UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Interpretación del lenguaje de señas utilizando redes neuronales

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Carlos Alberto Monares Zabaleta

DIRECTORA DE TESIS

M. I. Norma Elva Chávez Rodríguez

Ciudad Universitaria, Cd. Mx., 2017

Índice

CAPÍTULO 1
INTRODUCCIÓN

1.1.- Objetivo

El presente trabajo tiene como finalidad la creación de una herramienta que permita facilitar la comunicación entre la población en general y las personas que padecen de pérdida de audición y por tanto utilizan algún lenguaje de señas como principal medio de comunicación.

Para lograr este fin se propone la creación de una herramienta que sea capaz de capturar las imágenes correspondientes a algunos de los signos utilizados en el lenguaje de señas americano (ASL por sus siglas en ingles), analizarlas y generar como resultado a que letra o número corresponden.

En los últimos años los dispositivos móviles han experimentado una gran adopción por parte de la población en general a tal grado que hoy en día es difícil encontrar alguna persona que no cuente con este tipo de aparato.

Por ello, la herramienta propuesta en el presente trabajo consistirá en una aplicación móvil que hará uso de la cámara del dispositivo para capturar las imágenes que requieren ser analizadas.

El proceso de análisis de las imágenes obtenidas se realizará utilizando una red neuronal ya que este tipo de sistemas han demostrado buenos resultados en tareas relacionadas con la clasificación de imágenes.

El proceso de entrenamiento de la red neuronal se llevará a cabo utilizando una técnica denominada transferencia de aprendizaje la cual permite reducir de manera considerable los requerimientos de hardware, datos y tiempo que este tipo de sistemas normalmente demandan.

1.2.- Descripción breve de los capítulos

El capítulo 1 brinda una introducción a la problemática que se aborda en el presente trabajo así como una descripción general de la solución propuesta.

El capítulo 2 describe la problemática que se pretende resolver, indicando las razones por las cuales se requiere una solución además se presenta el objetivo general y los objetivos particulares de la solución propuesta.

El capítulo 3 proporciona el marco teórico que se requiere para desarrollar cada una de las partes que integran la solución propuesta.

El capítulo 4 muestra la implementación de cada una de las partes que conforman la solución propuesta.

El capítulo 5 menciona las pruebas realizadas a la herramienta generada para determinar su desempeño en la resolución de la problemática planteada.

El capítulo 6 proporciona las conclusiones a las que se llegó después de realizar el presente trabajo así como algunas recomendaciones para mejorar la herramienta generada.

El capítulo 7 describe las limitaciones de la herramienta generada así como las condiciones bajo las cuales los resultados producidos pueden experimentar variaciones.

El capítulo 8 menciona las fuentes y referencias bibliográficas consultadas durante la creación del presente trabajo.

CAPÍTULO 2

PLANTEAMIENTO DEL PROBLEMA

2.1. Problemática

De acuerdo con la organización mundial de la salud (WHO por sus siglas en ingles) en el mundo existen alrededor de 360 millones de personas que padecen de pérdida de audición incapacitante de los cuales 32 millones son niños.

La pérdida de audición incapacitante se refiere a una pérdida de audición mayor a 40 decibeles (dB) en adultos y una perdida mayor a 30 decibeles (dB) en niños.

Aproximadamente un tercio de las personas mayores de 65 años padecen de pérdida de audición incapacitante. Además, 1100 millones de personas jóvenes (entre 12 y 35 años de edad) están en riesgo de sufrir pérdida de audición debido a exposición al ruido en ambientes recreativos [3].

La organización mundial de la salud estima que la pérdida de audición no atendida deriva en una pérdida anual global de 750 mil millones de dólares. Esto incluye costos del sector salud, costos de apoyo educativo, pérdida de productividad y costos sociales.

Uno de los principales problemas derivados de la pérdida de audición es la dificultad que experimenta el individuo para comunicarse. El desarrollo del lenguaje hablado usualmente se retrasa en niños con problemas de audición que no son atendidos oportunamente además puede tener un efecto adverso considerable en el desempeño escolar.

Los niños que padecen de pérdida de audición incapacitante usualmente tienen una mayor probabilidad de fracaso escolar.

Aproximadamente 70 millones de personas que padecen de pérdida de audición incapacitante utilizan el lenguaje de señas como su lengua materna. Cada país tiene uno o a veces dos o más lenguajes de señas, aunque diferentes lenguajes de señas pueden compartir las mismas raíces lingüísticas.

El lenguaje de señas no es un lenguaje internacional pero existen características universales en los lenguajes de señas. Esto hace posible que usuarios de distintos lenguajes de señas puedan entenderse entre sí de una manera más rápida que los usuarios de lenguajes hablados [4].

Al igual que los lenguajes hablados, los lenguajes de señas tienen una estructura que puede ser dividida en segmentos más pequeños como frases, signos u otras unidades más pequeñas. Los lenguajes de señas son usados de acuerdo con ciertas reglas gramaticales.

Las personas que utilizan un lenguaje de señas enfrentan grandes obstáculos para comunicarse con las personas que utilizan lenguajes hablados ya que la mayoría de estas desconoce el significado de los signos utilizados en los lenguajes de señas, ante esta situación los usuarios de un lenguaje de señas se ven en la necesidad de recurrir a otros medios de comunicación como el escrito, sin embargo, mantener una conversación de esta manera resulta poco práctico.

Por lo anterior se requiere de un sistema que sea capaz de interpretar los signos utilizados en el lenguaje de señas de modo que sea más fácil para las personas que sufren de pérdida de audición incapacitante el comunicarse con el resto de las personas sin necesidad de recurrir a otros métodos poco prácticos y que al mismo tiempo no requiera de algún equipo especializado adicional que pudiera resultar costoso o difícil de adquirir.

2.2. Objetivo general

El objetivo general es la creación de un sistema que permita interpretar los signos utilizados en un lenguaje de señas para facilitar la comunicación de las personas que padecen de pérdida de audición incapacitante.

Para que el sistema sea de utilidad para el público en general este deberá contar con las siguientes características:

- ✓ Accesible, para que cualquier persona pueda hacer uso de él sin necesidad de adquirir algún dispositivo especializado.
- ✓ Portable, de manera que el sistema pueda ser utilizado sin depender de alguna ubicación geográfica en específico.
- ✓ Económico, el uso del sistema no debe generar gastos significativos adicionales para el usuario.
- ✓ Fácil de utilizar, de modo que cualquier persona sea capaz de aprovecharlo sin necesidad de contar con ningún tipo de estudio o entrenamiento previo.
- ✓ Eficiente, el sistema deberá entregar los resultados esperados en un tiempo razonable.

Para cumplir con estos objetivos se propone la creación de una aplicación para dispositivos móviles la cual será capaz de interpretar los signos comúnmente utilizados en el lenguaje de señas de manera que el usuario final sea capaz de reconocer su significado sin necesidad de contar con ningún tipo de entrenamiento previo.

La aplicación creada estará disponible para el sistema operativo Android ya que es el que cuenta con un mayor número de usuarios a nivel global además de que los dispositivos móviles que utilizan este sistema operativo se encuentran disponibles en un amplio rango de precios lo cual facilita su adquisición en caso de el usuario que desee utilizar la aplicación no cuente con uno.

Para realizar el procesamiento de las imágenes obtenidas la aplicación hará uso de una red neuronal profunda la cual será implementada utilizando la plataforma TensorFlow desarrollada por Google para lograr que el entrenamiento de la misma sea lo más eficiente posible.

El procesamiento de las imágenes se realizará utilizando los recursos físicos del dispositivo móvil de modo que no sea necesario utilizar un servidor remoto el cual necesariamente tendría que ser contactado por medio de una conexión a internet lo cual generaría gastos adicionales al usuario.

Además de lo anterior el presente trabajo servirá en general para mostrar la manera de integrar un sistema de inteligencia artificial, en este caso una red neuronal, con una aplicación móvil y utilizarlo para realizar el análisis de los datos empleando únicamente los recursos de hardware del dispositivo.

2.3. Objetivos particulares

Los objetivos particulares de esta tesis son los siguientes:

I. Creación de una aplicación móvil nativa para el sistema operativo Android utilizando el lenguaje de programación Java.

II. Programación del proceso que permita utilizar la cámara del dispositivo móvil para capturar imágenes.

III. Programación del módulo que permita a la aplicación móvil identificar y clasificar las imágenes obtenidas, utilizando para ello una red neuronal profunda implementada con la plataforma TensorFlow.

IV. Hacer uso de la técnica denominada transferencia de aprendizaje para facilitar el entrenamiento de la red neuronal reduciendo con ello la cantidad de datos, tiempo y recursos de hardware requeridos.

V. Programación de la interfaz gráfica de la aplicación de modo que esta sea fácil de utilizar para el usuario final.

VI. Evaluación de la red neuronal empleada para determinar la efectividad con la que realiza la clasificación de las imágenes.

VII. Identificar las limitaciones del sistema y proponer la manera de superarlas.

CAPÍTULO 3

MARCO TEÓRICO

3.1 Dispositivos móviles

Un dispositivo móvil es una computadora que posee unas dimensiones menores a las de una computadora tradicional, esto permite que puedan ser transportados con facilidad de modo que pueden ser utilizados para tareas en las que se requiere de movilidad, por ejemplo, el captar y procesar imágenes en exteriores, acceder a internet desde un vehículo en movimiento, encontrar la ruta más eficiente por medio de GPS, etc. [6]

Los dispositivos móviles en general cuentan con varias características en común, siendo las más relevantes:

- ✓ Pantalla táctil.
- ✓ Sistema operativo especializado.
- ✓ Conexión a internet mediante Wi-Fi.
- ✓ Conexión con otros dispositivos mediante Bluetooth.
- ✓ Cámara.
- ✓ Batería recargable.
- ✓ GPS.

En la actualidad existen varios tipos de dispositivos móviles orientados a satisfacer distintos tipos de necesidades, entre los más comúnmente utilizados se encuentran las tabletas, teléfonos inteligentes (Smartphone) y relojes inteligentes (Smartwatch).

3.1.1 Tabletas

Una tableta es una computadora portátil de propósito general, cuenta con una pantalla táctil y batería recargable, normalmente están equipadas con varios sensores como cámaras, micrófono, acelerómetro, etc. Usualmente son más grandes que un teléfono inteligente con pantallas de 18 cm o mayores (medidos diagonalmente).

Las tabletas cuentan con varias ventajas como lo es la portabilidad, sin embargo, la mayoría también cuenta con varias limitaciones principalmente de procesamiento

por lo que no son capaces de realizar tareas que demanden demasiados recursos de hardware, además, su reducido tamaño complica su uso para ciertas tareas, es por ello que en general las tabletas se utilizan como complemento a una computadora tradicional y no como un reemplazo.

Por lo general las tabletas utilizan el mismo sistema operativo que los teléfonos inteligentes siendo los más comunes iOS y Android, aunque existen otros menos utilizados como Windows y Ubuntu.

En la actualidad los principales productores de tabletas son Apple, que posee 24.7% del mercado, Samsung, con 15.1% del mercado y Amazon, con 9.7%. [8]

El mercado de tabletas ha presentado una tendencia a la baja durante los últimos años debido a varios factores como la proliferación de teléfonos inteligentes con pantallas más grandes, la falta de productos innovadores y el hecho de que el usuario tiene pocas razones para renovar su equipo. [10]

3.1.2 Teléfonos inteligentes

Un teléfono inteligente es un dispositivo que combina características de un teléfono móvil, tal como la capacidad de recibir y realizar llamadas, con las de una computadora personal, tal como el uso de un sistema operativo especializado y la capacidad de ejecutar programas normalmente conocidos como Apps.

Entre las características más comunes de los teléfonos inteligentes, además de la capacidad de realizar y recibir llamadas y mensajes, se encuentra la presencia de una pantalla táctil, cámara, micrófono, GPS, WiFi y batería recargable.

Los sistemas operativos más comunes para teléfonos inteligentes son iOS, desarrollado por Apple y Android, desarrollado por Google. Otros sistemas menos populares son Windows Mobile, Ubuntu Touch, Tizen, etc.

En la actualidad, los principales productores de teléfonos inteligentes son Apple, que posee 18.3% del mercado, Samsung, quien cuenta con 18.1% del mercado y Huawei, con 10.6% del mercado.

En general el mercado de teléfonos inteligentes ha presentado una tendencia al alza, registrando 428.5 millones de equipos vendidos en el último cuarto del 2016 lo cual representó un 6.9% de incremento con respecto a las ventas registradas en el último cuarto del 2015. [7]

3.1.3 Relojes inteligentes

Un reloj inteligente, también conocido como Smartwatch, es un reloj de pulsera computarizado capaz de realizar funciones más avanzadas que un reloj tradicional tales como ejecutar aplicaciones y, en algunos casos, incluso realizar llamadas telefónicas.

Los relojes inteligentes utilizan un sistema operativo especialmente diseñado para este tipo de dispositivos que les permite ejecutar aplicaciones que expanden su funcionalidad. Los principales sistemas operativos utilizados por los relojes inteligentes son Android, desarrollado por Google y WatchOS, desarrollado por Apple.

Normalmente los relojes inteligentes pueden interactuar con un teléfono inteligente para complementar algunas de sus funciones permitiendo al reloj inteligente mostrar información de llamadas, mensajes, calendario, etc.

Si bien los relojes inteligentes cuentan con una funcionalidad más limitada que un teléfono inteligente o una Tablet, estos son de gran utilidad en varias actividades tal como en el cuidado de la salud en donde pueden ser utilizados para monitorear el ritmo cardiaco de una persona o bien en los deportes en donde pueden ser utilizados para recolectar información útil sobre las actividades realizadas.

Los principales productores de relojes inteligentes son Apple, con 63.4% del mercado y Samsung con 9.8% del mercado.

En general el mercado de relojes inteligentes ha presentado una modesta tendencia al alza, creciendo 1% anualmente con 8.2 millones de unidades vendidas en el último cuarto del año 2016.

3.2 Android

Android es un sistema operativo desarrollado por Google basado en Linux y dirigido principalmente a dispositivos móviles que cuenten con pantalla táctil tal como tabletas y teléfonos inteligentes.

La interfaz gráfica de Android está basada principalmente en manipulación directa en donde el usuario toca los elementos presentes en la pantalla para manipularlos o indicar las acciones que deben ser ejecutadas, además, cuenta con un teclado virtual para que el usuario pueda ingresar texto.

Android fue inicialmente desarrollado por la empresa Android Inc. La cual fue posteriormente adquirida por Google en el año 2005, Android fue anunciado al público en el año 2007 y un año después se dio a conocer el primer dispositivo comercial que utilizaba este sistema operativo.

El sistema operativo Android ha pasado por muchas actualizaciones, la versión 7.0 denominada Nougat se encuentra disponible desde Agosto de 2016.

Android cuenta con la ventaja de poder ser instalado en dispositivos fabricados por diferentes empresas y con distintas configuraciones de hardware, provee soporte para arquitecturas ARM y x86-64. Los requerimientos de RAM para las versiones más actuales del sistema varían desde 512 MB hasta 2 GB, dependiendo de otras características del dispositivo como la densidad de pantalla.

Los dispositivos que utilizan el sistema Android normalmente incorporan diversos componentes de hardware opcionales, entre ellos:

- ✓ Cámara de video.
- ✓ GPS.
- ✓ Sensores de orientación.
- ✓ Acelerómetro.
- ✓ Giroscopio.
- ✓ Barómetro.
- ✓ Magnetómetro.
- ✓ Sensores de proximidad.
- ✓ Sensores de presión.
- ✓ Termómetro.

Con el tiempo, este sistema operativo ha sido expandido generando con ello la aparición de variantes orientadas a diferentes áreas, en la actualidad se encuentra presente en:

- ✓ Teléfonos inteligentes.
- ✓ Tabletas.
- ✓ Relojes inteligentes.
- ✓ Computadoras personales.
- ✓ Televisiones.
- ✓ Automóviles.
- ✓ Internet de las cosas.
- ✓ Consolas de videojuegos.

Un problema presente en este sistema operativo es la fragmentación del mercado, esto es, que existen muchas versiones distintas de Android operando al mismo tiempo esto trae como consecuencia problemas de seguridad y limitaciones para los desarrolladores pues si desean utilizar alguna funcionalidad de la versión más reciente corren el riesgo de que una gran parte de los usuarios no sea capaz de utilizar la aplicación creada.

La fragmentación del mercado se debe principalmente a la lenta adopción de las versiones más recientes debido a que diferentes fabricantes muchas veces tienen que adaptarlas a sus configuraciones de hardware en particular, además, en muchos casos los fabricantes tratan de diferenciarse unos de otros creando sus propias interfaces gráficas las cuales también deben ser adaptadas a las nuevas versiones del sistema lo cual lleva tiempo.

En la figura 3.1 se muestra la fragmentación del mercado para el sistema operativo Android.

