tanto, una vez que ya se encuentra bien entrenado, permitirá resolver tareas de ese tipo de manera automática.

Un ejemplo podría ser automatizar la tarea de etiquetar aquellas imágenes que se corresponden a unas vacaciones. Lo que se debe hacer es presentar al sistema de aprendizaje maquinal un montón de imágenes que ya fueron etiquetadas por humanos, y éste aprendería reglas estadísticas para asociar aquellas que se corresponden a este tipo de imágenes diferenciando de aquellas que no se corresponderían a este grupo.

La tarea descripta arriba es un claro ejemplo de un problema de **clasificación**, altamente relacionado a la problemática tratada en este proyecto donde el ML encuadra perfectamente como opción de resolución. Otros ejemplos de aplicación podrían ser problemas de Regresión, Aprendizaje no Supervisado, *Reinforcement Learning*, etc.[3]

2.2.1.3. Aprendizaje Profundo

El *deep learning* es un subcampo específico del aprendizaje maquinal: una nueva manera de aprender representación desde los datos poniendo énfasis en sucesivas *capas* desde las cuales se van obteniendo representaciones cada vez más significativas. Lo profundo (*deep*) en este subcampo del aprendizaje maquinal no hace referencia a ningún tipo de entendimiento profundo logrado por este enfoque, sino que está por la idea de sucesivas capas de representación por las que van pasando los datos. Modelos actuales envuelven entre diez o incluso miles de capaz sucesivas de representación y todas ellas aprenden de manera automática a partir de ser expuestas a los datos de entrada.

Estas capas de representación están conformadas por modelos llamados *redes neuronales*. Este término está referenciado en la neurobiología, pero aunque algunos de los conceptos centrales en el aprendizaje profundo fueran desarrollados en parte tomando como inspiración el entendimiento humano del cerebro, los modelos *deep learning* no son modelos del cerebro. No existe evidencia de que el cerebro humano implemente algo parecido a lo que se hace en esta ciencia de la computación.[16]

2.2.1.4. Redes neuronales

Las *Redes Neuronales Artificiales* (**RNA**) comprenden una rama antigua pero destacada, especialmente en la actualidad luego del surgimiento del aprendizaje produndo. Se entiende como **RNA** a un sistema con elementos que procesan información de cuyas interacciones locales depende su comportamiento global [**?**]. Este sistema trata de emular el comportamiento del cerebro humano, adquiriendo conocimiento de su entorno mediante un proceso de aprendizaje y almacenándolo para disponer de su uso [**?**].

Las **RNA**s son implementadas en computadoras, tanto en programas de software como en arquitecturas de hardware, por lo cual se utilizan para componer un sistema de aprendizaje maquinal.

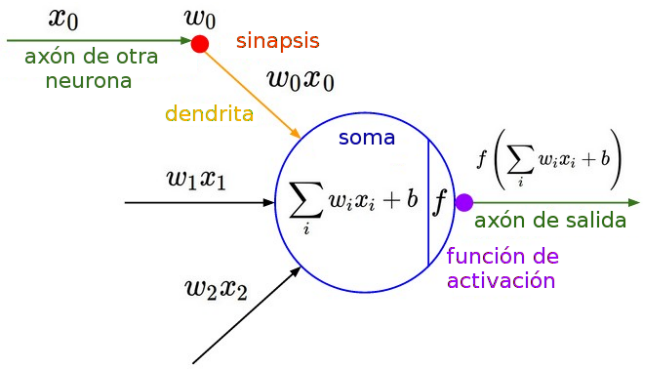


Figura 2.5: Modelo matemático de una neurona.

Para modelar el comportamiento de las neuronas, la idea es que las sinápsis que producen puedan controlarse mediante *pesos sinápticos* que definan una magnitud de la influencia que ejerce una con otra (y también dirección, al poder excitar o inhibir mediante pesos positivos o negativos, correspondientemente). Los pesos sinápticos w_0 multiplican las señales de entrada x_0 para ejercer una suma ponderada en el soma, y si el resultado supera un cierto umbral la neurona se "activa" enviando un estímulo hacia su salida. Dicha suma puede incorporar además un término de sesgo b que participa sin multiplicarse por la entrada [20]. En este modelo se considera que no interesa conocer el preciso tiempo en que se activa la neurona sino la frecuencia en que ocurre, por lo cual ello es representado mediante una *función de activación* [**?**]. En la Figura 2.5 se representa gráficamente el modelo explicado, el cual constituye lo que se denomina como "perceptrón simple'.

Para modelar una red neuronal artificial las neuronas se agrupan en capas, de forma tal que todas las unidades de una capa se conectan con todas las neuronas de sus capas próximas para formar una estructura interconectada. En la forma más simple, se tiene una capa de entrada que proyecta la señal entrante en una capa de salida. Cuando se incorporan capas intermedias (denominadas capas ocultas), la red adquiere más niveles de procesamiento entre su entrada y salida, y lo que se forma es un "perceptrón multicapa" (conocido en inglés como *Multi Layer Perceptron* o **MLP!**) [**?**]. En esta configuración, para producir la salida de la red se computan sucesivamente todas las activaciones una capa tras otra, donde puede notarse que una red con n capas equivale a tener n perceptrones simples en cascada, donde la salida del primero es la entrada del segundo y así sucesivamente. Además, cada capa puede tener diferente número de neuronas, e incluso distinta función de activación.

Siguiendo el modelo matemático de un perceptrón simple, los pesos sinápticos ahora pueden expresarse en conjunto de forma vectorial como matrices, así como también el sesgo se representa como un vector. Por lo tanto, dada un vector de entrada x, la suma ponderada efectuada en una capa se expresa como un producto entre la matriz de pesos sinápticos W y dicha entrada x, en la cual también se suma el vector de sesgo o *bias* b. Al resultado de esto se le aplica la función de activación, con lo cual se produce la salida final de la capa. En la Figura 2.6 se representa la arquitectura de un perceptrón multicapa, mostrando

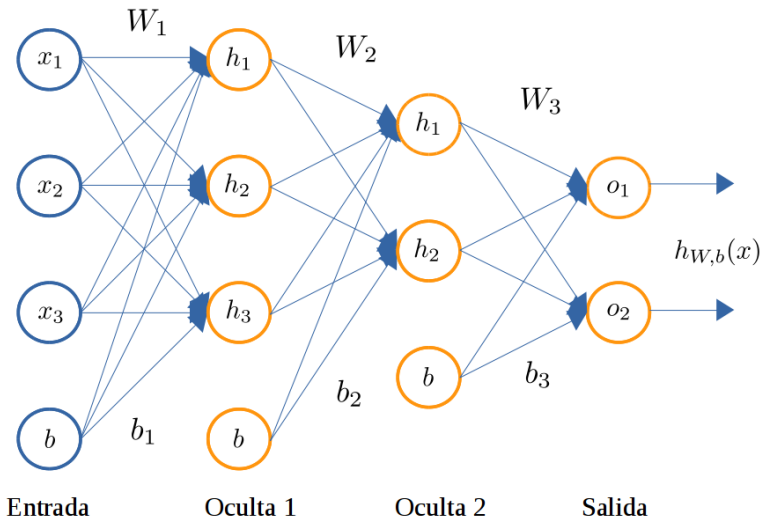


Figura 2.6: Arquitectura básica de un MLP con 4 capas.

las interacciones que tienen sus unidades desde la entrada hasta su salida [20].

2.3. [WIP]Aprendizaje profundo aplicado en problemas de visión computacional

A partir de la introducción sobre *computer vision* realizada en la sección Motivación de este documento, así como los antecedentes de sus aplicaciones exitosas mencionados allí también, se procede a profundizar acerca de las diferencias que ofrece la resolución de problemas de visión computacional aplicando técnicas de aprendizaje profundo, en comparación con las metodologías convencionales y utilizadas para el procesamiento digital de imágenes.

El proceso se conoce como **extracción de características**, y la elegancia del procesamiento de señales y procesamiento digital de imágenes es que los algoritmos han sido diseñados para adquirir y extraer características de las imágenes, tal como contornos, bordes, texturas, etc. Existen una gran variedad de técnicas y algoritmos, tanto libres como con licencia. Por ejemplo: SIFT y SURF los cuales se encuentran bajo patente, y ORB, FAST y BRISK los cuales son libres [27]. Todos ellos, describen de alguna manera a las imágenes y son usados como entrada para algoritmos de aprendizaje maquinal, por los cuales se hará la creación de funciones hipótesis (modelos), que luego pueden ser utilizados para resolver ejemplos diferentes a los que se usaron como entrenamiento.

Por otro lado, los algoritmos de visión computacional que aplican aprendizaje profundo, transforman las imágenes digitales en representaciones que son diferentes a la imagen original e incrementan la información acerca del resultado final. Se puede pensar a una red profunda como una operación de muchas etapas donde la información se va destilando, ya que los datos pasan a través de sucesivos filtros y sale *purificada* (ver Figura 2.7) [16].