Versión	Nombre	API	Distribución
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.0%
4.1.x	Jelly Bean	16	3.7%
4.2.x		17	5.4%
4.3		18	1.5%
4.4	KitKat	19	20.8%
5.0	Lollipop	21	9.4%
5.1		22	23.1%
6.0	Marshmallow	23	31.3%
7.0	Nougat	24	2.4%
7.1		25	0.4%

Figura 3.1. Fragmentación del mercado. [12]

Actualmente Android es el sistema operativo con más usuarios en el mundo, tan solo en el año 2016 casi 9 de cada 10 teléfonos inteligentes vendidos utilizaban este sistema operativo.

La figura 3.2 muestra la proporción de usuarios que utilizan un sistema operativo para dispositivos móviles.

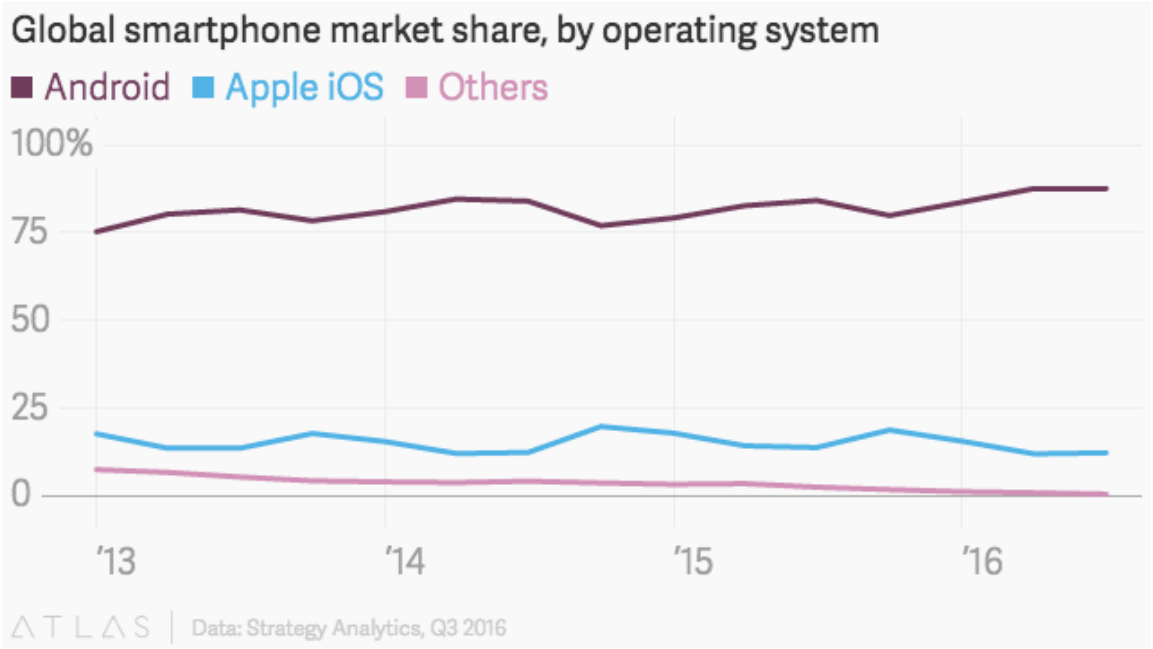

Figura 3.2 Proporción de usuarios por sistema operativo [11]

3.2.1 Arquitectura

Android es un sistema de código abierto basado en Linux el cual está conformado por varias capas cada una de las cuales desempeña una función específica.

La figura 3.3 muestra las diferentes capas que conforman el sistema operativo Android.

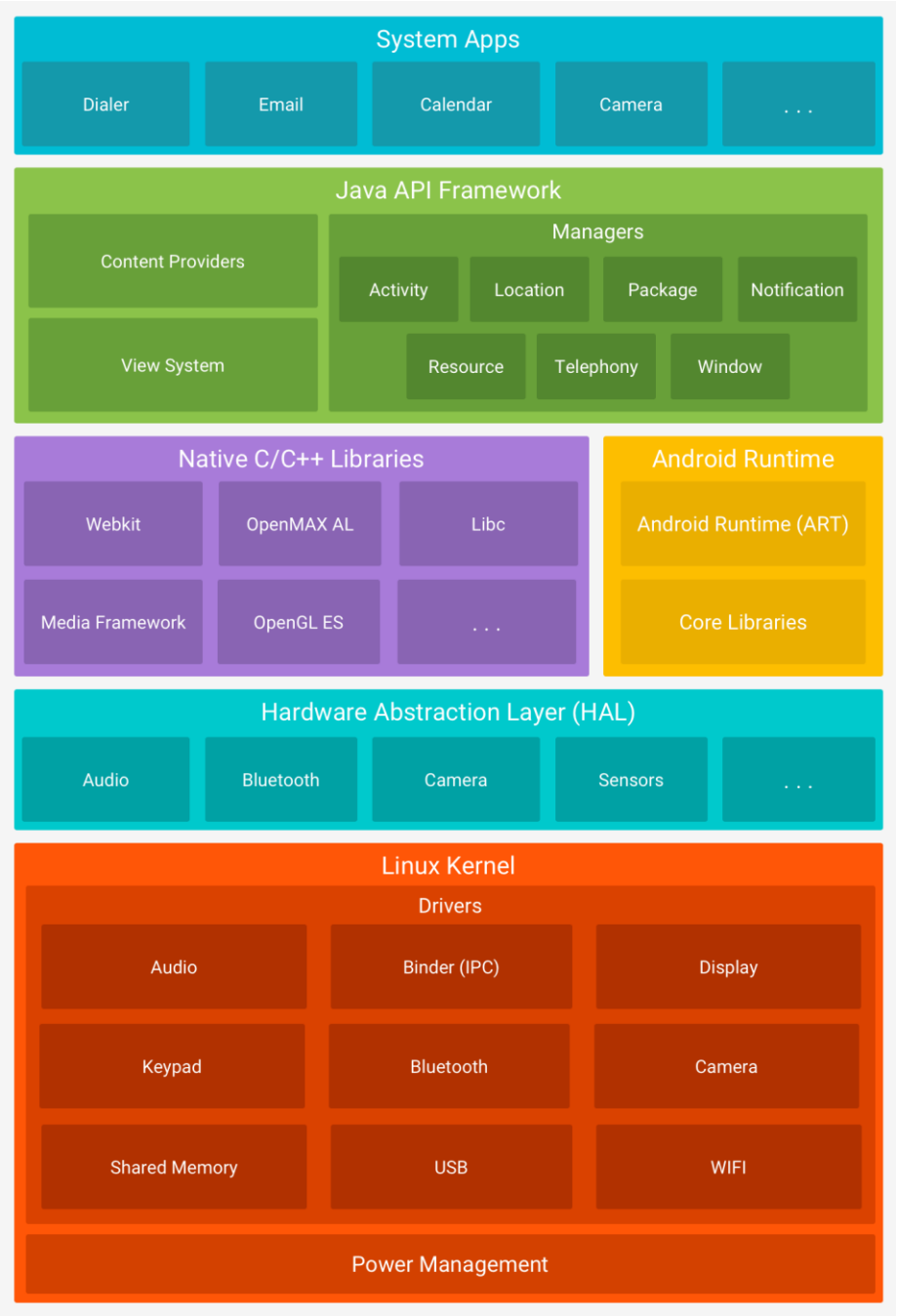

Figura 3.3 Principales componentes de Android. [13]

3.2.1.1 El núcleo Linux

La base de la plataforma Android es el núcleo Linux. Usar un núcleo basado en Linux le permite al sistema operativo aprovechar algunas características clave de seguridad además facilita a los fabricantes de dispositivos móviles la creación de controladores de hardware ya que Linux es una tecnología muy conocida.

3.2.1.2 Capa de abstracción de hardware

La capa de abstracción de hardware (HAL) provee de interfaces que exponen las capacidades de hardware del dispositivo a la API de java. HAL consiste de múltiples módulos de librerías cada una de las cuales implementa una interface para un tipo específico de componente de hardware. Cuando el API realiza una llamada para acceder al hardware del dispositivo el sistema carga el módulo correspondiente para ese componente de hardware.

3.2.1.3 Android Runtime

Para los dispositivos que utilizan la versión 5.0 (API nivel 21) o mayor de Android cada aplicación se ejecuta en su propio proceso y con su propia instancia de ART (Android Runtime).

ART está escrito para ejecutar múltiples máquinas virtuales en dispositivos que disponen de poca memoria mediante la ejecución de archivos DEX, el cual es un formato diseñado específicamente para Android optimizado para utilizar poca memoria.

Antes de la versión 5.0 de Android (API nivel 21), Dalvik era utilizado en lugar de ART.

3.2.1.4 Bibliotecas nativas C/C++

Muchos de los componentes y servicios principales del sistema como ART y HAL fueron creados con código nativo que requiere librerías escritas en C y C++. La plataforma Android provee un API para java para exponer la funcionalidad de algunas de esas bibliotecas nativas a las aplicaciones.

3.2.1.5 API de java

El set completo de funcionalidades de la plataforma Android está disponible para los desarrolladores por medio de API escritas en el lenguaje java. Estas API constituyen los bloques que requieren los desarrolladores para construir aplicaciones para la plataforma. Entre las funcionalidades incluidas se encuentran:

- ✓ Un sistema de visualización utilizado para construir la interfaz de usuario de la aplicación.
- ✓ Un gestor de recursos que provee acceso a recursos adicionales tales como gráficos, archivos de diseño, etc.
- ✓ Un gestor de notificaciones que permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- ✓ Un gestor de actividades que administra el ciclo de vida de las aplicaciones.
- ✓ Proveedores de contenido que permiten a las aplicaciones acceder a información de otras aplicaciones.

3.2.1.6 Aplicaciones del sistema

Android incluye un conjunto de aplicaciones básicas para correo electrónico, mensajes, calendario, contactos, etc. Las aplicaciones incluidas en la plataforma no gozan de algún estatus especial sobre las aplicaciones desarrolladas por terceros por lo que el usuario puede fácilmente cambiar que aplicación se ejecuta por default si así lo desea.

3.2.2 Componentes de una aplicación

La plataforma Android permite a los desarrolladores crear aplicaciones utilizando una serie de componentes reutilizables. Entre los principales componentes que los desarrolladores pueden utilizar se encuentran:

- ✓ Actividades.
- ✓ Fragmentos.

- ✓ Servicios.
- ✓ Proveedores de contenido.
- ✓ Transmisiones.
- ✓ Propósitos.
- ✓ Cargadores.

3.2.2.1 Actividades

Las actividades son uno de los bloques fundamentales para la creación de aplicaciones para la plataforma Android.

Sirven como punto de entrada para la interacción entre el usuario y la aplicación por medio de una ventana en la cual la aplicación muestra su interfaz gráfica.

La mayoría de las aplicaciones consta de múltiples pantallas lo cual significa que están conformadas por múltiples actividades.

Por lo regular una actividad es designada como la actividad principal de modo que será la primera pantalla que el usuario encontrará al momento de ejecutar la aplicación, posteriormente cada una de las actividades puede mandar llamar a otras actividades para realizar diferentes acciones.

A pesar de que las actividades trabajan juntas para formar una experiencia de usuario cohesiva en una aplicación cada actividad es relativamente independiente de las demás ya que existen pocas dependencias entre las actividades que conforman una aplicación. De hecho las actividades comúnmente mandan llamar actividades pertenecientes a una aplicación distinta.

A lo largo de su ciclo de vida las actividades pasan por una serie de estados los cuales sirven para describir algunas de las acciones que se realizan sobre la aplicación, por ejemplo, si la aplicación va a ser pausada, si se esté ejecutando por primera vez, si está a punto de ser destruida por el sistema, etc.

El desarrollador utiliza una serie de métodos (callbacks) para administrar las transiciones entre estados. Algunos de esos métodos son los siguientes:

- ✓ onCreate()
- ✓ onStart()
- ✓ onResume()
- ✓ onRestart()
- ✓ onPause()
- ✓ onStop()
- ✓ onDestroy()

Estos métodos permiten especificar que operaciones deben de realizarse cuando exista una transición entre estados, por ejemplo, se utilizan para preservar el estado de la actividad cuando el usuario navega hacia una pantalla distinta o bien para cargar información justo después de que la actividad ha sido ejecutada pero antes de que aparezca en pantalla.

Para lograr este fin los métodos deben ser sobrescritos, modificándolos de tal modo que se ejecute el método original de la clase padre (utilizando la palabra reservada super) y el código creado para realizar alguna tarea en específico.

La figura 3.4 muestra los diferentes estados que experimenta una actividad durante su ciclo de vida.

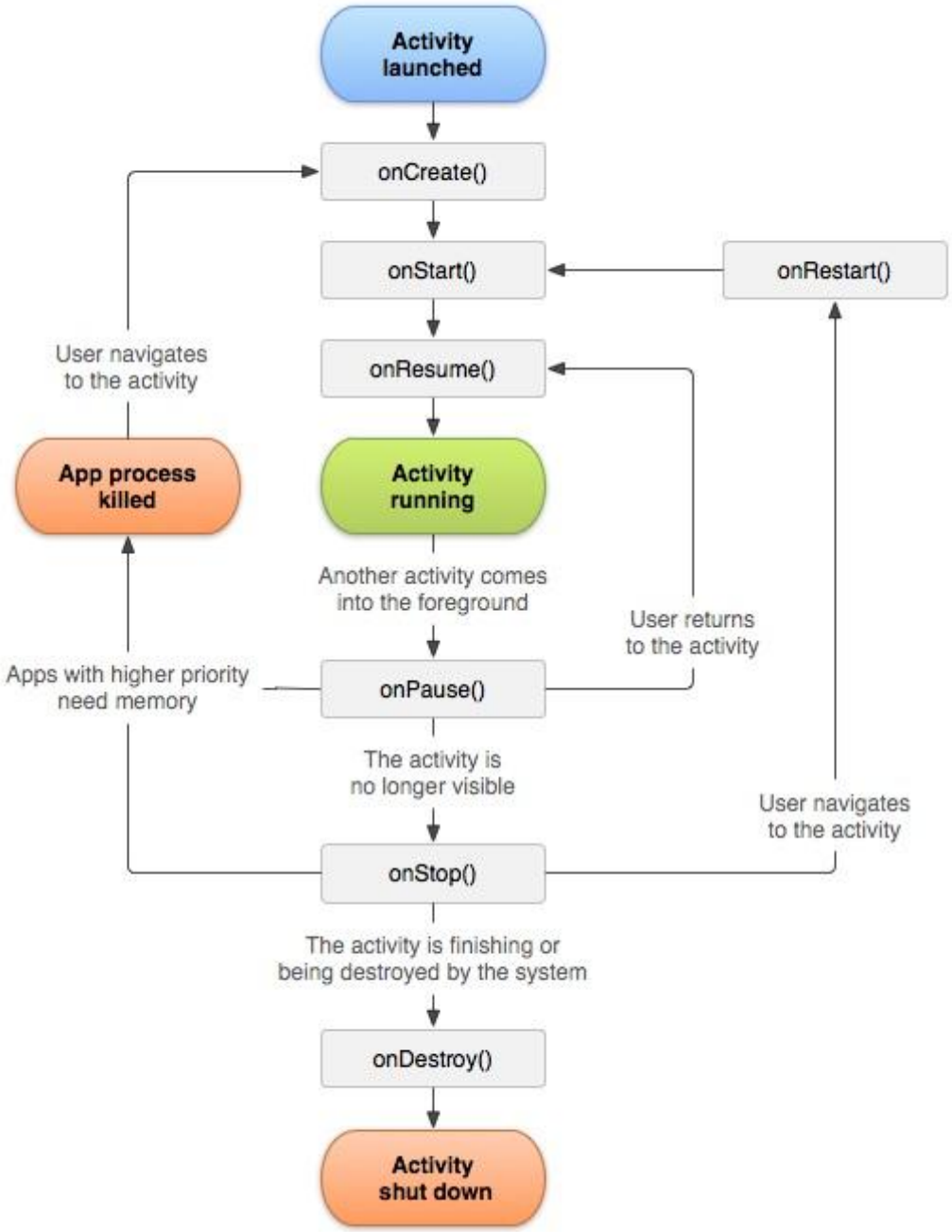

Figura 3.4 Ciclo de vida de una actividad. [13]

Normalmente, para realizar alguna tarea el usuario puede interactuar con una o varias aplicaciones, en este caso las actividades son colocadas en una pila (Back Stack) en el orden en que cada actividad es abierta.

Cuando la actividad actual manda llamar a otra, la nueva actividad es colocada en la pila y se convierte en la actividad actual (la que se muestra al usuario), la actividad anterior permanece en la pila pero se encuentra detenida. Cuando una actividad es detenida el sistema guarda el estado de su interfaz gráfica.

Cuando el usuario presiona el botón “atrás” la actividad actual es destruida y la actividad anterior reanuda su funcionamiento (el estado previo de su interfaz gráfica es restaurado).

Las actividades que se encuentran en la pila jamás son reordenadas, únicamente insertadas o sacadas de la pila respetando el principio de “ultimo en entrar, primero en salir” que caracteriza a este tipo de estructura de datos. La figura 3.5 muestra el funcionamiento de la pila.

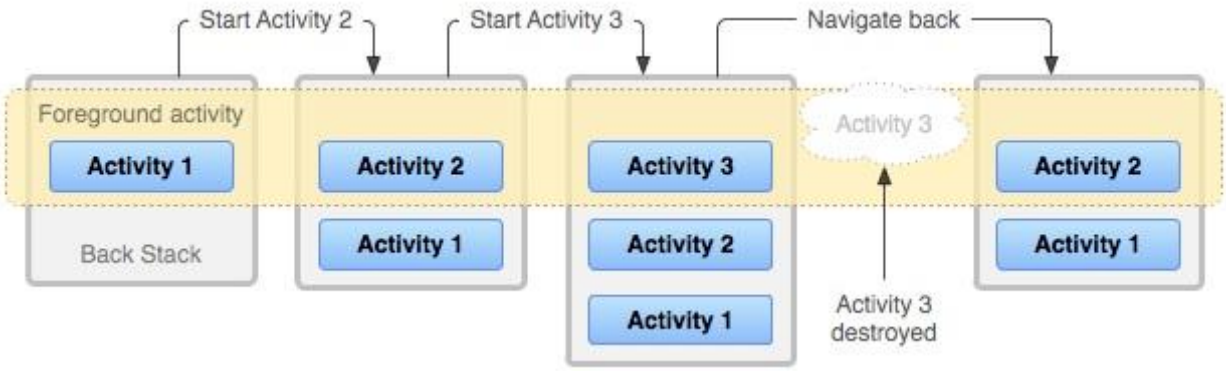

Figura 3.5 Funcionamiento de la pila. [13]

3.2.2.2 Propósitos

En el contexto de la plataforma Android un propósito es un objeto utilizado para solicitar la acción de un componente perteneciente a otra aplicación. Este tipo de objetos es utilizado principalmente en tres casos:

- **Iniciar una actividad:** Una actividad representa una sola pantalla o ventana dentro de una aplicación. Se puede iniciar una nueva actividad por medio de un propósito el cual el describe la actividad a iniciar y además lleva consigo los datos que se requieran. El propósito debe ser utilizado por el método startActivity().

- **Iniciar un servicio:** Un servicio es un componente que realiza una operación en segundo plano y que carece de una interfaz gráfica. A partir de la versión de Android 5.0 (API nivel 21) se puede iniciar un servicio utilizando JobScheduler.

- **Entregar una transmisión:** Una transmisión es un mensaje que cualquier aplicación puede recibir. El sistema entrega varios mensajes acerca de eventos

del sistema, como cuando el sistema inicia o el dispositivo se encuentra cargando. Se puede entregar una transmisión a otra aplicación pasando un propósito a los métodos sendBroadcast() o sendOrderedBroadcast().

Además, existen dos tipos de propósitos:

- **Propósitos explícitos:** Especifican el componente a iniciar por su nombre (nombre completo de la clase). Normalmente se utilizan para iniciar un componente de la misma aplicación ya que en este caso lo más probable es que se conozca el nombre de la actividad o servicio que se desea iniciar.

- **Propósitos implícitos:** No nombran a un componente en específico, en vez de eso declaran que acción se desea realizar, después, el sistema operativo basándose en la información declarada por los componentes de las otras aplicaciones selecciona que componente debe ser ejecutado para realizar la acción requerida. La figura 3.6 muestra el funcionamiento de un propósito implícito.

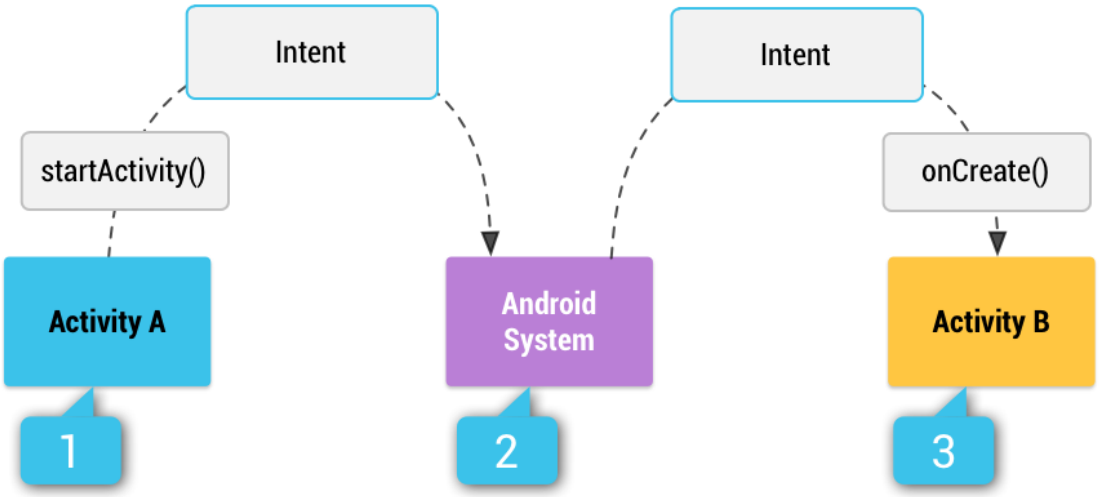

Figura 3.6 Propósito implícito [13]

3.2.2.3 Fragmentos

Un fragmento representa un comportamiento o una porción de la interfaz de usuario de una actividad. Se pueden combinar varios fragmentos en una sola actividad para crear una interfaz de usuario con múltiples paneles y reutilizar un fragmento en múltiples actividades.

Un fragmento puede ser visto como una sección modular de una actividad el cual tiene su propio ciclo de vida, recibe sus propias entradas y que puede ser agregado o removido mientras la actividad se está ejecutando.

Un fragmento siempre debe estar alojado en una actividad y su ciclo de vida es afectado de manera directa por el ciclo de vida de la actividad en la que se encuentra, por ejemplo, cuando la actividad es pausada también lo son todos sus fragmentos y cuando la actividad es destruida lo mismo ocurre con sus fragmentos.

Cuando un fragmento es agregado o removido de una actividad también hace uso de una pila administrada por la actividad esto permite al usuario revertir la última transacción aplicada al fragmento presionando el botón “atrás”.

Los fragmentos son principalmente utilizados en la creación de aplicaciones dirigidas a tabletas ya que estas disponen de más espacio para mostrar más elementos por lo que los desarrolladores pueden modificar la apariencia de la aplicación mientras esta se encuentra ejecutándose y además preservar dichos cambios en una pila sin necesidad de diseñar una interfaz de usuario que se vuelva demasiado compleja de administrar.

Cada fragmento debe ser diseñado como un componente modular y reusable de una actividad, ya que cada fragmento define su propia interfaz gráfica y su propio ciclo de vida este puede ser incluido en muchas actividades por lo que se debe evitar manipular un fragmento directamente desde otro fragmento. Esto es importante ya que el diseño modular de los fragmentos permite cambiar la combinación de fragmentos que se muestran dependiendo del tamaño de la pantalla. La figura 3.7 muestra un ejemplo de cómo los fragmentos pueden ser utilizados para alterar la interfaz gráfica de una aplicación según el tamaño del dispositivo.

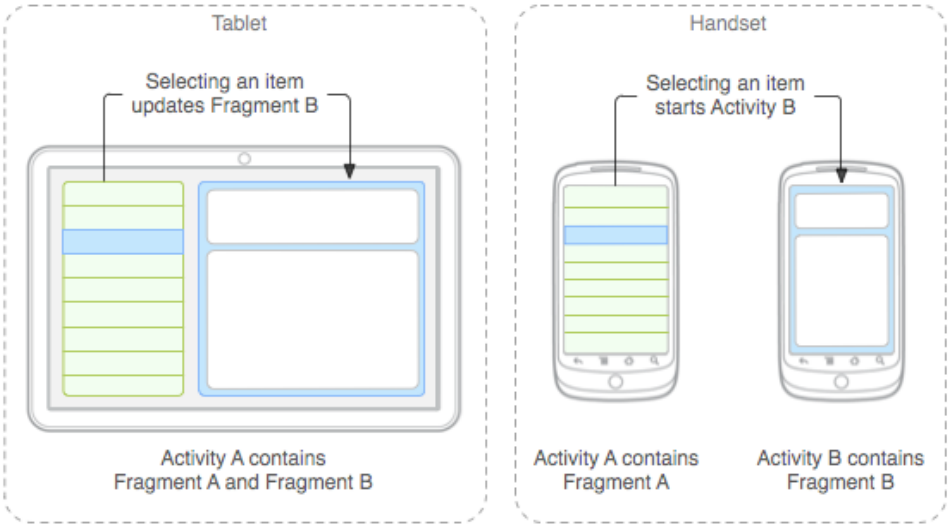

Figura 3.7 Diferentes combinaciones de fragmentos según el tamaño de la pantalla [13].

3.2.2.4 Cargadores

Un cargador permite cargar datos desde un proveedor de contenido u otra fuente de datos que sean requeridos por una actividad o fragmento. La utilización de cargadores trae consigo varios beneficios:

- ✓ Los cargadores se ejecutar en un hilo diferente para evitar ralentizar o bloquear la interfaz gráfica.

- ✓ Los cargadores simplifican la administración de sus hilos ya que proveen métodos para cuando algún evento ocurra.

- ✓ Los cargadores guardan los resultados aun cuando existan cambios de configuración para prevenir búsquedas duplicadas.

- ✓ Los cargadores pueden implementar un observador para monitorear cambios en la fuente de datos.

3.2.2.5 Servicios

Un servicio es un componente de una aplicación que puede realizar operaciones largas en segundo plano y que no cuenta con una interfaz gráfica. Un servicio puede ser iniciado por algún componente de la aplicación y este se seguirá ejecutando en segundo plano incluso si el usuario cambia de aplicación.

Existen tres tipos de servicios:

- **Programado:** Cuando un servicio es iniciado por una API como JobScheduler que permite registrar trabajos especificando sus requerimientos. Entonces el sistema programa los trabajos para su ejecución a un tiempo apropiado.

- **Iniciado:** Un servicio es iniciado cuando un componente de alguna aplicación manda llamar el método startService(). Después de ser iniciado un servicio puede ejecutarse en segundo plano de manera indefinida incluso si el componente que lo inicio es destruido. Usualmente un servicio iniciado realiza una sola operación y no retorna ningún resultado al componente que lo inicio, cuando la operación es completada, el servicio se detiene.

- **Ligado:** Un servicio es ligado cuando un componente de la aplicación se liga a él llamando el método bindService(). Un servicio ligado ofrece una interfaz

cliente-servidor que permite a los componentes interactuar con el servicio, enviar peticiones y recibir resultados. Un servicio ligado únicamente se ejecuta mientras algún otro componente de la aplicación se encuentre ligado a él. Múltiples componentes pueden ligarse al mismo servicio al mismo tiempo pero cuando todos ellos dejan de estar ligados el servicio es detenido.

Un servicio puede actuar como iniciado y ligado al mismo tiempo, es decir, puede ser iniciado para ejecutarse indefinidamente y además permitir que otros componentes se liguen a él.

Independiente de si el servicio es iniciado, ligado o ambos, cualquier componente de una aplicación puede utilizar el servicio (incluso desde una aplicación distinta) utilizando un propósito. Sin embargo, el servicio puede ser declarado como privado para bloquear el acceso desde otras aplicaciones.

Una consideración importante a tener en cuenta es que los servicios se ejecutan en el hilo principal del proceso que lo inició, es decir, un servicio no crea su propio hilo y no se ejecuta en un proceso separado a menos que se especifique otra cosa. Por esto es importante crear un hilo nuevo si es que el servicio va a ejecutar alguna tarea intensiva para reducir el riesgo de que la aplicación deje de responder.

3.2.2.6 Transmisiones

Las aplicaciones pueden enviar o recibir mensajes de otras aplicaciones y del propio sistema operativo. Estos mensajes son enviados cuando ocurre algún evento, por ejemplo, Android envía mensajes cuando ocurren ciertos eventos del sistema como cuando el sistema operativo es iniciado o cuando el dispositivo comienza a cargar.

Las aplicaciones también pueden enviar mensajes propios para notificar a otras aplicaciones acerca de algo en lo que puedan tener interés, por ejemplo, cuando se ha descargado nueva información.

Las aplicaciones pueden registrarse para recibir cierto tipo de transmisiones. Cuando una transmisión es realizada el sistema automáticamente la dirige hacia las aplicaciones que hayan sido registradas para recibir ese tipo de transmisión.

3.2.2.7 Proveedores de contenido

Los proveedores de contenido administran el acceso a un repositorio central de datos, permiten a una aplicación acceder a sus propios datos o bien a los datos almacenados por otra aplicación ya que proveen una manera de compartir información con otras aplicaciones.

Algunos de los casos en los que se utiliza un proveedor de contenidos:

- ✓ Compartir los datos de la aplicación con otras aplicaciones.
- ✓ Enviar datos a un Widget.
- ✓ Sincronizar los datos de la aplicación con los de un servidor.
- ✓ Cargar datos en la interfaz gráfica.

Un proveedor de contenido presenta los datos a las aplicaciones externas como una o más tablas las cuales son similares a las que se utilizan en las bases de datos relacionales. Un renglón de la tabla representa una instancia de algún tipo de datos recolectada por el proveedor y cada columna representa una pieza individual de los datos recolectados para una instancia.

Con el uso de este tipo de elementos se consigue una abstracción útil con respecto a la fuente o fuentes de datos empleadas en la aplicación de modo que se puede cambiar la fuente de datos sin que esto afecte a otras aplicaciones que requieran acceder a esos datos.

Los proveedores de contenido también permiten configurar la seguridad de acceso a los datos de modo que es posible restringir el acceso únicamente dentro de la misma aplicación, permitir el acceso desde otras aplicaciones o bien configurar diferentes permisos para lectura y escritura de datos.

3.3 Aprendizaje automático

El aprendizaje automático, también conocido como machine learning, es un campo de estudio de la inteligencia artificial que se encarga de otorgar a las computadoras la capacidad de aprender sin necesidad de ser programadas explícitamente.

Esta tecnología se centra en el desarrollo e implementación de algoritmos que sean capaces de aprender de los datos a los que son expuestos además de generar predicciones acerca de los mismos. Se utiliza principalmente para resolver tareas en las que la creación explicita de algoritmos específicos resulta difícil o imposible.

Algunas de las áreas en las que se ha utilizado esta tecnología son la clasificación de imágenes, detección de spam en correo electrónico, creación de vehículos autónomos, procesamiento de lenguaje natural, analizar y predecir movimientos de la bolsa de valores, detección de anomalías, etc.

En general, para llevar a cabo el proceso de aprendizaje con este tipo de algoritmos se deben llevar a cabo una serie de tareas.

1. Se debe seleccionar el algoritmo apropiado según el tipo de tarea que se desee realizar y de los datos con los que se cuente.

   Es común que existan varios algoritmos capaces de realizar la misma tarea, sin embargo, cada uno cuenta con ventajas y desventajas que deben ser consideradas para saber cuál es el más apropiado para resolver un determinado problema.

2. Se reúnen los datos con los que se va a trabajar, para el caso de algoritmos utilizados en aprendizaje supervisado, es importante que estos datos tengan asociada una etiqueta indicando a que categoría pertenecen, esta etiqueta será utilizada por el algoritmo para determinar si está obteniendo los resultados esperados o si necesita ajustarse.

3. Los datos deben ser preparados para ser procesados por el algoritmo, esto es, que se encuentren en un formato apropiado, que no existan duplicados, eliminar aquellas características que no sean útiles para el algoritmo, verificar que los datos se encuentren balanceados (que se cuente con un numero de ejemplos similar para las diferentes categorías), etc.

4. Dividir los datos en dos conjuntos, uno de entrenamiento y otro para pruebas, el conjunto de datos de entrenamiento se utilizará para entrenar el algoritmo mientras que el conjunto de datos de prueba será utilizado para verificar que el algoritmo es capaz de generalizar y producir buenos resultados en datos con los que no ha estado en contacto previamente.

   Los porcentajes pueden variar pero una buena práctica es asignar el 70% del total de datos para entrenamiento y el otro 30% para pruebas.

5. Ejecutar el algoritmo, el cual utilizará el conjunto de datos de entrenamiento para ajustar sus parámetros internos hasta lograr producir resultados satisfactorios según la tarea de que se trate.

6. Finalmente, utilizar el conjunto de datos de prueba para verificar que el algoritmo es capaz de generalizar y producir buenos resultados en datos nuevos.