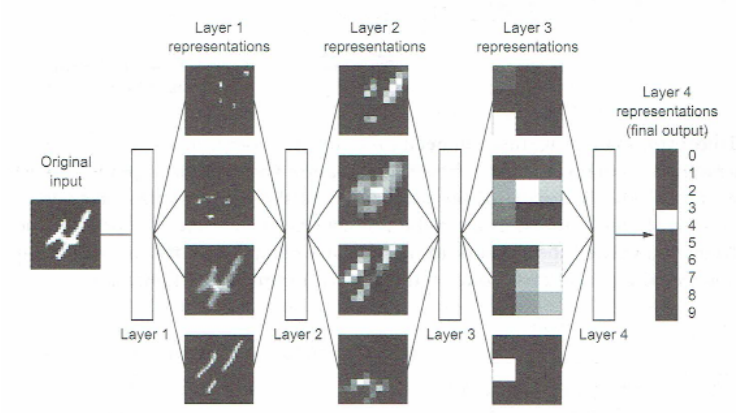


Figura 2.7: Representación de un modelo de aprendizaje profundo, para clasificación de dígitos manuscritos. Recuperado de [16]

Estas redes se denominan convolucionales (**CNN**) y actualmente son el estado del arte en lo que refiere a problemas de clasificación de imágenes [37]. CNN aplica una serie de filtros sobre los datos crudos de las imágenes (píxeles) para extraer y aprender características, las cuales luego el modelo usará para clasificar. Estas redes está compuestas por tres componentes:

- **Capas convolucionales:** son las encargadas de aplicar una serie de filtros en la imagen. Para cada subregión de la misma, se aplican un conjunto de operaciones matemáticas que producen un valor único por cada una de ellas.
- **Capas pooling:** obtienen los datos extraídos por las capas convolucionales con el objetivo de reducir la dimensión en orden de acortar los tiempos de proceso.
- **Capas densas (totalmente conectadas):** realizan la clasificación de las características extraídas por las capas convolucionales y reducidas por las capas de pooling. En una capa densa, cada nodo de la capa está conectado a cada nodo de la capa anterior.

Normalmente, una CNN se compone de una pila de módulos convolucionales que realizan la extracción de características. Cada módulo consta de una capa convolucional seguida de una capa pooling. El último módulo convolucional es seguido por una o más capas densas que realizan la clasificación. La última capa densa en una CNN contiene un solo nodo para cada clase de objetivo en el modelo (todas las clases posibles que el modelo puede predecir), la cual a través de una función de activación genera un valor entre 0-1 para cada nodo (la suma de todos estos valores es igual a 1), lo cual representa en que porcentaje se corresponde a cada clase (ver Figura 2.8) [37].

Hoy en día el *Deep Learning* sigue en constante crecimiento. Al ser un concepto relativamente nuevo, la gran cantidad de recursos que existen puede ser un punto abrumador, para aquellos que buscan entrar en el campo. Una buena

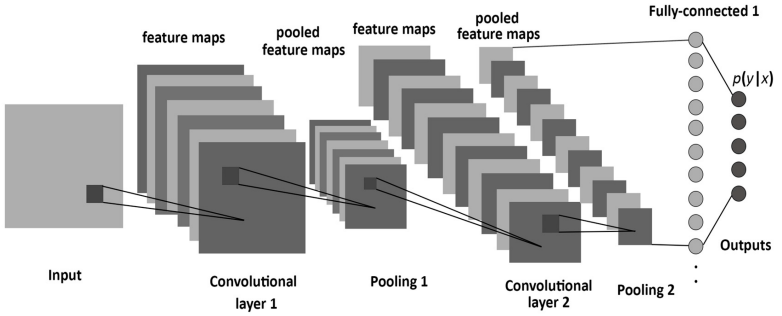


Figura 2.8: Red neuronal convolucional (CNN)

manera de mantenerse al día con las últimas tendencias es interactuar con la comunidad, y saber qué herramientas la industria brinda a desarrolladores para poder llevar adelante sus propios proyectos y desafíos [17].

A continuación, se presentará la biblioteca TensorFlow, la cual fue utilizada en esta tesis como herramienta principal para la descripción y extracción de características de las imágenes que se procesarán en los dispositivos móviles.

2.3.1. TensorFlow

Es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas. Fue desarrollada 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 usados por los humanos. Actualmente es utilizado tanto en la investigación como en los productos de Google. Fue originalmente desarrollado por el equipo de Google Brain para uso interno de la compañía antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015 [1]. TensorFlow es multiplataforma, funciona tanto en GPUs como CPUs -incluyendo plataformas móviles y embebidas- e incluso unidades de procesamiento tensorial (TPUs), que son un hardware especializado en realizar cálculos tensoriales.

Entre sus distintas soluciones se puede encontrar TensorFlow Lite, la cual es de gran utilidad a la hora de correr modelos de ML en dispositivos móviles o embebidos, sin la necesidad de contar con una conexión a Internet, ya que utiliza técnicas para lograr baja latencia, performance y modelos que ocupan poco almacenamiento. Uno de los modelos propuestos por la biblioteca fue utilizado para este proyecto final de carrera, y se denomina *MobileNets* [38].

2.3.1.1. MobileNets

Son una familia de modelos de redes neuronales profundas utilizadas en problemas de visión computacional, diseñados para maximizar la precisión de forma efectiva, teniendo en cuenta los recursos restringidos para una aplicación integrada en un dispositivo móvil. Son modelos pequeños, de baja latencia y baja potencia, parametrizados para satisfacer las limitaciones de recursos de una gran variedad de casos de uso. Se pueden utilizar para la clasificación, detección,

incrustación y segmentación de imágenes [25]. Las MobileNet están basadas en los modelos Inception V3, pero son más pequeñas y con una exactitud (*accuracy*) un poco más baja. Estos modelos pueden ser descargados y ajustados a proyectos de manera libre y gratuita, y existen diferentes tipos, variando sobretodo en la optimización que se necesite para cada caso de aplicación. En el siguiente capítulo se ahondará sobre el uso particular que tiene en este proyecto final de carrera.

2.4. [WIP] Análisis de agrupamiento

El análisis de agrupamiento (*clustering*) es la tarea de agrupar un conjunto de objetos de tal manera que los miembros del mismo grupo (llamado clúster) sean más similares, en algún sentido o bajo algún criterio. Es la tarea principal de la minería de datos exploratoria y es una técnica común en el análisis de datos estadísticos. Es utilizada en múltiples campos como el aprendizaje automático, la bioinformática, la compresión de datos y la computación gráfica [28].

El análisis de grupos no consiste en un algoritmo específico. Se puede hacer el agrupamiento utilizando varios de ellos, los cuales difieren significativamente en su idea de qué constituye un grupo y cómo encontrarlos eficientemente. Las ideas clásicas de grupo incluyen distancias pequeñas entre los miembros del mismo, áreas densas del espacio de datos, intervalos o distribuciones estadísticas particulares. El algoritmo apropiado y los valores de los parámetros (incluyendo valores como la función de distancia para utilizar, un umbral de densidad o el número de grupos esperado) depende del conjunto de datos que se analiza y el uso que se le dará a los resultados. El *clustering* como tal, no es una tarea automática, sino un proceso iterativo de minería de datos que implica prueba y fracaso. A menudo es necesario hacer un pre-procesamiento de los datos y un ajuste de los parámetros del modelo hasta que el resultado tenga las propiedades deseadas.

2.4.1. Agrupamiento basado en grafos

Para este proyecto, el algoritmo de agrupamiento utilizado pertenece a un modelo basado en grafos (en ingles: *graph clustering*). Los grafos son estructuras de datos en las cuales los nodos representan entidades, y las aristas representan relaciones (ver Figura 2.9). Por ejemplo, una persona podría construir un grafo sobre las conexiones que existen entre sus amigos en Facebook. Allí, cada nodo se correspondería con sus amigos, y las aristas representarían una relación de amistad entre dos de ellos.

Para el caso de aplicación abordado durante este proyecto, se pensó a cada imagen como una entidad y a cada arista como el grado de similitud o contexto que existe entre ellas. El algoritmo utilizado para llevar adelante esta tarea encuadra dentro de esta categoría y se denomina "Algoritmo de agrupamiento de Markov" (**MCL**: *Markov Cluster Algorithm*), el cual se introduce a continuación.

2.4.1.1. Algorítmo de agrupamiento de Markov

El algoritmo MCL es la abreviatura de *Markov Cluster Algorithm*, un algoritmo de clúster rápido y escalable para grafos (también conocido como redes)

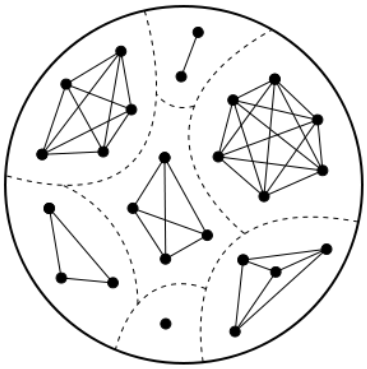


Figura 2.9: Agrupamiento basado en grafos

basado en la simulación del flujo (estocástico) en grafos [39]. Fue inventado por Stijn van Dongen en el Centro de Matemáticas e Informática (CWI) en Holanda. La tesis doctoral *Graph clustering by flow simulation* se centra en este algoritmo, siendo los temas principales la teoría matemática que lo sustenta, su posición en el análisis de clústeres y en la agrupación de grafos, las cuestiones relacionadas con la escalabilidad, la implementación, y los criterios de rendimiento para la agrupación de grafos en general. El trabajo para esta tesis se llevó a cabo bajo la supervisión de Jan van Eijck y Michiel Hazewinkel.

El algoritmo simula el flujo usando dos operaciones algebraicas simples en matrices. No requiere instrucciones de procedimiento para ensamblar, unir o dividir los grupos. La estructura se inicia a través de un proceso de flujo que se ve afectado de manera inherente por cualquier estructura de grupo presente [39].

La primera operación utilizada es la expansión, que coincide con la multiplicación normal de la matriz. La expansión modela la dispersión del flujo, convirtiéndose más homogénea. La segunda es la inflación, que es, matemáticamente hablando, una potencia de Hadamard seguida de una escala diagonal. La inflación modela la contracción del flujo, haciéndose más espesa en las regiones de mayor corriente y más delgada en las regiones de menor corriente. El proceso MCL hace que el flujo se extienda dentro de los cúmulos naturales y se evapore entre los diferentes cúmulos. En la Figura 2.10 se puede observar un ejemplo de 4 iteraciones de un proceso MCL.