   Es importante que el conjunto de prueba no sea utilizado para modificar los valores internos del algoritmo, únicamente debe ser empleado para verificar que el algoritmo haya sido entrenado correctamente.

Como se mencionó anteriormente, es necesario separar los datos en al menos dos conjuntos, uno para entrenamiento y otro para pruebas, esto debido a que en muchas ocasiones el algoritmo puede ajustarse demasiado a los datos con los que ha sido entrenado dando resultados satisfactorios con ese conjunto pero siendo incapaz de generalizar por lo que comete muchos errores al momento de encontrar datos nuevos, este fenómeno se conoce como sobre ajuste (overfitting).

El conjunto de datos de prueba es entonces utilizado para determinar si el algoritmo es capaz de generalizar o si se encuentra realizando un sobre ajuste.

En caso de ser necesario el sobre ajuste puede ser reducido por diferentes medios como puede ser el reducir parámetros en los datos, normalización, obtener más datos para entrenar el algoritmo, etc.

En la figura 3.8 se muestra el proceso de entrenamiento de un algoritmo el cual utiliza los datos de entrenamiento para generar una hipótesis, es decir, una función o modelo capaz de producir los resultados esperados de acuerdo a los datos de entrada.

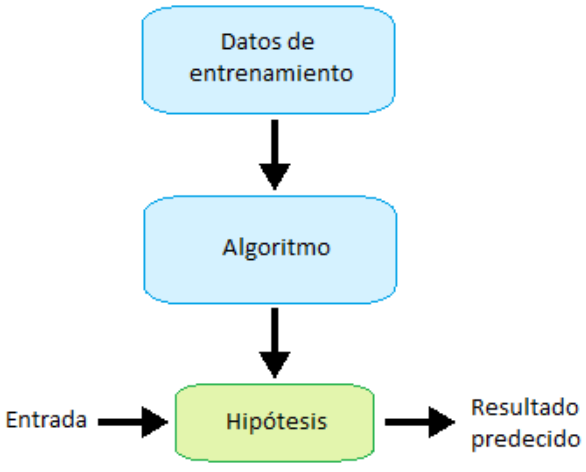

Figura 3.8 Entrenamiento de un algoritmo

3.3.1 Aprendizaje supervisado

Este tipo de algoritmos aprenden por medio de ejemplos, reciben un grupo de datos y el tipo de resultados que se espera que produzcan a partir de ellos, con esa información son capaces de aprender las reglas que deben ser aplicadas a las entradas para producir los resultados esperados.

Es llamado aprendizaje supervisado pues el proceso que siguen los algoritmos para aprender de los datos puede ser visto como una analogía a un maestro que supervisa a los algoritmos, proporcionando ejemplos, verificando que estos obtengan el resultado esperado y corrigiéndolos cuando no lo hacen.

Este tipo de aprendizaje es principalmente utilizado para tareas de clasificación y regresión lineal.

3.3.1.1 Regresión lineal

La regresión lineal es un modelo matemático que se utiliza para describir la relación existente entre una variable escalar dependiente y una o varias variables escalares independientes. El caso de una sola variable independiente se denomina regresión lineal simple mientras que cuando se cuenta con múltiples variables independientes se conoce como regresión lineal múltiple.

Este modelo matemático utiliza una función cuyos parámetros son derivados de los datos para predecir el valor de una variable ***y***.

La figura 3.9 muestra un ejemplo de regresión lineal simple, la gráfica muestra una serie de puntos (x, y), en donde ***y*** es la variable dependiente y ***x*** es la variable independiente, los parámetros de la función son derivados de los datos proporcionados.

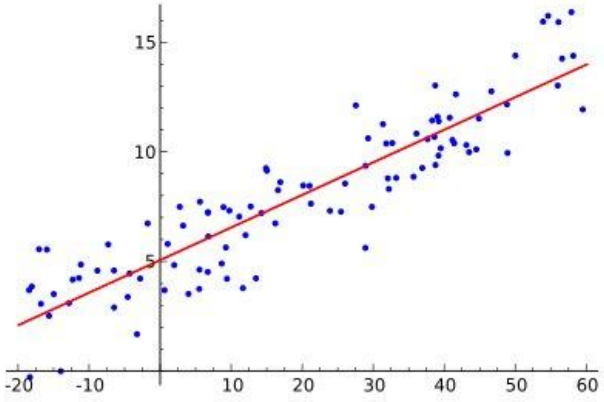

Figura 3.9 Regresión lineal simple

Para determinar el valor de los parámetros de la función, es decir la hipótesis, primero se asignan valores aleatorios a la hipótesis y después se utiliza lo que se conoce como función costo, esta consiste en calcular la diferencia promedio de todos los resultados calculados por la hipótesis para las entradas ***x*** y el valor real de la variable dependiente ***y***.

La ecuación 1 muestra la función costo

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

……ecuación 1

El objetivo es minimizar la función costo de manera que los resultados calculados por la hipótesis sean lo más cercanos posible a los valores reales modificando los valores de los parámetros theta 0 y theta 1 utilizando una técnica denominada gradiente descendente.

La figura 3.10 muestra una representación gráfica del objetivo de la función costo, el cual es minimizar la diferencia entre los valores calculados y los valores reales.

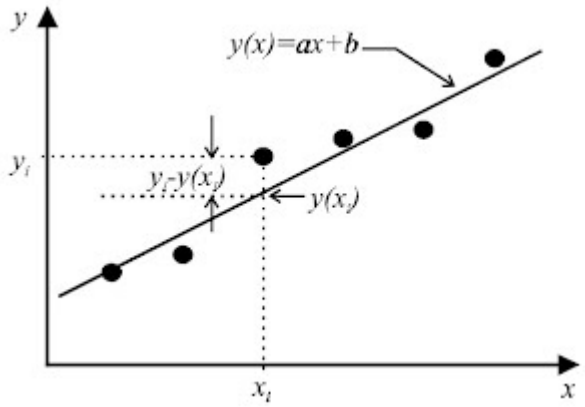

Figura 3.10 Diferencia entre valores calculados y reales

Este modelo matemático también puede generar funciones polinomiales más complejas para describir conjuntos de datos con más variables.

La regresión lineal es utilizada para resolver problemas en muchas áreas, como las finanzas, en donde puede ser utilizada para estimar el valor de las acciones de una empresa en la bolsa de valores, también encuentra uso para tratar de predecir las condiciones meteorológicas de algún lugar basado en datos sobre eventos pasados y en general, para determinar la tendencia que sigue un grupo de datos y tratar de predecir qué valor tomaran en el futuro.

3.3.1.2 Clasificación

Clasificación consiste en identificar a que categoría pertenece un dato utilizando un algoritmo previamente entrenado con un conjunto de datos pertenecientes a categorías conocidas. La categoría que le será asignada al nuevo elemento pertenece al conjunto de categorías con las que fue entrenado previamente el algoritmo.

Un algoritmo capaz de realizar la clasificación de datos es conocido como clasificador. Existen diversos modelos matemáticos que permiten realizar la clasificación de datos, uno de los más comunes es conocido como regresión logística.

La figura 3.11 muestra un sencillo problema de clasificación, en el que se requiere determinar si un nuevo dato pertenece a la primera categoría, representada en color azul o a la segunda, representada en color rojo.

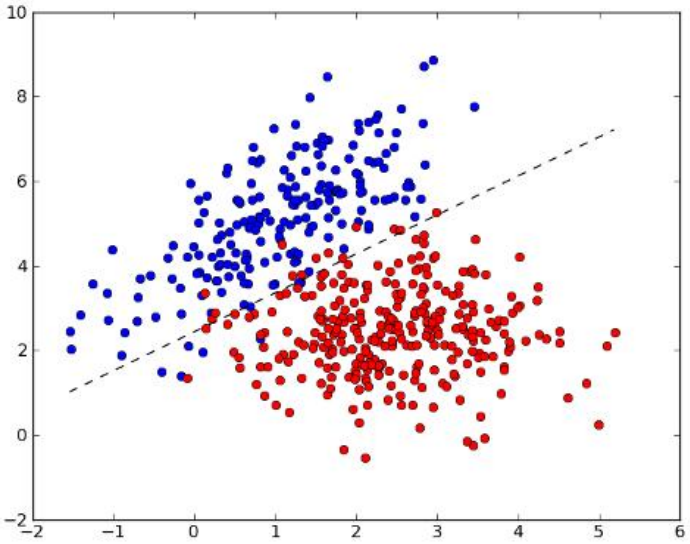

Figura 3.11 Clasificación con dos categorías.

La regresión logística es un modelo matemático que se utiliza para describir la relación existente entre una variable dependiente y una o varias variables independientes, se diferencia de la regresión lineal en que la variable dependiente es categórica, es decir, indica la probabilidad de que el dato analizado pertenezca o no a una determinada categoría.

Para determinar la probabilidad de que un dato pertenezca a una categoría en específico se hace uso de la función sigmoid, esto debido a que los valores de esta función varían entre 0 y 1.

La ecuación 2 muestra la función sigmoid

$$f(x) = \frac{1}{1+e^{-(x)}}$$ …….ecuación 2

La figura 3.12 muestra la gráfica de los valores que puede tomar la función sigmoid.

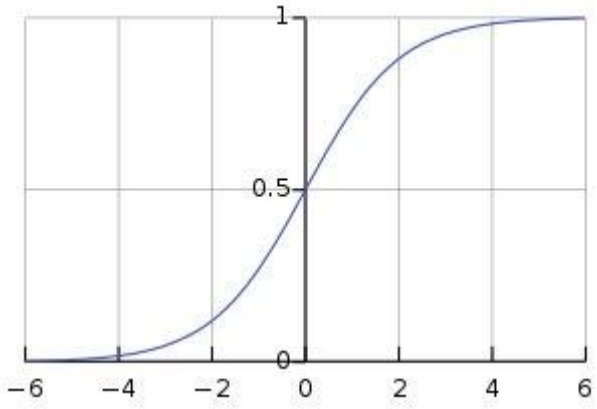

Figura 3.12 Grafica producida por la función sigmoid.

Por lo tanto, para determinar si un dato pertenece o no a cierta categoría primero se fija un límite, los valores que superen este límite son considerados como 1, es decir, que pertenece a la categoría mientras que los valores menores son considerados como 0, es decir, que no pertenecen a esa categoría.

En el caso de regresión logística también se utiliza una función costo, la cual deberá ser minimizada usando para ello el procedimiento matemático conocido como gradiente descendiente.

La ecuación 3 muestra la función costo utilizada en regresión logística

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$ ...ecuación 3

Por lo que para realizar la clasificación de un nuevo elemento se utiliza la función mostrada en la ecuación 4

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$ …….ecuación 4

Además de regresión logística también existen otros modelos que son utilizados para realizar la clasificación de datos, algunos de ellos son:

- ✓ Arboles de decisión
- ✓ Redes neuronales
- ✓ Vecinos más cercanos KNN (K- Nearest Neighbors)
- ✓ SVM (Support Vector Machine)
- ✓ Naive Bayes

Los algoritmos de clasificación son usados en muy diversas áreas, como análisis de imágenes, reconocimiento de voz, reconocimiento de caracteres, análisis de textos, clasificación de documentos, etc.

3.3.2 Aprendizaje no supervisado

Los algoritmos que pertenecen a esta categoría son capaces de descubrir patrones en los datos sin necesidad de que se les proporcionen ejemplos.

Este tipo de aprendizaje tiene la ventaja de poder ser utilizado en conjuntos de datos sin necesidad de etiquetarlos primero, por lo que pueden representar un ahorro en tiempo y recursos.

Los algoritmos más comunes de esta categoría son aquellos que sirven para realizar agrupamiento, es decir, separar los datos de entrada en varios sub conjuntos de datos similares.

3.3.2.1 Agrupamiento

El agrupamiento (clustering) consiste en agrupar un conjunto de datos en subconjuntos de tal forma que los datos contenidos en cada subconjunto sean más similares entre sí que con los datos contenidos en otros subconjuntos.

La tarea de agrupamiento puede ser lograda por diferentes algoritmos, cada uno de los cuales define métricas distintas de lo que constituye un subconjunto valido, además, algunos permiten que un solo dato pertenezca a más de un subconjunto al mismo tiempo mientras que otros solo permiten la pertenencia a un solo subconjunto.

Por ejemplo, el algoritmo llamado vecinos más cercanos (nearest neighbors) define la pertenencia a un subconjunto basado en la distancia que existe entre el dato y sus vecinos, siendo agrupado en el subconjunto con el cual exista la menor distancia.

La figura 3.13 muestra una representación gráfica de un dato y la distancia con sus vecinos más cercanos.

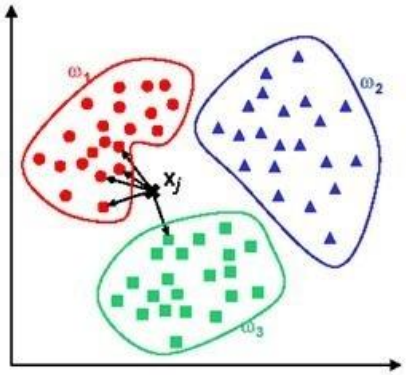

Figura 3.13 Distancia entre un dato y sus vecinos más cercanos

Por otro lado, los enfoques probabilísticos se enfocan en determinar la probabilidad que existe de que un dato pertenezca a un subconjunto. Uno de estos enfoques es el denominado mezcla de gaussianos el cual consiste en una serie de distribuciones gaussianas, cada una de las cuales representa un subconjunto.

En la figura 3.14 se muestra de manera gráfica un ejemplo de mezcla de gaussianos.

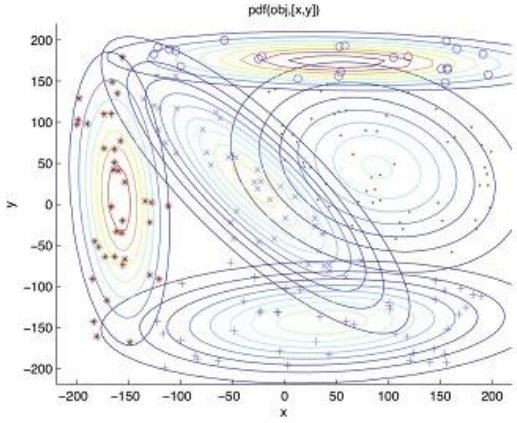

Figura 3.14 Mezcla de gaussianos

Existen otros algoritmos como el denominado latent Dirichlet allocation (LDA) el cual tiene la particularidad de permitir que un mismo elemento pueda pertenecer a más de un subconjunto al mismo tiempo. Por ejemplo, en el caso de documentos, en el cual un mismo texto puede tratar de más de un tema al mismo tiempo.

3.3.3 Aprendizaje reforzado

Esta área de estudio se enfoca en el desarrollo de agentes que sean capaces de determinar cuál es el comportamiento ideal dentro de un contexto específico para maximizar su desempeño.

Se requiere de un sistema de recompensas para que el agente aprenda su comportamiento ideal, estas recompensas son también conocidas como la señal de refuerzo.

El modelo básico de aprendizaje reforzado consiste de:

- ✓ Un conjunto de estados para el agente y su entorno.
- ✓ Un conjunto de acciones que puede realizar el agente.
- ✓ Reglas de transición de estados a acciones.
- ✓ Reglas que determinan la recompensa escalar inmediata de una transición.
- ✓ Reglas que describen lo que el agente observa.

Esta tecnología tiene diversas aplicaciones, como es en la navegación de vehículos autónomos, control de robots, juegos de lógica, etc.

3.3.4 Redes neuronales

Una red neuronal es un paradigma de procesamiento de información basado en una larga colección de simples unidades neuronales cada una de las cuales se encuentra conectada a otras unidades neuronales y cuyos enlaces pueden aumentar o inhibir la activación de estas. Este modelo está inspirado en el funcionamiento de las neuronas presentes en los seres vivos.

Las neuronas en este caso son unidades computacionales que reciben una serie de entradas las cuales son procesadas utilizando una función matemática (por ejemplo la función sigmoid) para producir una salida. A la función utilizada para procesar las entradas se le conoce normalmente como función de activación.

En una red neuronal artificial las neuronas se encuentran conectadas entre sí por medio de enlaces, cada uno de estos enlaces tiene asignado un valor numérico el cual es conocido normalmente como peso, estos pesos son utilizados para promover o inhibir la activación de otras neuronas.

La figura 3.15 muestra una neurona con n entradas, la cual utiliza una función de activación para producir una salida, esta a su vez puede ser utilizada como entrada por alguna otra neurona de la red.

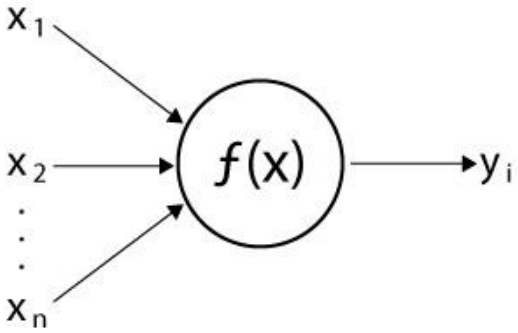

Figura 3.15 Neurona [27]

Normalmente las neuronas se encuentran organizadas en múltiples capas, por lo regular la primera capa se encarga de leer las entradas proporcionadas a la red, posteriormente una o más capas se encargan de realizan el procesamiento de los datos, a estas capas se les denomina capas ocultas, finalmente la última capa se encarga de producir el resultado final.

A las redes neuronales que están conformadas por múltiples capas ocultas también se les conoce como redes neuronales profundas.

El aumentar el número de capas ocultas puede aportar varios beneficios como el aumentar la efectividad de la red pero también puede traer como consecuencia que exista un problema de sobre ajuste por lo que el desarrollador debe tomar en cuenta las características propias del problema que se pretende resolver así como los datos de que se dispone para decidir si es necesario agregar más capas a la red.

La figura 3.16 muestra una red neuronal profunda con cinco capas ocultas

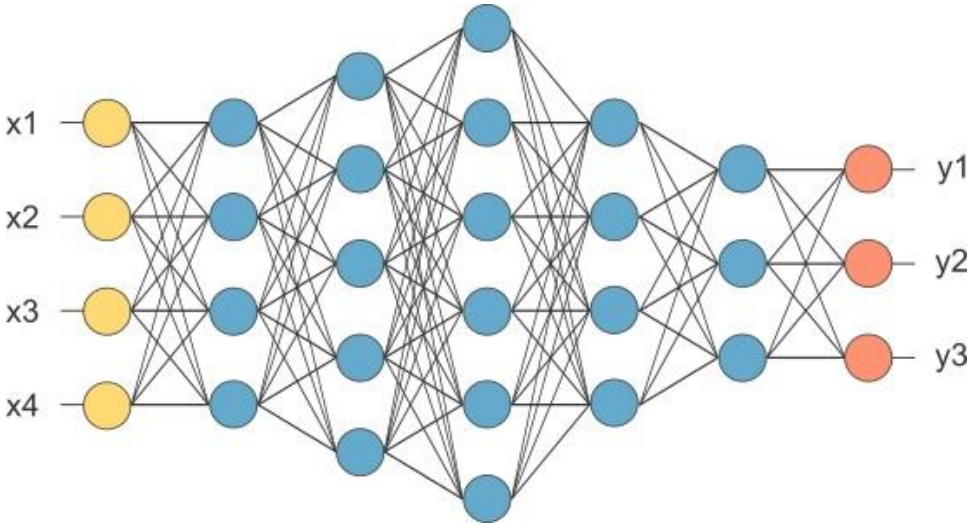

Figura 3.16 Red neuronal profunda [26]

Para determinar el valor que tiene cada neurona los diferentes valores numéricos que conforman la red son expresados en forma matricial, esto debido a que las bibliotecas de software especializadas en cálculos matemáticos están optimizadas para realizar operaciones con matrices por lo que resulta más rápido que utilizar ciclos o alguna otra técnica para recorrer los valores uno por uno.

Por lo general a cada capa oculta se le agrega una neurona extra denominada unidad de parcialidad la cual siempre tendrá el valor de 1.

Por ejemplo, si se cuenta con una red neuronal simple que consiste en la capa de entrada, una sola capa oculta y una capa de salida conformada por una sola neurona, esta puede ser representada por las matrices mostradas en la figura 3.17.

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_\theta(x)$$

Figura 3.17 Matrices de ejemplo

Los valores para cada una de las neuronas que se encuentran en la capa oculta se calculan según lo muestran las ecuaciones 5, 6 y 7, en donde theta es la matriz conformada por los pesos asignados a los enlaces entre la capa de entrada y la primera capa oculta.

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$ .....ecuación 5

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$ .....ecuación 6

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$ .....ecuación 7

Para obtener la salida de esta red neuronal se sigue un procedimiento similar, se multiplican los valores generados por la capa oculta con los pesos de los enlaces entre la capa oculta y la capa de salida, así como lo muestra la ecuación 8.

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$ .....ecuación 8

Si la red neuronal contara con más capas ocultas se seguiría aplicando este mismo procedimiento en cada una de las capas.

Como se puede apreciar el resultado final está influenciado por el procesamiento realizado en las capas ocultas, además, cada capa construye sus resultados basada en los resultados obtenidos por la capa anterior.

A este tipo de redes neuronales en las cuales los enlaces entre neuronas no forman un ciclo se les conoce como redes de propagación hacia adelante, La información fluye en un solo sentido, hacia adelante, comenzando por las neuronas de entrada, pasando por las neuronas que se encuentran en las capas ocultas (si es que existen) y finalizando con las neuronas de salida.

En una red neuronal que utiliza propagación hacia adelante las funciones de activación son fijas, además los datos de entrada son proporcionados por el usuario por lo que el modelo no tiene influencia sobre estos, por lo tanto, para entrenar una red neuronal de este tipo lo que se debe cambiar son los valores asociados a los enlaces entre neuronas, es decir, los pesos de la red.

Una red neuronal pertenece a la categoría de aprendizaje supervisado es decir, que requiere de ejemplos con la solución correcta para aprender a detectar patrones en los datos.

Los ejemplos son utilizados por la red para realizar ajustes a los valores de sus enlaces hasta lograr obtener resultados satisfactorios. El método utilizado para este fin es conocido como propagación hacia atrás.

En este caso se utilizará una función costo que representa la diferencia entre los valores calculados por la red neuronal y los valores correctos por lo cual debe ser minimizada utilizando la técnica de gradiente descendente.

La ecuación 9 muestra la función costo utilizada

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$ ..ecuación 9

El método de gradiente descendiente nos permite actualizar los pesos de la red. Ya que lo que se desea lograr es minimizar la función costo se debe restar el valor del gradiente del valor actual de los pesos para cada uno de los ejemplos disponibles.

Para efectuar el algoritmo de propagación hacia atrás se requiere seguir una serie de pasos, teniendo un conjunto de datos de entrenamiento conformado por entradas (representados por x) y sus respectivos resultados (representados por y) mostrado en la figura 3.18

$$\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$$

Figura 3.18 Conjunto de datos de entrenamiento

Para cada uno de los elementos contenidos en el conjunto de datos usado para entrenamiento, desde t=1 hasta m:

Los valores de activación de la primera capa deberán corresponder con los valores de entrada, tal como se muestra en la ecuación 10:

$$a^{(1)} := x^{(t)}$$ ……..ecuación 10

Calcular los valores de activación de las siguientes capas utilizando el algoritmo de propagación hacia adelante $a^{(l)}$ para l = 1, 2 , 3 …L

Usando los valores reales de los resultados calcular el error de la última capa, es decir, la diferencia entre los valores calculados por la red neuronal y los valores reales como se muestra en la ecuación 11:

$$\delta^{(L)} = a^{(L)} - y^{(t)}$$ ……..ecuación 11

Para las siguientes capas, el error se calcula utilizando la ecuación 12:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \, .* \, a^{(l)} \, .* \, (1 - a^{(l)})$$ ……..ecuación 12

También se utiliza un acumulador que permitirá ir agregando los resultados de las diferentes capas, mostrado en ecuación 13:

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$ ……..ecuación 13

Se procede a calcular la ecuación 14 para los casos en que j > 0, es decir, ignorando el nodo de parcialidad:

$$D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right)$$ ……ecuación 14

Y para los casos en que j = 0, se utiliza la ecuación 15:

$$D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$$ ….....….ecuación 15

Estos términos corresponden a la derivada parcial de la función costo (ignorando el término de normalización), como se muestra en la ecuación 16:

$$D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}$$ ….....…….ecuación 16

Finalmente, este gradiente dado por la derivada parcial de la función costo es utilizado para actualizar los pesos de la red.

Además de las redes neuronales de propagación hacia adelante existen otras variaciones que son más eficientes para ciertos tipos de tareas como son las redes neuronales recurrentes, las cuales han sido utilizadas en aplicaciones de procesamiento natural del lenguaje y las redes neuronales convolucionales, las cuales son más eficientes que las redes neuronales tradicionales para procesar imágenes.

3.3.4.1 Redes neuronales convolucionales

Las redes neuronales convolucionales son muy similares a las redes neuronales tradicionales ya que también están conformadas por unidades denominadas neuronas y son entrenadas modificando el valor asignado a los enlaces entre neuronas. Cada neurona recibe una entrada, realiza un cálculo y entrega un resultado.

Para una red neuronal tradicional el procesamiento de una sola imagen puede requerir una gran cantidad de parámetros pues se debe asignar una neurona de entrada por cada elemento presente en la imagen, por ejemplo, si tenemos una imagen de tamaño 200x200x3 (alto x ancho x profundidad) se requerirán de 120 000 neuronas solamente para la entrada además hay que agregar las neuronas de las capas ocultas y el hecho de que cada neurona normalmente tiene más de un enlace que debe ser actualizado y entonces se tiene una situación en la que se requiere de una cantidad considerable de procesamiento para una sola imagen.

Las redes neuronales convolucionales están diseñadas para trabajar con imágenes por lo que su arquitectura ha sido modificada para ser más eficiente procesando imágenes que las redes neuronales tradicionales requiriendo menos parámetros y aumentando la calidad de los resultados.

Este tipo de redes recibe como entrada una imagen y entrega como resultado una serie de probabilidades indicando la categoría a la que pertenece la imagen analizada.

Para realizar el procesamiento de imágenes una red neuronal convolucional cuenta con varios tipos de capas, los principales son la capa convolucional o capa conv, la capa de agrupación y la capa completamente conectada.

La capa conv consiste en una serie de filtros, cada filtro tiene unas dimensiones espaciales relativamente pequeñas (con respecto al ancho y altura) pero tiene la misma profundidad que el volumen de entrada. Este filtro se desplaza a lo largo y alto del volumen de entrada realizando el producto punto entre los valores del filtro y la entrada en cualquier posición.

Conforme el filtro se desplaza a lo largo y ancho del volumen de entrada se produce un mapa bidimensional de activación que indica las respuestas de ese filtro para cada posición.

De esta manera la red aprenderá una serie de filtros que se activaran cuando encuentren algún tipo de característica visual como puede ser una esquina, una mancha de color y eventualmente figuras más complejas en las capas posteriores.

Es común insertar una capa de agrupamiento entre capas conv, su función es la de reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y procesamiento requerido en la red y de esa manera también controlar el sobre ajuste.

Para lograr esto la capa de agrupación opera de manera independiente sobre cada nivel de profundidad del volumen de entrada utilizando operaciones como max, promedio, etc.

Por su parte, en la capa completamente conectada las neuronas están completamente conectadas a las activaciones de la capa anterior, al igual que en las redes neuronales tradicionales. Es esta capa la que generará el resultado final de la clasificación.

Normalmente, las redes neuronales convolucionales consisten en una serie de capas de estos tres tipos básicos, organizados de distintas formas según la arquitectura que se desee utilizar.

En general las arquitecturas más complejas que además están conformadas por múltiples capas requieren de una gran cantidad de datos y de recursos de procesamiento para su entrenamiento por lo que no es aconsejable crear este tipo de redes desde cero.

Lo más común es obtener un modelo previamente entrenado con un conjunto de datos denominado ImageNet, el cual consta de más de un millón de imágenes y después adaptarlo para trabajar con un nuevo conjunto de imágenes utilizando una técnica denominada transferencia de aprendizaje.

La transferencia de aprendizaje es una técnica para re entrenar una red neuronal con un nuevo conjunto de datos, esto se logra cambiando los valores de las últimas capas de la red mientras se mantienen intactos el resto de los valores. [5]

Capítulo 4

IMPLEMENTACIÓN

4.1 Red Neuronal

El desarrollo de este proyecto requiere la implementación de una red neuronal para realizar la identificación y clasificación de las imágenes obtenidas. Para lograr este fin primero se debe contar con el software adecuado.

La red neuronal requerida se implementará haciendo uso del lenguaje de programación Python en su versión 3.6.

La instalación de Python puede realizarse de una manera más sencilla con el uso de la plataforma Anaconda la cual incluye múltiples bibliotecas que son comúnmente utilizadas en aplicaciones de ciencia de datos y además permite crear diferentes entornos, cada uno de los cuales puede contar con una versión distinta de Python así como diferentes bibliotecas según se requieran para un proyecto en particular.

Una vez instalada la plataforma Anaconda se procede a crear un nuevo entorno llamado tensorflow el cual utilizará la versión 3.6 de Python, con el comando:

conda create --name tensorflow python=3.6

Posteriormente se procede a instalar la aplicación web Jupyter Notebook la cual permite al usuario crear programas y obtener visualizaciones de los datos utilizando un navegador web sin necesidad de instalar un IDE.

La instalación de esta plataforma se realiza con el comando:

conda install jupyter

Además se requiere contar con el conjunto de bibliotecas denominado SciPy el cual agrupa varias bibliotecas que son utilizadas en aplicaciones de matemáticas, ciencia e ingeniería.

Para realizar su instalación se utiliza el comando:

conda install scipy

La plataforma Tensorflow permite hacer uso de las capacidades que brindan las tarjetas gráficas para realizar cálculos matemáticos de una manera más rápida que los procesadores convencionales, esto permite reducir de manera considerable la cantidad de tiempo requerida para entrenar una red neuronal o algún otro algoritmo que requiera realizar una gran cantidad de operaciones matemáticas.

Para instalar la versión de Tensorflow que permite realizar operaciones matemáticas utilizando la tarjeta gráfica se utiliza el siguiente comando:

pip install tensorflow-gpu

Además de lo anterior se requieren de algunas bibliotecas adicionales para que la plataforma Tensorflow pueda interactuar con la tarjeta gráfica, estas bibliotecas son específicas de la marca y el modelo de tarjeta gráfica con la que se requiera trabajar.

En este caso se hará uso de hardware fabricado por la empresa Nvidia la cual proporciona las bibliotecas necesarias para que el desarrollador pueda utilizar el GPU para acelerar de manera considerable la ejecución de programas.

Para que las bibliotecas puedan ser utilizadas se requiere contar con una tarjeta gráfica que cuente con la tecnología CUDA, la cual es una plataforma de procesamiento en paralelo y modelo de programación desarrollada por Nvidia.

Esta tecnología soporta procesamientos heterogéneos en los que las aplicaciones utilizan tanto el CPU como el GPU, las porciones seriales de las aplicaciones son ejecutadas por el CPU mientras que las porciones paralelas son procesadas por el GPU.

El CPU y el GPU son tratados como dispositivos separados con sus propios espacios de memoria. Esta configuración permite realizar procesamientos simultáneos en el CPU y el GPU sin tener que preocuparse por los recursos de memoria.

Para verificar que el equipo cuenta con una tarjeta gráfica con la tecnología CUDA, se utiliza el siguiente comando:

lspci | grep -i nvidia

También se requiere contar con la biblioteca gcc, para verificar que se encuentre instalada se utiliza el comando:

gcc –version

Además se deben instalar los paquetes de desarrollo para el sistema operativo empleado, en este caso Ubuntu:

sudo apt-get install linux-headers-$(uname -r)

Posteriormente se obtiene el paquete NVIDIA CUDA Toolkit de la página de Nvidia el cual contiene el driver y demás herramientas necesarias para crear y ejecutar aplicaciones que hagan uso de la tarjeta gráfica.

Una vez que este paquete se encuentra instalado es necesario además contar con la biblioteca cuDNN la cual se encuentra optimizada para ejecutar muchas de las operaciones utilizadas en la creación de redes neuronales profundas aprovechando las capacidades de la tarjeta gráfica. Esta biblioteca puede ser descargada de la página oficial de Nvidia, en esta página también se encuentran instrucciones detalladas para su instalación.

Como siguiente paso se debe instalar la biblioteca libcupti-dev para lo cual se utiliza el siguiente comando:

sudo apt-get install libcupti-dev

La figura 4.1 muestra la salida que se produce cuando se carga la plataforma Tensorflow configurada para utilizar el GPU.