Variando un solo parámetro, se pueden encontrar agrupaciones en diferentes escalas de granularidad. El número de grupos no puede y no necesita ser especificado de antemano, pero el algoritmo puede ser adaptado a diferentes contextos. La cuestión de "¿cuántos grupos se obtendrán?" no se trata de manera arbitraria, sino más bien mediante una fuerte lógica interna. La estructura de cada grupo deja sus marcas en el proceso de flujo simulado por el algoritmo, y los parámetros de flujo controlan la granularidad de la impresión del cluster.

El límite del proceso es en general extremadamente escaso. Esto da los medios para escalar el algoritmo drásticamente, lo que conduce a una complejidad que

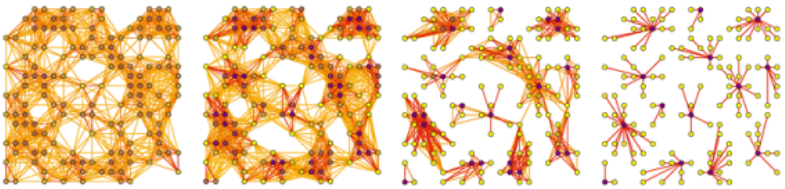


Figura 2.10: Representación gráfica del algoritmo de Markov

en el peor de los casos es del orden O(N k2), donde N es el número de nodos del grafo de entrada, y k es un umbral para el número de recursos asignados a cada nodo [39].

MCL ha sido aplicado en un gran conjunto de diferentes dominios, mayormente en problemas relacionados a la bioinformática y en el próximo capítulo se explicará en detalle su implementación y adaptación para la resolución de este trabajo.

Parte II

IMachineApp

Esta segunda parte de la tesis está dedicada a detallar todas las características de la aplicación lograda. Se describe el patrón de programación elegido para su implementación, justificando las elecciones de diseño realizadas, y las propiedades que la caracterizan como aplicación para sugerir una estructura de organización de imágenes según el grado de similitud o contexto, dando además la posibilidad de ser administradas por el usuario en caso de que así lo crea conveniente.

Capítulo 3

Descripción del sistema

La manera de empezar es dejar de hablar
y empezar a hacer.

Walt Disney

Resumen: En este capítulo se describe en primer lugar el diseño elegido para el sistema, explicando cada uno de sus componentes y justificando su composición. Luego, se explican todas las cuestiones técnicas relacionadas al motor de procesamiento, como por ejemplo la manera en que se caracterizan las imágenes, como se mide la similitud entre ellas, como se aplica el algoritmo MCL para obtener los grupos finales, etc. Por último, se presentan las herramientas con las que cuenta el usuario tanto para su experiencia durante la utilización de la aplicación como para administrar el resultado final.

El producto desarrollado en este proyecto se identifica como *IMachineApp*, donde la I al inicio de su nombre simboliza las imágenes a procesar, la palabra *Machine* hace referencia a la "Máquina" por la cual pasan las mismas para poder obtener el resultado. Por último, la sigla App deja en claro que el producto está destinado a ser desplegado en dispositivos móviles (ver Figura 3.1).

3.1. Diseño del sistema

En esta tesis se desarrolló un software que se conoce como aplicación móvil. La misma fue diseñada para ser ejecutada en teléfonos inteligentes y *tablets*, y permite al usuario elegir un directorio (o un conjunto de ellos) con imágenes y luego de un procesamiento por el que pasan las mismas, sugerirle una estructura organizacional, basándose en el rango de similitud o contexto que tengan entre ellas. A su vez, cuando finaliza el proceso, tiene la posibilidad de administrar el resultado, según su criterio para obtener así el resultado esperado, o bien deshacer todos los cambios y volver las imágenes hacia su estado original.

La aplicación, fue diseñada para que todo el proceso ocurra en 4 etapas:

Figura 3.1: Ícono de la aplicación desarrollada.

1. Elección del/los directorio/s a procesar: consiste en elegir algún directorio del dispositivo que cuente con imágenes a procesar, como por ejemplo donde quedan alojadas aquellas fotos que fueron tomadas por las cámaras del dispositivo.

2. Procesamiento de las imágenes a través del motor: esta etapa cuenta con una serie de pasos que se encarga de extraer las características de las imágenes, procesar las mismas afín de medir su similitud y poder obtener un conjunto de grupos donde en cada uno de ellos estarán las imágenes que más relación tengan entre sí. Esta etapa constituye el *core* de la aplicación.

3. Administración del resultado arrojado por el procesamiento: una vez que el motor terminó con el proceso de las imágenes y encontró una estructura de organización sugerida, en esta etapa el usuario podrá realizar tareas típicas como eliminar, mover o copiar imágenes entre directorios, o bien eliminar un directorio completo, moverlo hacia otra ubicación, etc.

4. Administración del resultado final: luego de que la etapa de administracón de las carpetas sugeridas por el proceso ha finalizado, el usuario tendrá la posibilidad de tomar diferentes decisiones respecto al resultado obtenido: descartar todos los cambios y volver las imágenes a su versión original, moverlas hacia una nueva carpeta creada por el programa, o copiarlas y mantenerla en dos lugares diferentes.

3.1.1. Metodología de trabajo

Para el cumplimiento óptimo de todos los objetivos del proyecto, se implementó el ciclo de vida del software siguiendo un modelo iterativo denominado Desarrollo Rápido de Aplicaciones (**RAD** del inglés, *Rapid Application Development*). Fue creado en 1980 por James Martin, siguiendo el modelo tradicional de desarrollo de software en cascada [13]. El modelo está compuesto por cuatro etapas, donde en la primera de ellas se realiza la definición de requerimientos, siguiendo por el diseño del sistema, el desarrollo del mismo y por último la etapa

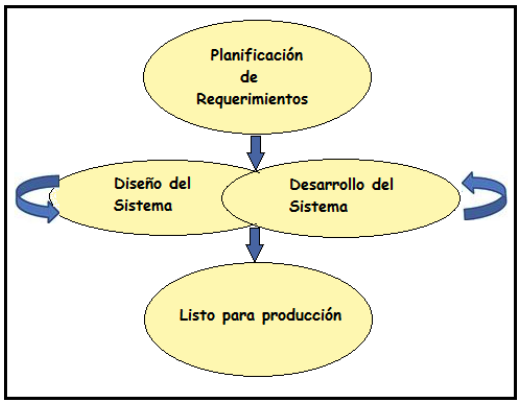


Figura 3.2: Modelo RAD (IMAGEN A CAMBIAR)

donde el producto está listo para ser enviado a producción. Si bien tiene un grado de similitud al modelo en cascada, en las etapas de diseño e implementación se trabaja de forma iterativa e incremental hasta cumplir con los criterios de aceptación dispuestos (ver Figura 3.2).

Se consideró a RAD el tipo de metodología que mejor se adaptaba al proyecto, en base a las diferentes etapas que lo conformaban y a la manera en que se trabajaría en cada una de ellas. Esta decisión se debe a que el proyecto tuvo una componente fundamental que fue la experiencia de usuario, y como se ha podido observar en publicaciones académicas que comparan diferentes metodologías para llevar adelante proyectos de software, se considera que aquellas de desarrollo lineal (como el modelo en cascada, por ejemplo) no son apropiadas [14]. Esto se debe a que los requerimientos de diseño se cumplieron de forma iterativa e incremental y no estrictamente en una única fase del proyecto, a fin de verdaderamente hacer lo que se solicitaba. En base a los estudios señalados anteriormente, los componentes relacionados a la experiencia de usuario se fueron incorporando y mejorando de forma iterativa gracias a la retroalimentación o *feedback* que se obtuvo por parte de los *stakeholders*.

Una vez definida la ERS (Especificación de los requerimientos del software), el producto se realizó en cuatro versiones de manera incremental, donde en cada una de ellas se entregaron avances, sujetos a lo especificado en el documento predefinido. Al término de cada versión, se obtuvo un prototipo del producto, para luego poder refinar y evaluar qué cosas eran necesarias cambiar, agregar o quitar, con el objetivo de que se cumplan exitosamente los criterios de aceptación de los entregables que conformaban el proyecto. Cada versión estaba compuesta por una parte de diseño, implementación y testeo, y son denominadas por la metodología como *time boxing* [13]. La finalización de todas las versiones conformó el producto buscado.

A continuación, se explicitan cada uno de los prototipos que conformaron la realización del proyecto, especificando lo que se realizó en cada una de ellos. Cabe aclarar una vez más que el producto cuenta con dos bloques principales a diseñar e implementar. El primero de ellos es la parte encargada de la extracción de características de las imágenes, procesamiento de las mismas y agrupamiento,

y el otro bloque está conformado por la aplicación, encargada de realizar la gestión y administración del resultado obtenido luego del procesamiento.

- **Prototipo 1:** Integración inicial de tecnologías y componentes de desarrollo.
  - Sistema Android: La tarea que debió cumplir fue mostrar al usuario un conjunto de directorios, pero siempre eligiendo uno por defecto. Este directorio ingresaba al motor de procesamiento. Después de que se ejecutaba el motor, debía mostrar por pantalla que el proceso había sido exitoso.
  - Motor de procesamiento: debía levantar la biblioteca TensorFlow y leer las imágenes ingresadas por la aplicación, devolviendo una simulación de grupos resultado para el conjunto de imágenes procesadas.
- **Prototipo 2:** Agregación de nuevas características al sistema e implementación de la metodología de clustering en el motor.
  - Sistema Android: se agregó a lo ya realizado la opción de elegir un directorio determinado. Luego del proceso, debía crear directorios en el dispositivo y copiar cada imagen al directorio correspondiente.
  - Motor de procesamiento: se aplicó las metodologías definidas para realizar el procesamiento, devolviendo al sistema a qué grupo debía ir cada una de las imágenes.
- **Prototipo 3:** Continuación del desarrollo de nuevas funcionalidades y presentación formal del motor de procesamiento.
  - Sistema Android: debía permitir la utilización de las diferentes tareas de administración una vez que el procesamiento fue realizado. Además, se agregó la opción de guardar el resultado final o eliminarlo y volver todo al estado original.
  - Motor de procesamiento: sufrió modificaciones a fin de lograr mejores resultados, además se trabajó en tareas de optimización.
- **Prototipo 4:** Finalización del desarrollo e integración final de todos los componentes.
  - Sistema Android: debía presentar todas las interfaces de usuario terminadas y listas para utilizar, el ícono que distingue a la aplicación y la aprobación de todas las pruebas de validación.
  - Motor de procesamiento: sufrió modificaciones a fin de lograr mejores resultados. Debió superar las pruebas presentadas en el protocolo de experimentación.

3.1.2. Patrón de programación seleccionado

Como se dijo en la Sección 2.1, la aplicación será desarrollada para el sistema operativo Android, de manera nativa, afín de aprovechar todas las ventajas que las librerías proveen para los desarrolladores. El framework Android, no recomienda ninguna forma específica de diseñar aplicaciones. Eso, de alguna

manera, hace a los desarrolladores más poderosos y vulnerables, al mismo tiempo [2].

Las aplicaciones pasan por muchos ciclos de vida donde se van agregando/extrayendo características, ocasionando estragos si la misma no se diseña adecuadamente con una "separación de preocupaciones". No es una buena práctica cargar de lógica a las actividades que se encargan de interactuar con el usuario, sino mas bien deberían ser clases por las cuales simplemente la información que interactúa entre el sistema y el usuario fluye, desligándose de cualquier tipo de lógica de negocio. Es por ello, que se decidió seguir el desarrollo de la aplicación bajo un patrón de diseño Modelo-Vista-Presentador (**MVP**).

3.1.2.1. Modelo-Vista-Presentador

Este patrón de diseño es un conjunto de pautas que, si se siguen de manera correcta, desacoplan el código para su reutilización y testeabilidad. Básicamente, divide los componentes de la aplicación basados en su función, denominado *separación de preocupaciones*, es decir, MVP divide a la aplicación en tres componentes básicos [2]:

1. **Modelo:** es el responsable de manejar todo lo que corresponda a la lógica de negocio de la aplicación, procesando la información y tomando decisiones.

2. **Vista:** es la responsable presentar las vistas al usuario e interactuar con el mismo.

3. **Presentador:** es un puente que conecta al modelo con la vista. Además, actúa como un instructor para la vista.

Definidos cada uno de los componentes, MVP establece algunas reglas básicas para ellos:

- La única responsabilidad de una Vista, es dibujar la interfaz de usuario según las instrucciones del Presentador. Es la parte "tonta" de la aplicación.

- La Vista delega todas las interacciones con el usuario hacia el Presentador.

- La Vista **nunca** se comunica con el Modelo directamente.

- El Presentador es el responsable de delegar los requerimientos de la vista hacia el Modelo, y instruir a la Vista con acciones para eventos específicos.

- El Modelo es el responsable de realizar toda la lógica de negocio requerida, como por ejemplo obtener los datos desde el servidor, la base de datos o el sistema de archivos.

A modo de resumen, cada Vista tendrá una relación 1 a 1 con un Presentador, el cual se encargará de llevar y traer toda la información, indicándole a la Vista que acción y en qué momento se debe ejecutar. Dicha comunicación se establece a través de Interfaces, las cuales servirán como *contratos* para dejar pre-establecidos que medios de comunicación existirán entre ellos. De manera

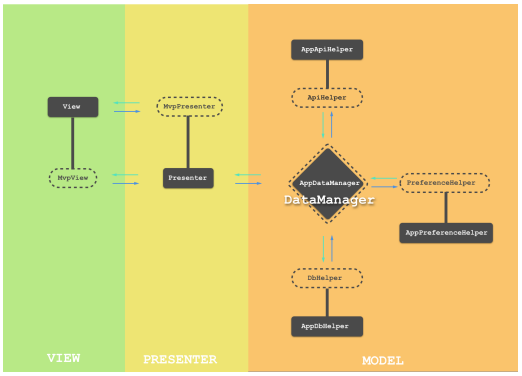


Figura 3.3: Patrón de diseño Modelo-Vista-Controlador

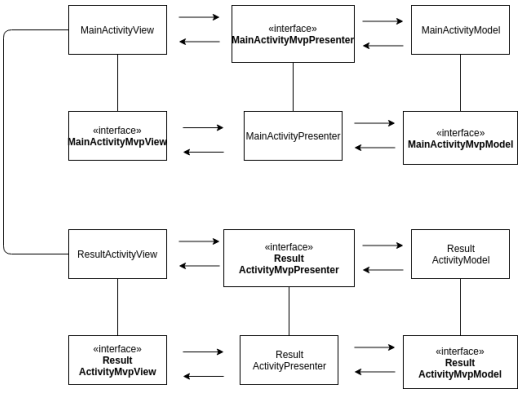


Figura 3.4: Diseño de la aplicación IMachineApp

similar y bajo el mismo criterio, el Presentador se comunicará con el Modelo a través de Interfaces, pero sin la necesidad de que la relación entre ellos sea estrictamente 1 a 1, ya que el modelo podría modularizarse tanto como se requiera. De esta manera, el Modelo se desentiende de los eventos y sólo es llamado cuando se lo requiere (ver Figura 3.3).

La aplicación fue pensada en dos bloques diferentes. El primero se corresponde a la parte principal, donde se elige el directorio a procesar y se lleva adelante toda la lógica para lograr la sugerencia de organización de las imágenes. El segundo, ya tiene más que ver con las opciones de administración sobre el resultado, con la posibilidad de guardarlo, de mover todas las imágenes hacia un nuevo directorio final o bien descartar todos los cambios. En la Figura 3.4 se puede ver en lineas generales el diseño de la aplicación.

Como conclusión y aprendizaje, la arquitectura es en lo primero que se debe trabajar para cualquier software. Una arquitectura cuidadosamente diseñada, minimizará muchas repeticiones en el futuro, a la vez que proporcionará la facilidad de escalabilidad. La mayor parte de proyectos se desarrollan hoy en día en equipo, por lo que la legibilidad y la modularidad del código deben considerarse como elementos sumamente importantes de la arquitectura. También, se de-

pende en gran medida de bibliotecas de terceros, cambiando entre alternativas, debido a casos de uso, errores o soporte.

3.2. Motor de procesamiento de las imágenes

En esta sección se detallará el paso a paso por el que transita el conjunto de imágenes para que se puedan obtener los grupos sugeridos por la aplicación, ordenándolas según el grado de similitud o contexto que tengan entre ellas.

El motor de procesamiento de las imágenes se llama **CIEngine**. El nombre está compuesto por las letras C, haciendo referencia a la palabra *Clustering* (agrupamiento en inglés), la letra I de *Images*, y por último la palabra *Engine*, que en inglés significa motor. El mismo fue pensado y realizado como un módulo aparte de la aplicación propiamente dicha, y agregado a través de un paquete. Esto tiene como beneficios que sea independiente, por lo que la mantenibilidad y testeo se realizan sin depender exclusivamente de la aplicación, además de que se puede pensar como trabajo a futuro en la adaptación a otros ambientes, como puede ser en un programa escritorio, un servicio alojado en la nube, una aplicación nativa para otros sistemas operativos utilizados en dispositivos móviles, etc.

En la Figura 3.5 se puede observar el camino que recorren las imágenes dentro del motor, desde la lectura de todas ellas hasta la conformación de cada uno de los grupos resultantes. Las tareas fueron divididas en etapas, afín de que el trabajo esté estructurado y sea más entendible, localizando en cada una de ellas diferentes técnicas, relacionadas entre sí. A continuación, se describe en detalle cada uno de los pasos.

3.2.1. Etapa 1: Pre-procesamiento y Extracción de Características

Como se dijo en el capítulo anterior, se utiliza la biblioteca TensorFlow, específicamente un modelo de red neuronal denominado MobileNet, para realizar la descripción y extracción de características de las imágenes. Se eligió este tipo de modelo ya que se encuentran preparados y optimizados para ser utilizados en dispositivos móviles, logrando buenos resultados en ambientes productivos ya que son hoy por hoy el estado del arte en lo que corresponde a problemas de clasificación de imágenes [37]. El modelo tiene un tamaño total de 16.9MB, ideal para el uso en una aplicación móvil, ya que se busca que la misma no ocupe tanto espacio, destacando la no necesidad de una conexión a Internet, haciendo todo el proceso *on premise*, es decir, explotando los recursos del dispositivo.

La red utilizada en este proyecto fue entrenada con 1001 grupos pertenecientes a la base de datos de imágenes *ImageNet*. Esta base pertenece a un proyecto que fue diseñado para el uso de sistemas relacionados al reconocimiento de objetos, ya que cuenta con más de 14 millones de URL's de imágenes que han sido anotadas a mano por personas para indicar qué son los objetos que se encuentran en cada una de las imágenes. La base de datos cuenta con más de 20.000 categorías, y si bien las anotaciones de las imágenes realizadas por los colaboradores está disponible de manera gratuita, las imágenes reales no son propiedad de *ImageNet* [34].