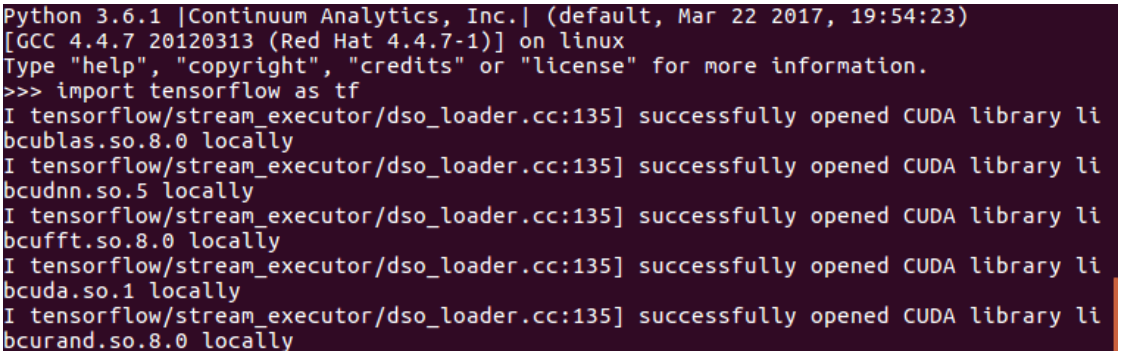

Figura 4.1 Bibliotecas CUDA

La figura 4.2 muestra la salida que se produce cuando el sistema identifica el GPU que será utilizado para realizar el procesamiento

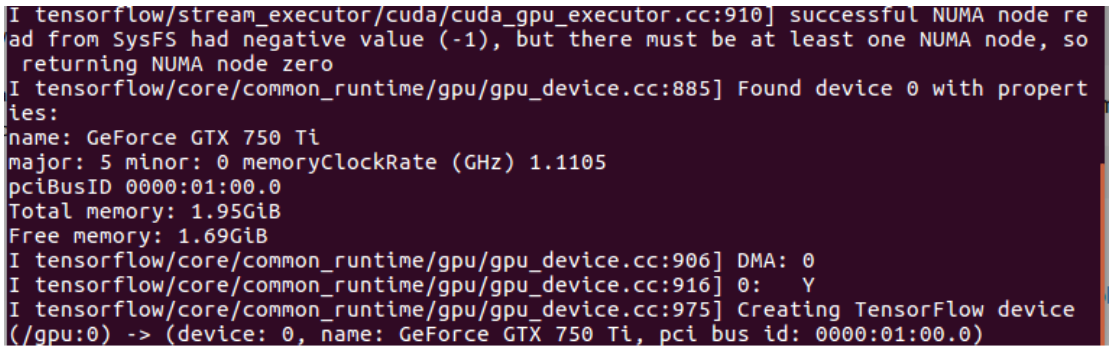

Figura 4.2 Uso del GPU

Para que una red neuronal pueda realizar la clasificación de las imágenes primero es necesario entrenarla, para ello se requiere de un conjunto de datos conformado por diferentes imágenes que correspondan con las clases o categorías que se requieren identificar, dicho conjunto de datos debe ser relativamente grande de manera que la red neuronal tenga suficientes muestras para aprender los patrones que caracterizan a cada categoría.

Debido a que se trata de un método de aprendizaje supervisado, además de las imágenes el sistema requiere que cada elemento tenga una etiqueta asociada a cada uno de ellos indicando a que categoría pertenecen.

Por ello se obtiene un conjunto de datos consistente en imágenes que muestran cada uno de los gestos que se requieren identificar.

Tensorflow proporciona un algoritmo que permite asociar las imágenes con su respectiva etiqueta pero primero se requiere que los archivos se encuentren organizados de cierta manera, manteniendo la siguiente estructura:

datos/nombre_clase_1/imagen1.jpg
datos/nombre_clase_1/imagen2.jpg
datos/nombre_clase_1/imagen3.jpg

datos/nombre_clase_2/imagen1.jpg
datos/nombre_clase_2/imagen2.jpg

datos/nombre_clase_3/imagen1.jpg
datos/nombre_clase_3/imagen2.jpg
datos/nombre_clase_3/imagen3.jpg

La etiqueta que el algoritmo asignará a cada imagen estará determinada por el nombre del folder en el cual se encuentra. Los nombres de cada archivo no son relevantes para este algoritmo.

Para que una red neuronal pueda obtener mejores resultados al momento de realizar la clasificación de imágenes es necesario que las imágenes que se utilicen para su entrenamiento sean variadas, es decir, que muestren al objeto que se desea clasificar en diferentes condiciones de iluminación, diferentes ubicaciones y en diferentes orientaciones.

Lo anterior se realiza con la finalidad de que la red neuronal aprenda a identificar las características que son relevantes para la identificación del objeto y descartar aquellas que no lo son.

Con este fin se procede a crear nuevas imágenes basadas en el conjunto original de imágenes al cual se le realizan algunas modificaciones como puede ser el utilizar una escala de grises, aumentar el brillo de la imagen, cambiar su ubicación, aumentar o disminuir la escala, etc.

Para lograr esta tarea se hace uso de la herramienta de software denominada ImageMagick la cual es una plataforma de código abierto que permite realizar diversas modificaciones a una imagen.

Por ejemplo, el siguiente comando permite cambiar el tamaño de una imagen a 299 por 299 pixeles y, en caso de que la imagen original no sea cuadrada, llenar las partes faltantes con un fondo negro, además guarda el resultado en un nuevo directorio llamado “data”:

mogrify -resize 299x299 -background black -gravity center -quality 100 -extent 299x299 -path data bg.jpg

En ocasiones se requiere aplicar una operación a todas las imágenes presentes en un directorio, en este caso es más conveniente utilizar un script que se encargue de realizar alguna operación sobre cada imagen, por ejemplo, el siguiente script permite recorrer cada una de las imágenes presentes en el directorio actual y asignarles un nuevo fondo, colocando la imagen original en el centro, el resultado de la operación es colocado en un nuevo directorio preservando los nombres originales de los archivos.

```
for filename in *.jpg; do
    basename="$(basename "$filename" .jpg)"
    composite -gravity center $basename.jpg bg.jpg ../converted/${basename}.jpg
done
```

Para este proyecto se utilizan imágenes generadas al aplicar una o más operaciones a cada una de las imágenes del conjunto original:

- ✓ Cambios de escala
- ✓ Cambios en la orientación
- ✓ Escala de grises
- ✓ Inversión de colores
- ✓ Cambios de brillo

Existen diferentes arquitecturas para redes neuronales, en particular, el presente proyecto utilizará la arquitectura conocida como Inception v3 la cual fue propuesta en el artículo titulado “Rethinking the Inception Architecture for Computer Vision” [1].

Esta arquitectura consiste en una red neural profunda con muchos niveles en los cuales se realizan muchas de las operaciones que normalmente se encuentran en redes neuronales convolucionales más sencillas como lo es la capa conv, la capa que realiza el promedio sobre una región de los elementos de la capa anterior, la capa completamente conectada, etc.

Esta gran cantidad de capas permite a este modelo obtener una exactitud mayor que otras redes neuronales más sencillas.

La figura 4.3 muestra la arquitectura de esta red neuronal.

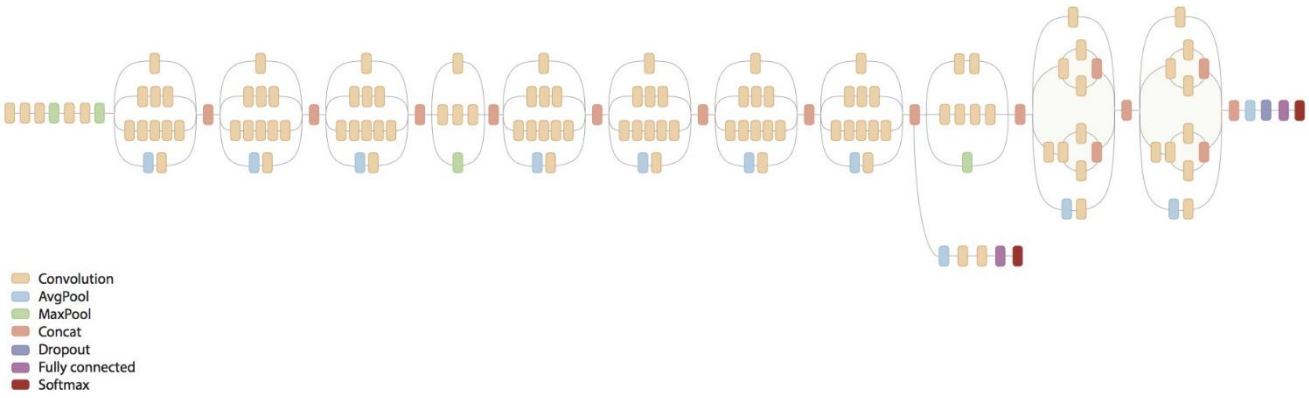

Figura 4.3 Arquitectura Inception v3 [1].

Al tratarse de una estructura con tantas capas se requiere de una gran cantidad de datos, así como de recursos de hardware y de tiempo para entrenarla y generar resultados satisfactorios.

Para minimizar los requerimientos que este tipo de arquitectura demanda se utiliza una técnica denominada transferencia de aprendizaje la cual consiste en tomar una red neuronal previamente entrenada y, mediante algunas modificaciones menores, adaptarla para trabajar con un nuevo conjunto de datos.

En este caso, el sistema Tensorflow proporciona un modelo previamente entrenado en el conjunto de imágenes conocido como ImageNet el cual consiste en más de un millón de imágenes correspondientes a más de mil categorías. Este modelo

fue entrenado con varias tarjetas gráficas operando en paralelo durante varias semanas. [18]

Por lo tanto el primer paso es descargar el modelo previamente entrenado, descomprimir el archivo y colocarlo en un folder previamente creado para este fin dentro de la estructura de archivos correspondiente a esta parte del proyecto.

Posteriormente, para realizar el reentrenamiento del modelo obtenido se deben cambiar los valores correspondientes a la última capa de clasificación utilizando las imágenes correspondientes a los gestos que se desean identificar. Las capas anteriores permanecen sin modificaciones pues estas se encargan de reconocer aspectos más básicos en una imagen como pueden ser líneas, esquinas, curvas, etc.

La plataforma Tensorflow proporciona una serie de algoritmos que permiten realizar esta tarea pero es necesario modificar algunos parámetros para adaptarlos a los datos con los que se cuenta así como los recursos de hardware disponibles.

En particular, al utilizar una tarjeta gráfica para realizar el entrenamiento de la red neuronal se deben tener en cuenta las limitaciones de memoria del dispositivo ya que debido a la gran cantidad de datos que se requieren así como de los múltiples valores que se actualizan en la red el proceso puede llegar a requerir de una gran cantidad de memoria.

Para solucionar este problema el reentrenamiento de la red se realiza utilizando solo una porción aleatoria de los datos disponibles en cada pasada, el tamaño de dicha porción podrá ser modificado en base a la cantidad de memoria disponible en la tarjeta gráfica.

Se hace uso del algoritmo denominado “retrain.py” para realizar el reentrenamiento de la red neuronal, este algoritmo se encuentra disponible en el repositorio de Tensorflow. [21]

Antes de ejecutar el algoritmo es necesario indicar algunos parámetros que permiten ajustar el proceso para el tipo de hardware con el que se cuenta además de especificar algunas operaciones que se pueden realizar sobre las imágenes antes de que estas sean procesadas, esto con el fin de generar un conjunto de datos más variado, en particular se utilizaron los siguientes valores:

- ✓ how_many_training_steps = 12000
- ✓ train_batch_size = 12

- ✓ random_brightness = 150

- ✓ flip_left_right = True

- ✓ random_scale = 30

El parámetro “how_many_training_steps” indica la cantidad de pasos que el algoritmo debe ejecutar para realizar el reentrenamiento de la red.

El parámetro “train_batch_size” indica la cantidad de elementos que deben ser utilizados para realizar el reentrenamiento de la red en cada pasada, este valor debe ser modificado considerando la cantidad de memoria con la que se cuenta.

El parámetro “random_brightness” es un porcentaje indicando el rango aleatorio de valores por los que los pixeles de la imagen de entrada deben ser multiplicados.

El parámetro “flip_left_right” indica si algunas de las imágenes deben ser volteadas como si estuvieran reflejadas en un espejo.

El parámetro “random_scale” es un porcentaje indicando un rango aleatorio de valores para una escala que debe ser aplicada a las imágenes antes de que estas sean procesadas.

La figura 4.4 muestra parte de la salida que se obtiene durante el reentrenamiento de la red neuronal.

```
2017-05-06 20:32:26.168782: Step 8630: Train accuracy = 58.3%
2017-05-06 20:32:26.168848: Step 8630: Cross entropy = 1.463254
2017-05-06 20:32:29.011012: Step 8630: Validation accuracy = 75.4% (N=3188)
2017-05-06 20:32:34.992858: Step 8640: Train accuracy = 83.3%
2017-05-06 20:32:34.992926: Step 8640: Cross entropy = 0.846141
2017-05-06 20:32:37.892440: Step 8640: Validation accuracy = 74.4% (N=3188)
2017-05-06 20:32:43.889484: Step 8650: Train accuracy = 75.0%
2017-05-06 20:32:43.889555: Step 8650: Cross entropy = 1.114956
2017-05-06 20:32:46.694983: Step 8650: Validation accuracy = 73.1% (N=3188)
2017-05-06 20:32:52.683983: Step 8660: Train accuracy = 83.3%
2017-05-06 20:32:52.684051: Step 8660: Cross entropy = 0.777754
2017-05-06 20:32:55.548811: Step 8660: Validation accuracy = 74.2% (N=3188)
2017-05-06 20:33:01.502098: Step 8670: Train accuracy = 58.3%
2017-05-06 20:33:01.502168: Step 8670: Cross entropy = 1.065355
2017-05-06 20:33:04.459138: Step 8670: Validation accuracy = 74.2% (N=3188)
2017-05-06 20:33:10.463655: Step 8680: Train accuracy = 83.3%
2017-05-06 20:33:10.463725: Step 8680: Cross entropy = 0.727506
2017-05-06 20:33:13.270538: Step 8680: Validation accuracy = 73.9% (N=3188)
2017-05-06 20:33:19.294786: Step 8690: Train accuracy = 66.7%
2017-05-06 20:33:19.294859: Step 8690: Cross entropy = 1.192508
2017-05-06 20:33:22.169715: Step 8690: Validation accuracy = 73.8% (N=3188)
2017-05-06 20:33:28.373374: Step 8700: Train accuracy = 83.3%
2017-05-06 20:33:28.373451: Step 8700: Cross entropy = 0.871566
```

Figura 4.4 Reentrenamiento de la red neuronal.

Una vez que el algoritmo concluye con el reentrenamiento de la red neuronal se generan dos archivos, el primero, un archivo de texto el cual contiene las etiquetas correspondientes a cada una de las categorías que la red es capaz de detectar. El segundo archivo generado contiene los nuevos valores de la red neuronal, estos serán utilizados para realizar la clasificación de nuevas imágenes.

4.2 Aplicación móvil

Para que el usuario final pueda hacer uso de la red neuronal previamente entrenada para identificar imágenes este requiere de una plataforma fácil de usar, en este caso, dicha plataforma consiste en una aplicación móvil que permite capturar imágenes con la cámara de un teléfono inteligente para posteriormente identificar a que clase corresponde.

Para desarrollar una aplicación para el sistema operativo Android primero se debe contar con ciertos elementos, como son:

- ✓ JDK 7 (Java Development Kit).
- ✓ JRE 6 (Java Runtime Environment).
- ✓ Android SDK.
- ✓ Android NDK.
- ✓ Android Studio.
- ✓ Emulador.

La plataforma Tensorflow se encuentra escrita en el lenguaje C++ por lo que para poder ser utilizada por Android primero es necesario contar con una interfaz que permita a las aplicaciones escritas en java interactuar con ella.

Para esto existen varias opciones, si se desea generar los archivos requeridos manualmente se puede hacer uso de la herramienta Bazel, la cual junto con Android NDK permitirá que la aplicación creada pueda hacer uso de las bibliotecas de Tensorflow.

También existe la posibilidad de obtener los archivos necesarios directamente de los repositorios de Tensorflow de modo que la aplicación puede hacer uso de ellos directamente sin necesidad de generarlos manualmente. [16]

Independientemente de la opción elegida, la aplicación requerirá de dos archivos para poder interactuar con la biblioteca Tensorflow:

- ✓ libandroid_tensorflow_inference_java.jar
- ✓ libtensorflow_inference.so

El primer archivo deberá ser colocado en la carpeta libs de la aplicación mientras que el segundo archivo deberá ser colocado en la carpeta que corresponda al tipo de arquitectura con él que cuente el dispositivo móvil.

La figura 4.5 muestra la estructura de archivos utilizada para guardar el archivo libtensorflow_inference.so

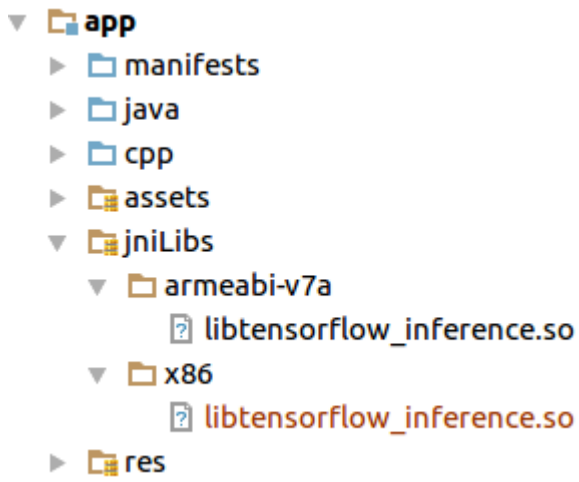

Figura 4.5 Estructura de archivos para Tensorflow en Android.