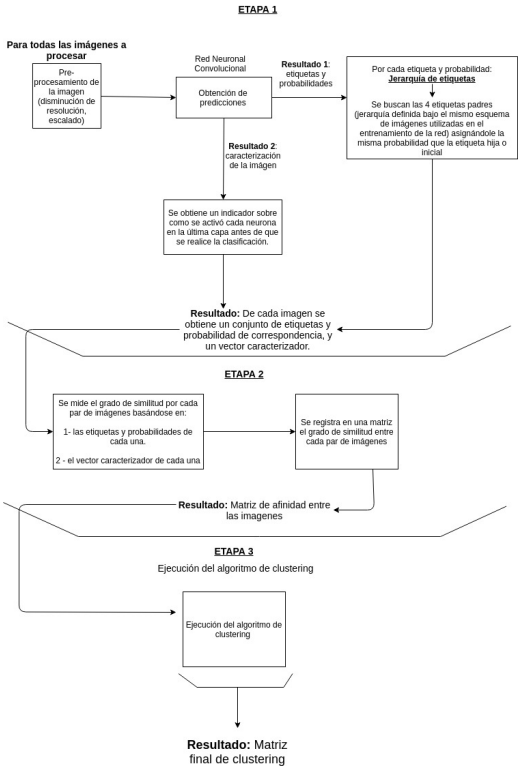


Figura 3.5: Diseño del motor de procesamiento

La forma de procesar que tiene la red es: a partir de una imagen de entrada, devolver el grado de similitud que tiene con los 1001 grupos con los que cuenta la red entrenada. Esto significa que por cada uno de los grupos, se obtendrá como salida un número entre 0 y 1, indicando el porcentaje de similitud. La suma de los 1001 resultados, será 1 (ver Figura 3.6.

A su vez, estos grupos están organizados acordes a la jerarquía de etiquetas o *tags* que tiene *WordNet*, una gran base de datos léxica para el idioma inglés, compuesta de sustantivos, verbos, adjetivos y adverbios que se agrupan en conjuntos, donde cada uno de ellos expresan un concepto distinto. Para el uso de *ImageNet*, sólo se utilizaron *tags* sustantivos, y los mismos están jerarquizados partiendo desde uno madre o genérico, bajando hacia aquellos que detallan en concreto el/los objetos que se encuentran en la imagen (ver Figura 3.7). Un punto importante es que esta base también es libre y gratuita, enfocada principalmente en el procesamiento automático del lenguaje natural y en aplicaciones de inteligencia artificial [10].

Dichas anotaciones, tienen relación con una base de datos de etiquetas o *tags* que fue organizada acorde a la jerarquía de palabras que se encuentran en la base de datos léxica WordNet. Cada una de las palabras que se encuentran allí están internacionalizadas por medio de relaciones conceptuales-semánticas y léxicas. Como sucede con ImageNet, esta base de datos también es libre y pública por lo que se encuentra disponible para su descarga.

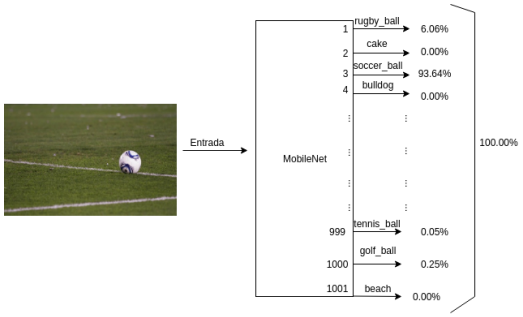


Figura 3.6: Ejemplo de aplicación de una MobileNet

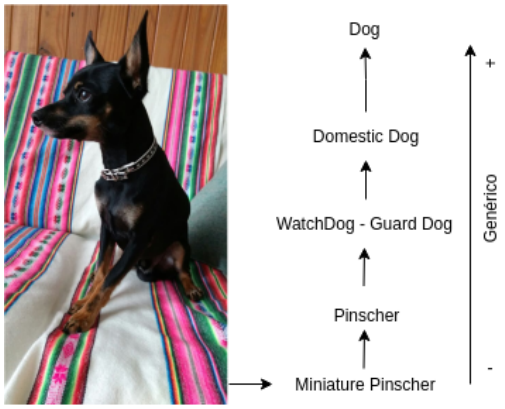


Figura 3.7: Ejemplo del uso de etiquetas - WordNet

3.2.1.1. Pre-procesamiento de las imágenes

Para que las imágenes puedan ser interpretadas por la red neuronal, deben pasar por un pre-procesamiento. Debido a una restricción que se tuvo cuando entrenaron los distintos modelos que componen el grupo MobileNet, la red que obtiene mejores resultados en términos de clasificación fue entrenada con imágenes cuyo tamaño fue de 224x224 píxeles, por lo tanto, para poder utilizarla, las imágenes de entrada al motor de procesamiento deben tener este tamaño también. Para lograr ello, las imágenes son cargadas en memoria a través de un objeto que provee Android denominado Bitmap, el cual permite realizar distintas operaciones sobre ellas [7]. Con el objetivo de reducir los recursos computacionales utilizados, las mismas son cargadas en memoria bajando su resolución a un tamaño en el cual su ancho y/o alto es de 224 píxeles, adaptando la otra medida afín de que la imágen siga siendo proporcional. Una vez realizado esto, luego si es estrictamente re-escalada al tamaño solicitado por la red. Entonces, una vez que ya se encuentra en memoria la imagen con una resolución de 224x224 píxeles lista para ser procesada por la CNN, se convierte en un objeto java denominado ByteBuffer [26], el cual es necesario en este formato ya que así lo solicita la red neuronal (ver Figura 3.8).

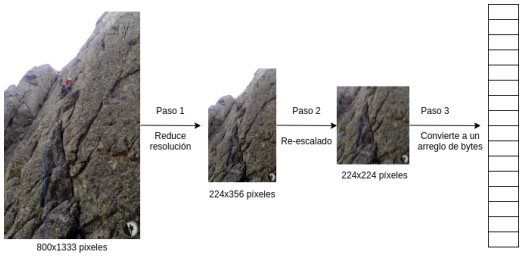


Figura 3.8: Pre-procesamiento de las imágenes

3.2.1.2. Extracción de características

Una vez que la imagen pasa por la red neuronal, se obtienen como predicciones dos resultados:

1. **Resultado 1: etiquetas y probabilidades**

   La primer salida que se obtiene de la red es una lista de las 5 mayores etiquetas o *tags* (en caso de que existan) cuya probabilidad de similitud sea mayor o igual a un umbral establecido. Una vez que se obtienen dichos *tags*, se busca por cada uno de ellas las 4 etiquetas "padres" (utilizando la jerarquía definida por *WordNet*) asignándoles la misma probabilidad que la etiqueta "hija" o inicial.

2. **Resultado 2: caracterización de la imagen**

   La segunda salida que se obtiene de la red es un indicador sobre cómo se activó cada neurona en la última capa antes de que se realice la clasificación. Esta última capa tiene como nombre *embeddings* y es utilizada en el proceso como una caracterización de la imagen por parte de la red. Los *embeddings* son importantes para el aprendizaje maquinal, ya que se convierten datos a una representación de características donde ciertas propiedades pueden ser representadas por nociones de distancia. Un modelo entrenado para clasificar imágenes puede permitir convertir una imagen en un vector de números, de manera que otra imagen similar tenga una distancia (por ejemplo, euclidiana) pequeña [36].

Como conclusión, se obtiene por cada una de las imágenes a procesar un conjunto de etiquetas y probabilidades de correspondencia, y un vector con valores numéricos que indica como la red neuronal caracterizó a la imagen. A continuación, se explicará la etapa 2, en donde se realiza la medición de similitudes entre las imágenes.

3.2.2. Medición de similitud entre las imágenes

Una vez que fue recolectada toda la información que se describió durante la sección anterior, se procede a medir el grado de similitud que existe entre cada par de imágenes basándose en las etiquetas y probabilidades que se obtuvo de cada una de ellas, como así también del vector de características provisto por la capa de *embeddings*. Al final de esta etapa, se obtendrá una matriz ponderada por dos matrices de afinidad distintas, que se describen a continuación:

1. **Matriz de afinidad gramatical:**

   Esta matriz indica la afinidad gramatical que existe entre cada par de imágenes. Por lo tanto, se obtendrá una matriz de tamaño NxN, siendo N el número de imágenes a procesar, y la posición *i,j* dentro de la matriz el grado de similitud gramatical que existe entre la imagen i y la imagen j.

   Para poder medir dicha similitud, lo que se hace en una primera instancia es obtener una especie de *diccionario* con todos los *tags* existentes, provistos por todas las imágenes incluidas en el proceso. Luego, por cada una de las imágenes, se confecciona un mapa que tiene como clave todos los *tags* provistos por el diccionario y como valor la probabilidad de similitud obtenida en la sección anterior, en caso de que exista ocurrencia, sino directamente se le asigna como valor un 0. De esta manera, se logra un vector normalizado por cada una de las imágenes, por lo que resta medir la similitud que existe entre cada par de ellas. Para obtener dicha medida, se utiliza el **coeficiente de correlación de Pearson**, un índice estadístico que puede utilizarse para medir el grado de correlación que tienen dos vectores [11]. El valor puede variar entre [-1,1], indicando el signo el sentido de la relación, siendo 1 una correlación perfecta. El criterio que se tomó en este proyecto es quedarse solamente con aquellas correlaciones que superan un umbral de 0.65, y en cualquier otro caso directamente se toma como inexistente una correlación entre ambas imágenes. De esta manera, se llega a la obtención de la matriz de afinidad gramatical entre las imágenes.

2. **Matriz de afinidad según embeddings:**

   De manera similar a como se obtuvo la matriz de afinidad gramatical, se procesan cada uno de los vectores que se obtuvieron por caracterizar a la matriz gracias a la capa de *embeddings*. De ella se obtiene por cada imagen un vector de 1024 números, el cual es normalizado y procesado de a pares a través del coeficiente de Pearson, con el objetivo de obtener una nueva matriz de NxN, similar a la obtenida anteriormente.

3.3. Arranca lo de LEA

Tal como se aclaró en la Sección 1.5.1, el código fuente de IMachineApp sigue un Diseño Orientado a Objetos (**DOO!**) el cual cuenta con los siguientes beneficios para desarrolladores [19]:

- *Modularidad*, al encapsular detalles de la implementación detrás de interfaces estables y sencillas de utilizar.
- *Reutilización*, definiendo componentes genéricos que pueden ser reaplicados para crear nuevas aplicaciones. Con ello se aprovecha el dominio de conocimiento y el esfuerzo previo de desarrolladores experimentados para evitar rehacer y revalidar soluciones ya existentes.
- *Extensibilidad*, al proveer métodos acoplables que permiten a las aplicaciones extender sus interfaces estables.

- *Inversión de control*, al invertir el flujo de ejecución del programa dejando que alguna entidad lleve a cabo las acciones de control que se requieran, en el orden necesario y para todos los sucesos que deban ocurrir, en lugar de hacerlo imperativamente mediante llamadas a procedimientos o funciones.

La última propiedad mencionada refiere a que por medio del framework se explicita una solicitud concreta (e.g. entrenar una red neuronal sobre un conjunto de datos), y el mismo decide la secuencia de acciones necesarias para atenderla. En cuanto a las demás, el resto del presente capítulo muestra evidencia de su cumplimiento mediante las características que se van presentando.

Como la mayoría de los proyectos desarrollados en Python, Learninspy está compuesto por paquetes que agrupan módulos en común, y en cada uno de estos últimos se reúnen las clases (en términos de DOO) que tengan mayor relación. A continuación se describe brevemente la lógica de cada módulo y paquete, aunque para mayor detalle se debe consultar el manual de referencia [1]:

- **Core**: Como bien dice el nombre, es el módulo principal o el núcleo del framework. El mismo contiene clases relacionadas con la construcción de redes neuronales profundas, desde la configuración de los parámetros usados hasta la optimización del desempeño en las tareas asignadas. Se detallan entonces cada uno de los submódulos que lo componen:
  - *Activations*: En el mismo se implementan las funciones de activación (con su correspondiente derivada analítica) que se podrán utilizar en las capas de una red neuronal.
  - *Autoencoder*: Se extienden las clases desarrolladas en el submódulo *model*, mediante herencia de métodos y atributos, para implementar autocodificadores y su uso en forma apilada.
  - *Loss*: Provee dos funciones de error, las cuales son utilizadas en base a la tarea designada a una red neuronal: clasificación, mediante la función de *Entropía Cruzada*, y regresión, con la función de *Error Cuadrático Medio*.
  - *Model*: Es el submódulo principal de **core** ya que contiene las clases referidas directamente a redes neuronales, el diseño de sus capas y la configuración de los parámetros que manejan.
  - *Neurons*: Este submódulo contiene una clase para manejar las matrices de pesos sinápticos y los vectores de sesgo que componen las capas de una red neuronal. Dichos arreglos se implementan mediante NumPy para que se almacenen de forma local (i.e. se alojan por completo en un mismo nodo físico de ejecución), aunque se tiene pensado extender esta clase para que puedan manejarse en forma distribuida.
  - *Optimization*: Implementa los algoritmos y funcionalidades de optimización que se utilizan para mejorar iterativamente el modelado de las redes neuronales. Los algoritmos presentes han sido explicados en la Sección **??** (salvo Adagrad, ya que en su lugar se implementó Adadelta) y para la implementación fueron adaptados desde el desarrollo hecho en Climin [**?**], el cual es un framework de optimización pensado para escenarios de aprendizaje maquinal.

[1] Documentación de Learninspy: `http://learninspy.readthedocs.io/`

  - *Search*: Realizado para abarcar algoritmos de búsqueda que optimicen los parámetros de un modelo en particular. El único algoritmo desarrollado en esta versión es el de búsqueda aleatoria, que fue detallado anteriormente en la Sección **??**.
  - *Stops*: Recopila distintos criterios de corte para frenar la optimización de las redes en base a una condición determinada. Al igual que el submódulo *optimization*, está basado en el trabajo hecho en Climin.

- **Utils**: Este módulo abarca todas las utilidades desarrolladas para posibilitar tanto la construcción de redes neuronales como el funcionamiento total del framework. El mismo dispone de los siguientes submódulos:

  - *Checks*: Contiene funcionalidades para comprobar la correcta implementación de las funciones de activación y de error, basándose en las instrucciones de un tutorial de aprendizaje profundo [**?**].
  - *Data*: Es el submódulo principal de **utils**, ya que posee clases útiles para construir los conjuntos de datos que alimentan las redes neuronales, y también funcionalidades para muestrearlos, etiquetarlos, partirlos y normalizarlos.
  - *Evaluation*: Se proporcionan clases para evaluar el desempeño de las redes neuronales en tareas de clasificación y regresión, mediante diversas métricas que fueron explicadas en la Sección **??**.
  - *Feature*: Se implementan funcionalidades referidas a extracción de características o tipos de pre-procesamiento sobre los datos que alimentan una red neuronal. Un ejemplo de ello es el análisis de componentes principales o PCA (mencionado en la Sección **??**), que fue implementado siguiendo tutoriales clásicos de deep learning [**?**] [**?**].
  - *Fileio*: Submódulo con funciones para realizar manejo de archivos y la configuración del logger de Learninspy.
  - *Plots*: Reúne todas las funcionalidades referidas a gráficas y visualizaciones (como el ajuste de una red durante el entrenamiento).

Además, a la misma altura que estos dos módulos, existe un script denominado *context* en donde se configura e instancia el contexto de Spark a utilizar en el framework. En la Sección **??** del siguiente capítulo se mencionan las configuraciones que contempla este script referidas al rendimiento de Spark.

En base al diseño planteado, se identifican dos perfiles de acceso al framework: a) de usuario, en el cual mediante conocimientos básicos de Python se puede utilizar la plataforma y añadirle algunas funcionalidades, siguiendo un paradigma de programación imperativa; b) de desarrollador, que requiere usar un paradigma de programación orientado a objetos y funcional, para la comprensión total del código mediante conocimientos de Python y Spark.

La estructura presentada se considera exhaustiva en cuanto a contenido del framework, por lo cual cualquier desarrollador que quiera modificar o agregar componentes al mismo debería poder valerse de los módulos disponibles en la arquitectura comprendida.

3.4. Características

A continuación, se detallan las particularidades de Learninspy que lo hacen un framework útil para construir redes neuronales con aprendizaje profundo sobre un conjunto de datos y en forma distribuida:

- *Diseño que permite extender funcionalidades con pocas modificaciones y sin romper el funcionamiento de otros módulos.*

  Esto se relaciona con la propiedad de *extensibilidad* en un framework, mencionada en la anterior Sección 3.1. Por ejemplo, para agregar una función de activación y su derivada analítica, basta con incorporar sus definiciones en el submódulo **core.activations** y, mediante una etiqueta apropiada, adjuntarlas a los diccionarios de Python (que se encuentran al final del módulo) para utilizarlas en el framework a través del mismo. Se puede realizar un tratamiento similar para agregar tanto funciones de error como algoritmos de optimización y sus criterios de corte.

- *El paradigma orientado a objetos permite aprovechar la naturaleza del diseño de las redes neuronales, para así expresar las relaciones existentes entre las entidades manejadas.*

  Por ejemplo, la composición de una red neuronal por capas de neuronas, donde cada una de ellas tiene asociado una matriz de pesos sinápticos y un vector de sesgo, y también el hecho de que un autocodificador sea un tipo especial de red neuronal por lo que tiene una relación de herencia de métodos y atributos.

- *Mínima cantidad de dependencias en el sistema.*

  A partir del énfasis que se tuvo en esta propiedad para el diseño, no se requiere instalar más que Spark (y Java por ello) y parte del ecosistema de SciPy (que es casi un estándar en las típicas aplicaciones de Python).

- *Optimización de un modelo mediante entrenamiento de réplicas en forma concurrente y distribuida.*

  Es la característica principal de optimización que se diseñó para el sistema, y es explicada detalladamente en la siguiente Sección 3.5.

- *Los resultados del modelado pueden reproducirse de forma determinística*

  A diferencia de otras herramientas que distribuyen las operaciones de modelado, en Learninspy es posible replicar de forma exacta un experimento con una configuración dada. Esto se debe a que internamente se gestiona en forma determinística el semillero que alimenta el generador de números aleatorios, los cuales son requeridos por varios algoritmos que intervienen en el modelado (e.g. inicializador de pesos sinápticos, DropOut, etc).

- *Soporte para procesar conjuntos de datos en forma local y distribuida*

  Mediante las funciones y clases del módulo *utils.data* presentado, se brindan funcionalidades para el tratamiento de datos tanto en forma local como distribuida (utilizando RDDs de Spark para este último caso).

- *Soporte para cargar y guardar modelos entrenados.*

  El trabajo de optimización de los modelos se puede realizar de forma diferida, ya que los mismos se pueden guardar y volver a cargar en formato binario. Esto tiene gran utilidad sobre todo cuando se someten a aprendizaje no supervisado, el cual puede realizarse en muchas pasadas hasta aplicarse el ajuste fino.

Como se puede ver, algunas características están referidas al diseño del software en general y otras son más específicas del procesamiento distribuido que involucra. Por lo tanto, se describe a continuación en qué formas se logran integrar estas características en el framework.