Además de lo anterior también será necesario contar con un modelo, en este caso una red neuronal, previamente entrenado con el tipo de imágenes que se requiere identificar.

El modelo entrenado previamente contiene una serie de nodos que si bien son útiles para realizar el entrenamiento de la red neuronal, no son requeridos para realizar la clasificación de las imágenes obtenidas por la aplicación, es por ello que se utiliza un script para removerlos. [22]

Esta operación tiene varias ventajas, como el hecho de reducir el tamaño del archivo correspondiente al modelo además de remover algunas operaciones que podrían no ser compatibles con ciertas plataformas.

Una vez que se han removido aquellos nodos innecesarios del modelo, se contará con dos archivos:

- ✓ output_graph.pb
- ✓ output_labels.txt

Estos archivos deberán ser colocados en la carpeta denominada assets dentro de la estructura del proyecto, tal y como se muestra en la figura 4.6

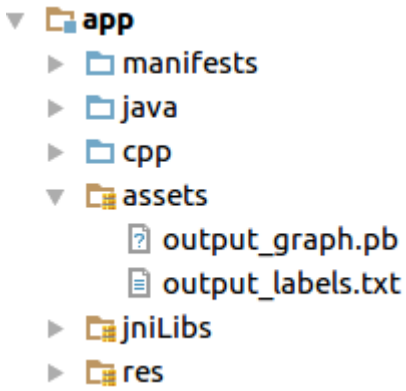

Figura 4.6 Estructura de archivos para el modelo previamente entrenado

Una vez que se cuenta tanto con las bibliotecas de Tensorflow así como con el modelo previamente entrenado es necesario crear los elementos requeridos para interactuar con ellos y realizar la clasificación de las imágenes obtenidas.

La aplicación consta de dos secciones principales, la primera permite al usuario realizar el análisis de las imágenes capturadas para saber a qué signo corresponden.

La segunda sección muestra los signos utilizados por el lenguaje de señas para representar las letras y algunos números, esto con la finalidad de que el usuario pueda conocerlos.

La primera sección se encarga del reconocimiento de signos, para ello muestra al usuario la imagen percibida por la cámara del dispositivo en ese momento, además cuenta con tres botones que le permiten capturar la imagen, cambiar de cámara y utilizar el flash del dispositivo.

Esta sección también cuenta con un área en la cual se muestra el resultado del análisis realizado a la imagen obtenida indicando el carácter al que la red neuronal ha asignado una mayor probabilidad de corresponder con la imagen.

Para facilitar la comunicación esta primera pantalla cuenta además con un área en la que se muestran los caracteres que han sido reconocidos recientemente de modo que sea más fácil entender que frases o palabras se quieren comunicar.

La figura 4.7 muestra la primera sección de la aplicación en el emulador.

Figura 4.7 Primera sección de la aplicación

Para realizar la navegación entre las secciones que conforman la aplicación se hace uso de un elemento denominado cajón de navegación el cual es un menú lateral que solo se hace visible cuando el usuario lo requiere, permaneciendo el resto del tiempo oculto, de modo que las actividades cuentan con mayor espacio para mostrar sus contenidos.

La figura 4.8 muestra el aspecto que tiene el cajón de navegación utilizado en esta aplicación.

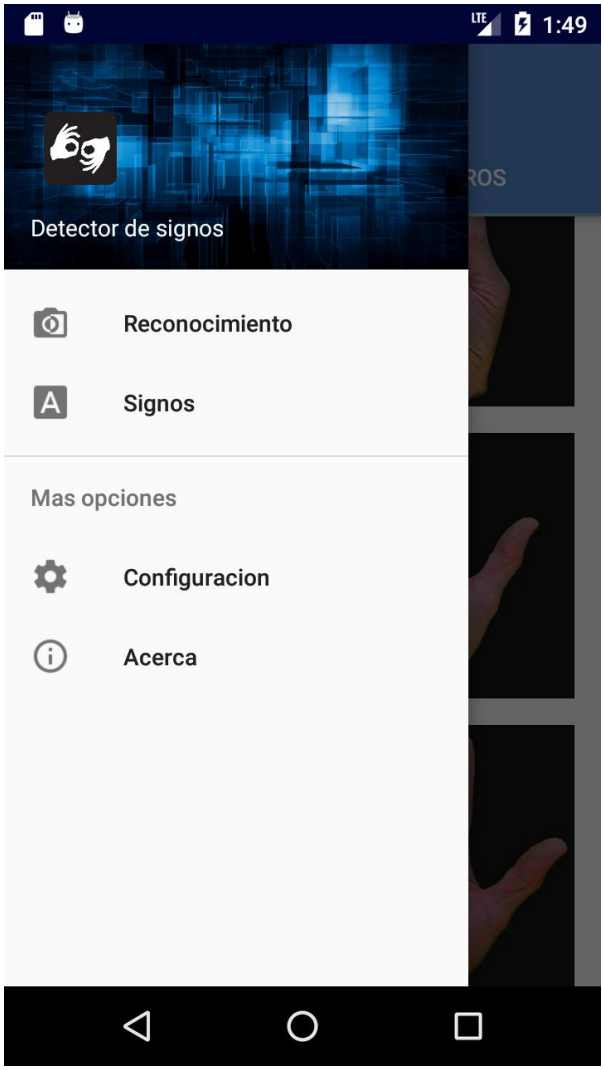

Figura 4.8 Cajón de navegación

La segunda sección muestra los signos del lenguaje de señas correspondientes a letras del abecedario y los números del 0 al 9.

Para lograr este fin se utilizan dos fragmentos dentro de la actividad los cuales están organizados por medio de pestañas.

Los fragmentos además permiten desplazar su contenido ya que los elementos ocupan mayor espacio que el que se encuentra disponible en la pantalla.

En la figura 4.9 aparece el fragmento que se encarga de mostrar los signos correspondientes a las letras del abecedario.

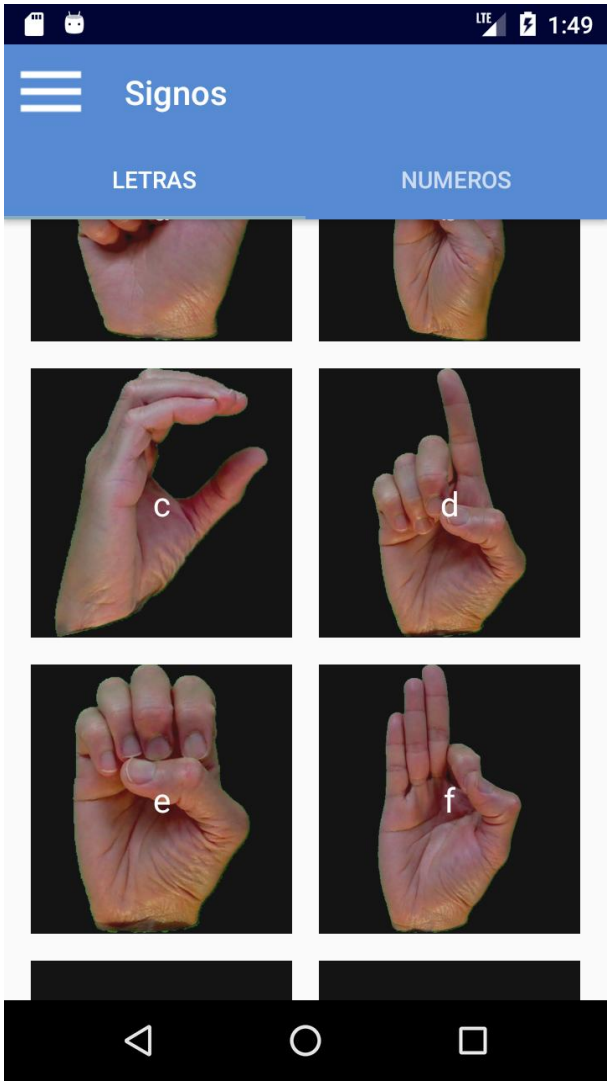

Figura 4.9 Signos utilizados para representar letras

En la figura 4.10 aparece el fragmento que se encarga de mostrar los signos correspondientes a los números del 0 al 9.

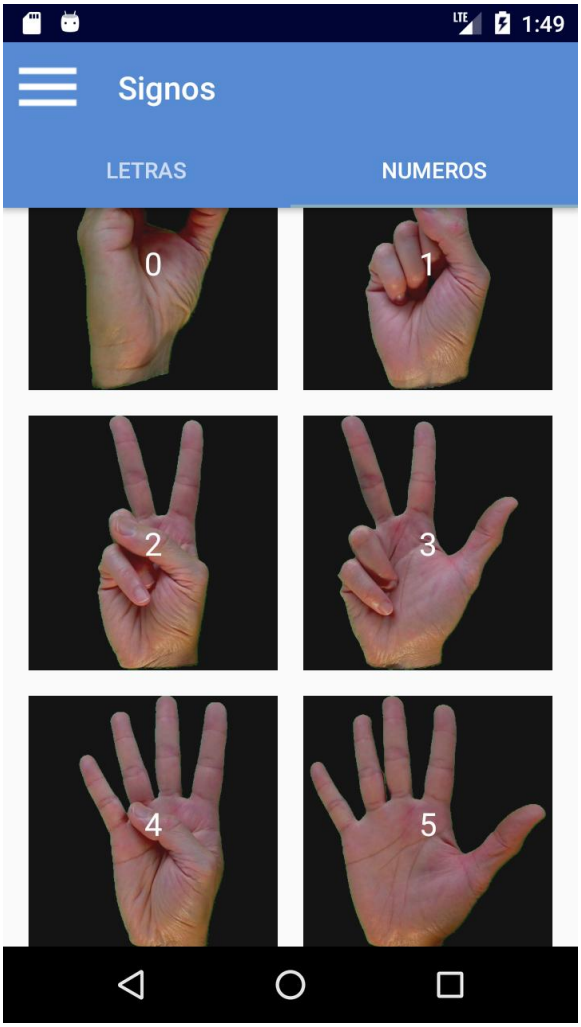

Figura 4.10 Signos utilizados para representar números

Capítulo 5

PRUEBAS Y RESULTADOS

Para realizar el entrenamiento de la red neuronal que se encarga de clasificar las imágenes capturadas por la cámara del dispositivo móvil se utiliza un conjunto de imágenes de ejemplo y un archivo de texto con las etiquetas correspondientes a cada una de las categorías que se desean identificar.

El conjunto de imágenes utilizado para el entrenamiento de la red neuronal es dividido en varios subconjuntos con el objetivo de ajustar los parámetros de la red y evaluar la calidad de los resultados generados, en este caso, los datos se dividen en:

- ✓ Datos de entrenamiento
- ✓ Datos de validación
- ✓ Datos de pruebas

El algoritmo utilizado para reentrenar la red neuronal genera una serie de registros que aportan información valiosa sobre el proceso.

Para poder visualizar estos registros, la plataforma Tensorflow proporciona una herramienta denominada Tensorboard la cual permite visualizar los datos generados en un navegador de internet.

Esta herramienta puede ser utilizada con el comando:

tensorboard logdir=/ruta/de/los/registros

Al invocar este comando se genera la salida mostrada en la figura 5.1, como se puede apreciar, la herramienta puede ser utilizada con cualquier navegador de internet, dirigiéndose a la dirección que se indica.

```
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
libcublas.so.8.0 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
libcudnn.so.5 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
libcufft.so.8.0 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
libcurand.so.8.0 locally
Starting TensorBoard b'41' on port 6006
(You can navigate to http://127.0.1.1:6006)
```

Figura 5.1 Herramienta Tensorboard

Una métrica útil para determinar el desempeño de la red neuronal es la exactitud con la que realiza la clasificación de los datos, es decir, la proporción de predicciones correctas de entre todas las predicciones realizadas por el sistema.

La figura 5.2 muestra la exactitud obtenida durante el proceso de reentrenamiento de la red neuronal con respecto al conjunto de datos de entrenamiento, representado por la línea naranja y el conjunto de datos de validación, representado por la línea verde.

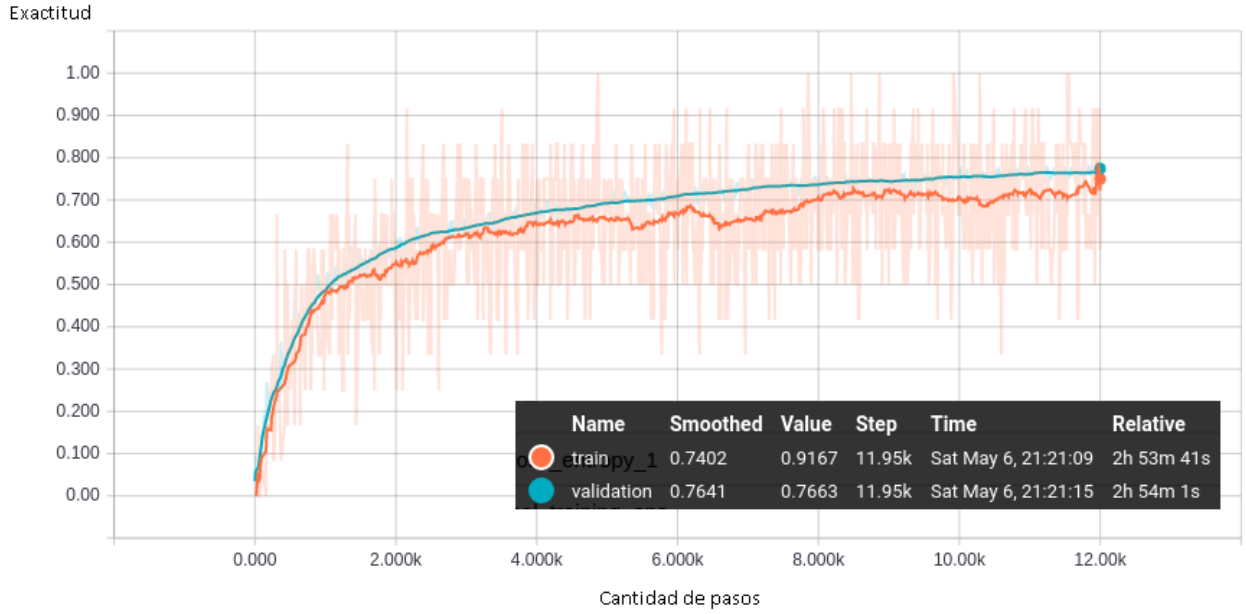

Figura 5.2 Exactitud de la red neuronal

Como se puede apreciar en un inicio la exactitud de la red neuronal se incrementa rápidamente con respecto al número de pasos pero eventualmente se estabiliza y la exactitud se incrementa cada vez menos conforme aumenta el número de pasos.

Por lo tanto, es posible observar que el realizar una gran cantidad de pasos de entrenamiento no necesariamente resultará en un incremento significativo de la exactitud de la red neuronal.

Una vez que ha concluido el reentrenamiento de la red neuronal, es necesario verificar su exactitud con el conjunto de datos de pruebas, esta operación también es realizada por el algoritmo utilizado para el reentrenamiento de la red neuronal generando el resultado que se muestra en la figura 5.3.

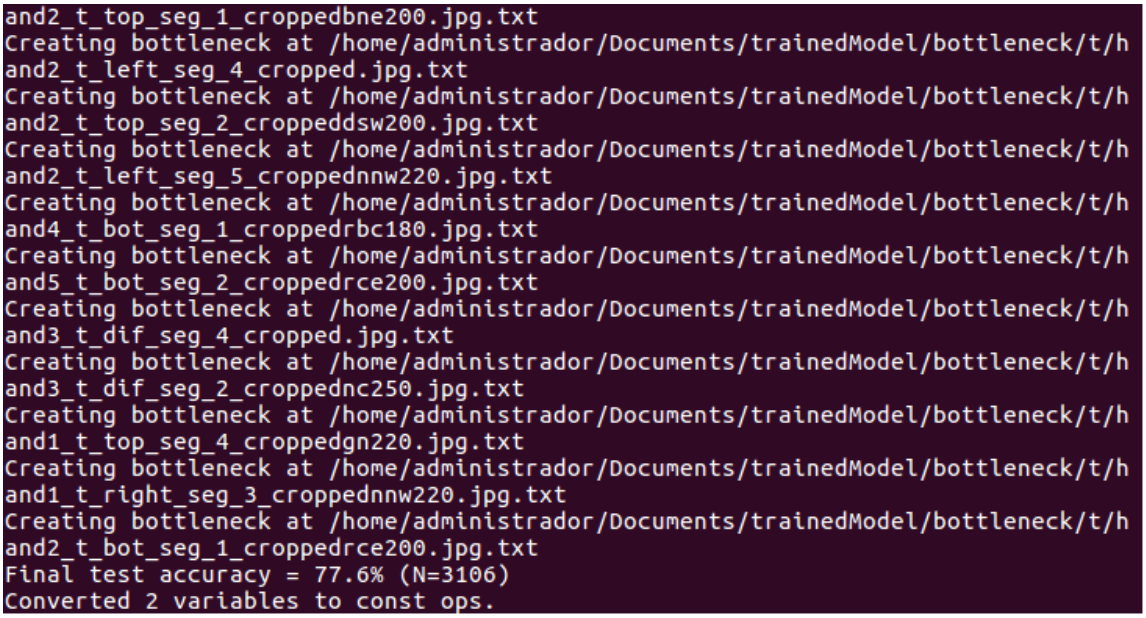

Figura 5.3 Exactitud final de la red neuronal

La red neuronal tiene una exactitud final de 77.6%, obtenida utilizando un conjunto de pruebas conformado por 3106 imágenes.