3.4.1. Explotación del cómputo distribuido

Como ya se dijo anteriormente en otras secciones, las aplicaciones que suelen tratarse con aprendizaje profundo están relacionadas con datos de gran dimensión, y por ello las herramientas que realizan dicho tratamiento requieren una ventaja computacional para resultar útiles en ello. Las formas en que Learninspy aprovecha el procesamiento distribuido de Spark son las siguientes:

1. **Preparar conjuntos de datos:** El framework provee una abstracción para manejar conjuntos de datos, la cual incluye el etiquetado de los patrones por clases, la normalización y escalado de los datos, el muestreo balanceado por clases, etc. Para grandes volúmenes de datos se provee una interfaz adecuada para los RDDs de Spark, con lo cual el pre-procesamiento puede realizarse en forma distribuida.

2. **Optimizar modelos en forma paralelizada:** Siendo quizás el valor principal del procesamiento distribuido en el framework, esta característica se basa en que, por cada iteración del ajuste de una red neuronal, el modelado se realice mediante instancias replicadas que se entrenan de forma independiente y luego convergen en un modelo único, reuniendo así las actualizaciones que adquirió cada instancia por separado.

3. **Ahorrar costos de comunicación, transfiriendo conjuntos de datos a los nodos por única vez (broadcasting):** Como se explicó en la Sección **??**, la funcionalidad de Broadcast que provee Spark permite que una variable muy utilizada se pueda enviar a los nodos computacionales una sóla vez (siempre que la usen únicamente en modo lectura). Esto resulta útil y eficiente con los conjuntos de datos empleados en el ajuste de las redes neuronales, el cual se hace iterativamente y de otra forma requeriría establecer una comunicación con los nodos activos por cada iteración.

4. **Configurar infraestructura fácilmente:** Mediante simples configuraciones en las variables de entorno, se puede conectar el framework forma sencilla a una estructura computacional definida con Spark (lo cual se menciona más adelante en la Sección **??**).

Para entender cómo se obtiene la segunda característica mencionada, que se considera la más importante y tiene cierta complejidad, la siguiente sección detalla la forma en que se implementa en Learninspy.

3.5. Entrenamiento distribuido

El procedimiento para minimizar la función de costo sobre una red neuronal es una característica clave de Learninspy, ya que es una de las formas principales en que se aprovecha el cómputo distribuido en el framework. Dado que los algoritmos de optimización utilizados para realizar ello son iterativos, la paralelización propuesta busca incorporar los beneficios de la concurrencia para sacar mayor provecho al proceso en cada una de sus iteraciones.

La idea no es nueva ya que es implementada en diversos esquemas como los explicados en la Sección **??**. Se basa en que el proceso de optimización de las redes neuronales se puede paralelizar de forma tal que se obtenga una mejoría en duración y hasta resultados respecto al procedimiento convencional sin concurrencia. Para ello se tiene que, por cada iteración del proceso, un modelo base es copiado a cada nodo computacional para que cada una de estas copias o réplicas se entrene de forma independiente sobre algún subconjunto muestreado del conjunto original de datos. El hecho de optimizar en cada iteración con un subconjunto de datos (conocidos como *mini-batch*) en lugar del conjunto completo permite acelerar el proceso y está demostrado en varios estudios que aún así obtiene buenos resultados, como fue explicado en la Sección **??**. Es preciso aclarar que dichos subconjuntos son obtenidos de un muestreo aleatorio sin reemplazos sobre el conjunto de entrenamiento, utilizando la función *sample* de la librería *random* que ofrece la versión usada de Python.

La cantidad de modelos replicados a entrenar en paralelo es configurable: para un mejor desempeño en términos de recursos, debe ser la cantidad de nodos/núcleos disponibles, pero también puede ser menor o mayor para tener otro impacto en los resultados. Una vez entrenadas las réplicas, se procede a mezclar los modelos de forma que converjan los aportes de la optimización en un único modelo. Para ello se emplea una "función de consenso" que toma los parámetros de cada modelo y los pondera en base al resultado de evaluación sobre los respectivos subconjuntos de datos que utilizaron.

En el Algoritmo 1 se esquematiza el procedimiento general que sigue el entrenamiento de una red neuronal en Learninspy. Notar que el mismo se estructura como una tarea MapReduce, ya que de esa forma es implementado mediante las primitivas de ese tipo que provee el motor Spark. Mediante la función *merge* se realiza el proceso de mezclado de modelos mediante una función de consenso, lo cual se explica en detalle a continuación.

3.5.1. Funciones de consenso

Una vez entrenados todos los modelos replicados de forma concurrente, se deben mezclar en uno solo tratando de reunir las contribuciones de cada uno al ajuste del modelo deseado. Para ello, se puede caracterizar a cada modelo optimizado por su desempeño o *scoring* s_i que es obtenido de dos formas posibles: por una métrica aplicada en su evaluación (e.g. *accuracy* de clasificación, o R^2 de regresión), o bien por el valor resultante en la función de costo definida. El valor escogido para caracterizar cada modelo puede utilizarse como parte de una ponderación realizada sobre todos los modelos durante la mezcla, la cual consiste simplemente en una suma de los parámetros W y b de cada capa, por cada uno de los modelos correspondientemente. Para ello se propone usar una

Algorithm 1 Entrenamiento distribuido en Learninspy

Require: Modelo actual $h_{W,b}$.

function TRAIN(Γ, μ, ρ) ▷ Parámetros: Conjunto de entrenamiento Γ; tamaño de mini-batch μ; cantidad de modelos concurrentes o "paralelismo"ρ

— MAP —

$H_{W,b} = copy\_model(h_{W,b}, \rho)$ ▷ Realizar ρ copias de $h_{W,b}$ sobre los nodos disponibles

for $H^{(i)}_{W,b} \forall i \in \{1, \ldots, \rho\}$ **do** ▷ Bucle de ejecución concurrente

$\Gamma_\mu = sample(\Gamma, \mu)$ ▷ Muestreo de μ ejemplos sobre el conjunto Γ

$s_i = minimize(H^{(i)}_{W,b}, \Gamma_\mu)$ ▷ Optimización de modelo réplica

end for

— REDUCE —

$h_{W,b} = merge(H_{W,b}, s)$ ▷ Mezcla de modelos con función de consenso

$results = evaluate(h_{W,b}, \Gamma)$ ▷ Evaluación sobre el conjunto de datos

return $h_{W,b}, results$

end function

función de consenso que, en base a una ponderación establecida, logre reunir las contribuciones de los modelos para obtener un único modelo representativo. Esta mezcla consiste en una suma de los parámetros mencionados estableciendo pesos en base a una ponderación elegida, y esa suma a su vez es escalada por la sumatoria de los pesos obtenidos de la siguiente forma:

$$f(l_{W,b}, w) = \sum_i \frac{w_i l_i}{\sum w_i} \tag{3.1}$$

Si el denominador es muy cercano a 0, el mismo se reemplaza por una constante $\epsilon = 1e-3$ para evitar divisiones por 0.

Por defecto, se incluyen tres tipos de ponderación: a) constante, con los mismos pesos valiendo 1 para todos los modelos (resultando una media aritmética de cada parámetro), b) lineal, donde se utiliza en forma directa el valor de s_i, c) logarítmica, de forma que la ponderación no tenga gran variación sobre valores altos de s_i (muy buen valor en la evaluación, o bien pésimo costo de la red):

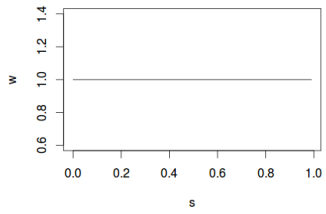
$$w_i = 1, \qquad \forall i \in \{1, \ldots, \rho\} \tag{3.2a}$$

$$w_i = s_i, \qquad \forall i \in \{1, \ldots, \rho\} \tag{3.2b}$$

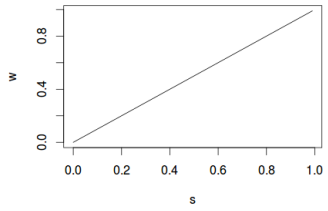
$$w_i = 1 + \ln(\text{máx}(s_i, \epsilon)), \qquad \forall i \in \{1, \ldots, \rho\}, \qquad \epsilon = 1e-3 \tag{3.2c}$$

Notar que para la ponderación logarítmica, si el dominio es menor o muy cercano a 0 se reemplaza por una constante $\epsilon = 1e-3$ para evitar conflictos con el dominio de la función logaritmo.

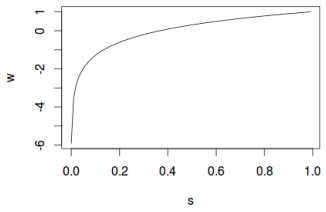
En la Figura 3.9 se representan gráficamente las funciones mencionadas, para un dominio definido en los valores del *scoring*. Para utilizar una función de consenso en particular, se debe configurar tanto la función como el *scoring* que utiliza mediante los parámetros de optimización que se definen para el modelado. Para ello, se debe instanciar un objeto OptimizerParameters del módulo



(a) Sin ponderación (constante)



(b) Ponderación lineal



(c) Ponderación logarítmica

Figura 3.9: Función que describe los pesos w que ponderan a cada modelo réplica en base a su valor s, suponiendo un dominio (0, 1] para dicho valor.

core.optimization indicando dichos parámetros en sus argumentos (ver detalles de uso en el manual de referencia).

3.5.2. Criterios de corte

En cualquier aplicación de aprendizaje maquinal, por lo general no se ejecuta la optimización de un modelo hasta obtener un desempeño deseado ya que puede ser que no se alcance dicho objetivo por la configuración establecida. Es por ello que, tal como se introdujo en la Sección **??**, resulta conveniente establecer ciertas heurísticas para monitorear la convergencia del modelo en su optimización. Un *criterio de corte* es una función que utiliza información de la optimización de un modelo durante dicho proceso (e.g. scoring sobre el conjunto de validación, costo actual, cantidad de iteraciones realizadas) y, en base a una regla establecida, determina si debe frenarse o no. Las reglas más comunes son:

- *Máximo de iteraciones:* Se detiene la optimización luego de que el número de iteraciones sobre los datos exceda un valor máximo establecido.
- *Alcanzar un valor mínimo deseado:* Se establece una tolerancia para un determinado valor de información en la optimización (como el scoring o el costo), con lo cual el proceso se frena cuando el valor se alcanza o supera.
- *Tiempo transcurrido:* Luego de exceder un intervalo de tiempo máximo fijado (en segundos, por lo general), se detiene el proceso de optimización.

Cada una de estas reglas devuelve Verdadero si el proceso debe frenar o Falso en caso contrario. Dichos resultados booleanos pueden combinarse con operadores lógicos AND y OR para armar reglas más expresivas que configuren la optimización de una forma más específica. Por ejemplo, se puede establecer que el proceso tenga un máximo de 100 iteraciones o bien que frene si se llega a un scoring de 0.9 estableciendo un OR entre las dos primeras reglas explicadas.

La implementación de ello se encuentra en el módulo **core.stops**, y se llevó a cabo mediante una adaptación del código provisto por Climin [2] con lo cual se puede extender el framework con nuevas reglas y criterios de corte para la optimización de los modelos.

Notar que para implementar este esquema de criterios en Learninspy, se deben especificar dos tipos de configuración: una para la optimización de los modelos réplicas a entrenar en paralelo sobre un batch de datos (llamada "optimización local"), y otra para la optimización en general del modelo final respecto a un conjunto de validación (denominada "optimización global"). En la Sección **??** del capítulo siguiente, un experimento de validación está dedicado a mostrar de forma empírica la relación entre ambos tipos de optimización.

3.5.3. Esquemas similares

En términos de comparación respecto a los esquemas mencionados en la Sección **??**, se identifcan las siguientes ventajas del esquema propuesto en este trabajo:

- **Simplicidad**: Gracias a las primitivas que provee Spark, implementar el esquema es sencillo y requiere pocas líneas de código para lograr que la optimización sea concurrente y además escale en recursos.
- **Convergencia**: Dado que se sincronizan las actualizaciones en cada iteración mediante el mezclado, se mitiga el riesgo de divergencia en la optimización que padecen tanto Downpour SGD como HOGWILD!, convergiendo a una solución comparablemente óptima con la desarrollada por el SGD sin paralelizar.
- **Elección del algoritmo de optimización**: Ya que el esquema es independiente del algoritmo utilizado para optimizar un modelo, se pueden implementar diversos tipos de algoritmos iterativos que estén basados en gradiente como los mencionados en la Sección **??**. Los mismos se pueden desarrollar en el módulo *core.optimization* del framework, donde actualmente se proveen dos algoritmos para optimizar redes neuronales.
- **Reproducibilidad de resultados**: En H2O, una plataforma que utiliza HOGWILD! para optimizar redes neuronales profundas, se debe ejecutar el entrenamiento con un único hilo de ejecución para obtener resultados replicables debido a las limitaciones del esquema. En Learninspy dicha reproducibilidad se logra independientemente del paralelismo empleado (por lo que se mencionó antes en la Sección 3.4), lo cual se ve como una ventaja muy importante a la hora de experimentar.

[2] Repositorio de código: `https://github.com/BRML/climin`

- **Personalización**: El mezclado de modelos no necesariamente se debe hacer promediando las contribuciones (como sucede en Iterative MapReduce y HOGWILD!), sino que se puede diseñar la función de consenso que decide cómo ponderar las mismas e incorporarla fácilmente en el framework. Por defecto se incluyen las tres funciones explicadas anteriormente.

De los tres algoritmos tratados en la comparación, se considera que el propuesto en este trabajo se asemeja mayormente al denominado Iterative MapReduce, ya que ambos incoporan la metodología de una tarea MapReduce en cada iteración de la optimización en un modelo. No obstante, en Learninspy se decidió implementar un esquema propio para definir el entrenamiento distribuido de una forma que, al igual que otras características de este framework, sea flexible y entensible respecto a las funcionalidades involucradas, asegurando además la propiedad de escalabilidad buscada.

Bibliografía

Cuando bebas agua, recuerda la fuente.
Proverbio chino

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] J. Ali. Essential Guide For Designing Your Android App Architecture. `https://blog.mindorks.com/essential-guide-for-designing-your-android-app-architecture-mvp-part-1-74efaf1cda40`. Accedido: 02/04/2018.

[3] E. Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[4] Android. Actividades. `https://developer.android.com/guide/components/activities`, 2018. Online, Accedido: 26/10/2018.

[5] Android. ART and Dalvik. `https://source.android.com/devices/tech/dalvik`, 2018. Online, Accedido: 14/05/2018.

[6] Android. Aspectos fundamentales de la aplicacion. `https://developer.android.com/guide/components/fundamentals`, 2018. Online, Accedido: 30/03/2018.

[7] Android. Bitmap. `https://developer.android.com/reference/android/graphics/Bitmap`, 2018. Online, Accedido: 13/04/2018.

[8] Android. Firmar tu aplicacion. `https://developer.android.com/studio/publish/app-signing`, 2018. Online, Accedido: 18/03/2018.

[9] AppBrain. Number of Android Apps on Google Play. `https://www.appbrain.com/stats/number-of-android-apps`. Accedido: 15/10/2018.

[10] M. Barning. WordNet and ImageNet. `https://www.web3.lu/wordnet-imagenet/`. Accedido: 13/12/2017.

[11] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[12] A. Bermudez. Android Studio 2.0 ya es oficial. `https://andro4all.com/2016/04/android-studio-2-0-novedades-caracteristicas`, 2016. Online, Accedido: 26/10/2017.

[13] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope. Rapid application development (rad): an empirical review. *European Journal of Information Systems*, 8(3):211–223, 1999.

[14] R. N. Burns and A. R. Dennis. Selecting the appropriate application development methodology. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, 17(1):19–23, 1985.

[15] C. Casado. Alternativas en la programacion para dispositivos moviles. `http://multimedia.uoc.edu/blogs/fem/es/alternativas-en-la-programacion-para-dispositivos-moviles/`, 2017. Online, Accedido: 30/09/2018.

[16] F. Chollet. *Deep learning with python*. Manning Publications Co., 2017.

[17] D. Clark. Top 16 Open Source Deep Learning Libraries and Platforms. `https://www.kdnuggets.com/2018/04/top-16-open-source-deep-learning-libraries.html`, 2018. Online, Accedido: 28/08/2018.

[18] J. DiMarzio and A. Android. Programmers guide, 2008.

[19] M. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.

[20] L. Ferrado. *Implementacion de un framework para la construccion de redes neuronales con aprendizaje profundo. Caso de aplicacion: clasificacion de seniales cerebrales*. Degree dissertation, Universidad Nacional del Litoral, 2016.

[21] R. C. González and R. E.Woods. *Digital Image Processing, 3rd. ed.* Prentice-Hall, 2008.

[22] V. Gonzalez. A brief history of machine learning. `http://www.synergicpartners.com/en/espanol-una-breve-historia-del-machine-learning`, 2018. Online, Accedido: 15/09/2018.

[23] C. Haseman. *Android Essentials*. Apress, 2009.

[24] O. Hosting. Ventajas y desventajas de las aplicaciones moviles nativas. `https://okhosting.com/blog/ventajas-desventajas-de-las-aplicaciones-moviles-nativas`, 2017. Online, Accedido: 22/01/2018.

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

[26] Java. Bytebuffer. `https://developer.android.com/reference/java/nio/ByteBuffer`, 2018. Online, Accedido: 13/04/2018.

[27] E. Karami, S. Prasad, and M. Shehata. Image matching using sift, surf, brief and orb: performance comparison for distorted images. *arXiv preprint arXiv:1710.02726*, 2017.

[28] J. Kogan. *Introduction to clustering large and high-dimensional data*. Cambridge University Press, 2007.

[29] Q. V. Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[30] M. Lewontin. How google photos uses machine learning to create customized albums. `https://www.csmonitor.com/Technology/2016/0324/How-Google-Photos-uses-machine-learning-to-create-customized-albums/`. Accedido: 02/10/2018.

[31] S. Lohr. Ibm is counting on its bet on watson, and paying big money for it. `https://www.nytimes.com/2016/10/17/technology/ibm-is-counting-on-its-bet-on-watson-and-paying-big-money-for-it.html`, 2016. Online, Accedido: 01/11/2018.

[32] M. Malhotra. How artificial intelligence (ai) is transforming mobile technology? `https://hackernoon.com/how-artificial-intelligence-ai-is-transforming-mobile-technology-9f810d734d35`, 2018. Online, Accedido: 13/08/2018.

[33] L. redaccion. Android vs iOS: quien gana la batalla. `https://www.lanacion.com.ar/1900814-android-vs-ios-quien-gana-la-batalla`. Accedido: 07/04/2017.

[34] Standford Vision Lab, Standord University. ImageNet. `http://image-net.org/about-overview`. Accedido: 12/12/2017.

[35] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[36] TensorFlow. Embeddings. `https://www.tensorflow.org/guide/embedding`, 2018. Online, Accedido: 23/04/2018.

[37] TensorFlow. Intro to Convolutional Neural Networks. `https://www.tensorflow.org/tutorials/estimators/cnn`, 2018. Online, Accedido: 15/02/2018.

[38] TensorFlow. Introduction to TensorFlow Lite. `https://www.tensorflow.org/lite/overview`, 2018. Online, Accedido: 11/06/2018.

[39] S. M. Van Dongen. *Graph clustering by flow simulation*. PhD thesis, 2000.

Lista de acrónimos

Parte III

Anexos