La exactitud no es mayor debido a que las imágenes utilizadas para resolver este problema son muy similares entre sí, todas las categorías consisten en imágenes de manos realizando diferentes señas por lo que las únicas diferencias entre una y otra son factores como la posición de los dedos o la orientación de la mano los cuales no generan diferencias lo suficientemente significativas, debido a esto la red neuronal puede llegar a confundir algunos de los signos, especialmente aquellos que son demasiado similares entre sí.

Para tener una idea más clara sobre el desempeño de la red neuronal, se utiliza una matriz de confusión, la cual muestra con más detalle que categorías están siendo clasificadas correctamente y cuales están siendo confundidas por el algoritmo.

Las figuras 5.4, 5.5, 5.6 y 5.7 muestran la matriz de confusión obtenida para las distintas categorías que se requieren identificar.

Real	0	1	2	3	4	5	6	7	8	9
0	78	0	0	0	0	0	0	0	0	0
1	0	48	0	0	0	0	0	1	3	4
2	0	1	42	0	3	0	21	2	2	4
3	0	0	1	63	0	0	0	0	0	3
4	0	0	0	1	41	13	8	2	7	8
5	0	0	0	0	2	81	2	0	1	5
6	0	0	0	0	2	0	55	4	3	4
7	0	0	2	1	0	0	0	38	25	9
8	0	0	0	0	0	0	1	1	75	9
9	0	0	1	0	1	2	0	2	6	75
a	0	0	0	0	0	0	0	0	0	0
b	1	0	0	0	0	0	0	0	0	0
c	2	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	3
e	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	3
g	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	3
k	0	0	0	0	0	0	0	0	1	2
l	0	0	0	0	0	0	0	0	0	2
m	1	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0
o	49	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0
r	1	0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	1
v	0	1	14	0	0	0	10	8	5	9
w	0	0	0	1	8	1	40	4	1	4
x	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0
Total	133	50	60	66	57	97	137	62	129	148

Figura 5.4 Matriz de confusión

Real	a	b	c	d	e	f	g	h	i
0	0	0	3	0	0	0	0	0	0
1	0	0	0	5	0	0	1	0	19
2	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0
4	0	5	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	4	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	0	2
8	0	1	0	1	0	0	0	0	3
9	0	0	0	0	0	0	0	0	1
a	74	0	0	0	0	1	0	0	4
b	0	96	0	1	0	1	0	0	0
c	0	0	96	0	0	0	0	0	0
d	0	1	0	79	1	1	0	0	4
e	0	0	0	1	79	0	0	0	6
f	0	9	0	0	1	58	0	0	3
g	0	0	0	0	0	0	82	3	0
h	0	0	0	0	0	0	21	63	0
i	0	0	0	0	1	0	0	0	92
k	0	1	0	0	0	0	0	0	3
l	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	0	0	0	2
n	0	0	0	0	0	1	0	0	2
o	0	1	1	0	0	1	0	0	1
p	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	1	0
r	0	0	0	0	0	0	0	0	2
s	2	0	0	0	3	0	0	0	21
t	7	0	0	0	2	0	0	0	0
u	0	5	0	2	0	3	0	0	4
v	0	0	1	3	0	0	0	0	7
w	0	2	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	6
y	0	0	0	0	0	0	0	0	7
Total	83	125	101	93	88	66	104	67	190

Figura 5.5 Matriz de confusión

Real	k	l	m	n	o	p	q	r
0	0	0	0	1	10	0	0	0
1	5	8	0	0	0	0	0	7
2	10	4	0	0	0	0	0	7
3	0	17	0	0	0	0	0	2
4	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	2	4	0	0	0	0	0	0
7	0	1	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0
a	0	0	1	2	0	0	0	0
b	0	0	0	0	0	0	0	0
c	3	1	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0
e	0	0	5	4	0	0	0	0
f	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	1	0
h	0	0	0	1	0	0	0	0
i	0	0	0	0	0	0	0	0
k	97	1	0	0	0	0	0	6
l	0	83	0	0	0	0	0	0
m	0	0	58	23	0	0	0	0
n	0	0	11	61	0	0	0	0
o	0	0	0	0	35	0	0	0
p	0	0	0	0	0	93	0	0
q	0	0	0	0	0	2	108	0
r	2	0	0	0	0	0	0	83
s	0	0	12	8	0	0	0	0
t	0	0	0	6	0	0	0	0
u	0	0	0	0	0	0	0	3
v	7	0	0	0	0	0	0	3
w	15	2	0	0	0	0	0	3
x	1	0	1	0	0	0	0	1
y	0	6	0	0	0	0	0	0
Total	143	128	88	106	45	95	109	116

Figura 5.6 Matriz de confusión

Real	s	t	u	v	w	x	y	Total
0	1	0	0	0	0	0	0	93
1	0	0	0	1	0	0	1	103
2	0	0	4	15	0	0	0	116
3	0	0	0	0	0	0	0	86
4	0	0	0	1	0	0	0	87
5	0	0	0	0	0	0	0	91
6	0	0	0	0	2	0	0	80
7	0	0	0	4	0	0	0	84
8	0	0	0	0	0	0	0	91
9	0	0	0	0	0	0	0	89
a	0	10	0	0	0	0	2	94
b	0	0	0	0	0	0	0	99
c	0	0	0	0	0	0	0	102
d	0	0	3	2	0	1	0	95
e	1	4	0	0	0	0	0	100
f	0	1	3	0	0	1	0	79
g	0	0	0	0	0	0	0	86
h	0	0	0	0	0	0	0	85
i	0	0	0	0	0	0	0	96
k	0	0	1	0	0	0	0	112
l	0	1	0	0	0	0	0	86
m	0	0	0	0	0	0	0	85
n	0	5	0	0	0	0	0	80
o	0	0	0	0	0	0	0	88
p	0	0	0	0	0	0	0	93
q	0	0	0	0	0	0	0	111
r	0	0	9	0	0	0	0	97
s	46	2	0	0	0	0	0	95
t	0	55	0	0	0	0	0	70
u	0	0	72	1	0	0	0	91
v	0	1	2	20	0	0	0	91
w	0	0	1	2	12	0	0	96
x	1	0	0	0	0	65	0	75
y	0	0	0	0	0	0	67	80
Total	49	79	95	46	14	67	70	3106

Figura 5.7 Matriz de confusión

Una vez que la red neuronal es integrada para trabajar con la aplicación móvil es necesario verificar que todo el sistema se encuentre operando correctamente, para ello primero se utilizan cuatro imágenes pertenecientes a cuatro categorías distintas seleccionadas al azar y se verifica que la aplicación se capaz de clasificarlas correctamente.

Para lograr este fin, se utiliza la prueba presentada en la figura 5.8.

```
@Test
public void classifyImageTest() throws Exception {

    MainActivity activity = (MainActivity)mActivityRule.getActivity();

    InputStream picture1 = this.getClass().getClassLoader().getResourceAsStream("gestureH.jpg");
    Bitmap bitmap1 = BitmapFactory.decodeStream(picture1);
    bitmap1 = Bitmap.createScaledBitmap(bitmap1, 299, 299, false);

    InputStream picture2 = this.getClass().getClassLoader().getResourceAsStream("gestureQ.jpg");
    Bitmap bitmap2 = BitmapFactory.decodeStream(picture2);
    bitmap2 = Bitmap.createScaledBitmap(bitmap2, 299, 299, false);

    InputStream picture3 = this.getClass().getClassLoader().getResourceAsStream("gestureP.jpg");
    Bitmap bitmap3 = BitmapFactory.decodeStream(picture3);
    bitmap3 = Bitmap.createScaledBitmap(bitmap3, 299, 299, false);

    InputStream picture4 = this.getClass().getClassLoader().getResourceAsStream("gestureA.jpg");
    Bitmap bitmap4 = BitmapFactory.decodeStream(picture4);
    bitmap4 = Bitmap.createScaledBitmap(bitmap4, 299, 299, false);

    final List<Classifier.Recognition> results1 = activity.classifyImage(bitmap1);
    Log.v(TAG, "Scores for image H: "+results1.toString());
    final List<Classifier.Recognition> results2 = activity.classifyImage(bitmap2);
    Log.v(TAG, "Scores for image Q: "+results2.toString());
    final List<Classifier.Recognition> results3 = activity.classifyImage(bitmap3);
    Log.v(TAG, "Scores for image P: "+results3.toString());
    final List<Classifier.Recognition> results4 = activity.classifyImage(bitmap4);
    Log.v(TAG, "Scores for image A: "+results4.toString());

    Assert.assertEquals("h", results1.get(0).getTitle());
    Assert.assertEquals("q", results2.get(0).getTitle());
    Assert.assertEquals("p", results3.get(0).getTitle());
    Assert.assertEquals("a", results4.get(0).getTitle());
}
```

Figura 5.8 Prueba del sistema encargado de clasificar imágenes

Como se puede apreciar, la prueba utiliza imágenes obtenidas del sistema de archivos en vez de utilizar la cámara, esto con el fin de obtener resultados consistentes para diferentes pruebas además de que el objetivo de este test en particular es el de verificar que el modulo encargado de la clasificación de imágenes se encuentre operando correctamente sin tomar en cuenta otros factores, como la integración de la cámara.

Además de verificar que las imágenes sean clasificadas correctamente la prueba genera algunas salidas mostrando el resultado de dicha operación y las probabilidades asignadas por la red neuronal para cada una de las categorías tal y como se muestra en la figura 5.9.

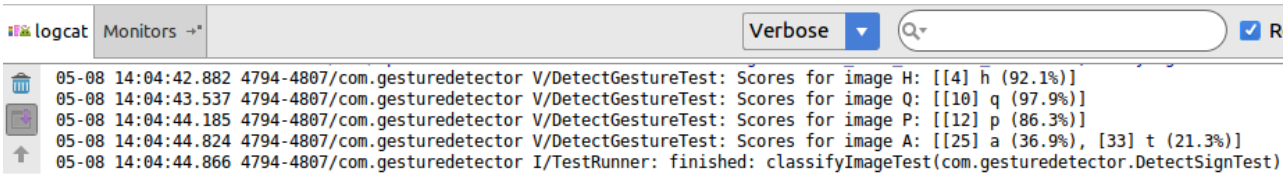

Figura 5.9 Resultado de la prueba de clasificación

Además de verificar que la aplicación sea capaz de clasificar las imágenes correctamente es necesario también realizar algunas pruebas para comprobar que la interfaz gráfica se encuentra funcionando correctamente.

Para lograr este fin se puede hacer uso de la biblioteca Espresso la cual permite crear pruebas para la interfaz gráfica de Android. Esta biblioteca permite verificar interacciones entre componentes y también el estado en el que se encuentran, por ejemplo permite comprobar si el contenido de un campo de texto ha cambiado después de presionar un botón.

La primera prueba que se realiza consiste en verificar que después de presionar el botón para capturar una imagen y que esta sea analizada por el sistema, el resultado se muestre en el campo de texto asignado para este fin.

Es necesario aclarar que debido a que las pruebas requieren el uso de la cámara del dispositivo los resultados del análisis de la imagen pueden variar de una prueba a otra, por lo que esta prueba se limita a verificar que se esté generando un resultado y éste se muestre en el campo de texto correspondiente, pero no verifica que el resultado de la clasificación sea el correcto.

Para este fin se utiliza el código mostrado en la figura 5.10

```java
@RunWith(AndroidJUnit4.class)
@LargeTest
public class GestureDetectionTest {

    private static final String TAG = "GestureDetectionTest";

    @Rule
    public ActivityTestRule mActivityRule = new ActivityTestRule<>(
            MainActivity.class);

    @Test
    public void captureImageAndShowResultTest() throws Exception{

        //Make sure the result textView starts with an empty string
        onView(withId(R.id.ResultGenerated)).check(matches(withText("")));

        //Use the camera to take a photo and process it.
        onView(withId(R.id.btnDetectObject)).perform(click());

        //Check if there is a result being shown in the result textView
        onView(withId(R.id.ResultGenerated)).check(matches(withText(containsString("Detectado:"))));
    }
}
```

Figura 5.10 Prueba para verificar el funcionamiento de la interfaz grafica

Si la prueba se realiza de manera satisfactoria despliega un mensaje tal y como se muestra en la figura 5.11

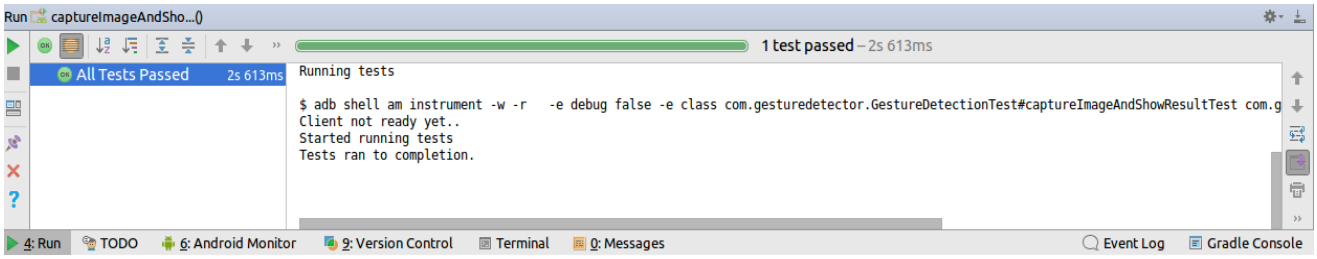

Figura 5.11 Resultado de la prueba en la interfaz grafica

Posteriormente se procede a realizar una prueba para verificar que la navegación de la aplicación se encuentre operando correctamente, para ello la prueba realizada primero se ubica en la pantalla principal de la aplicación, es decir, la encargada de realizar la identificación de las señas, después abre el navegador lateral y selecciona la opción que permite navegar hacia la pantalla que muestra la lista de los signos utilizados en el lenguaje de señas, una vez ahí, abre el navegador lateral nuevamente y regresa hacia la pantalla principal.

El código utilizado para realizar esta prueba se muestra en la figura 5.12

```java
@Test
public void showDrawerAndNavigateTest(){

    //Open drawer from the main activity
    onView(withId(R.id.drawer))
            .check(matches(isClosed(Gravity.START)))
            .perform(open());

    //Navigate to the showSigns activity
    onView(withId(R.id.nav_view))
            .perform(NavigationViewActions.navigateTo(R.id.nav_drawer_signs));

    //Open drawer from the ShowSigns activity
    onView(withId(R.id.drawer_ss))
            .check(matches(isClosed(Gravity.START)))
            .perform(open());

    //Navigate back to the main activity
    onView(withId(R.id.nav_view_ss))
            .perform(NavigationViewActions.navigateTo(R.id.nav_drawer_read));
}
```

Figura 5.12 Prueba de navegación de la aplicación

Nuevamente, la prueba se realiza de manera correcta, tal y como se muestra en la figura 5.13.

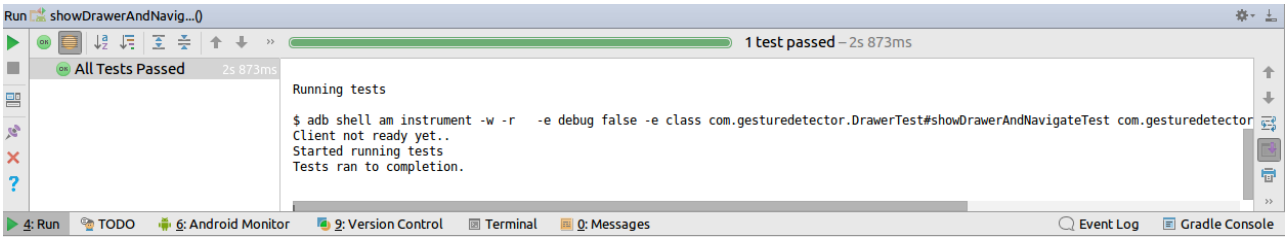

Figura 5.13 Resultado de la prueba de navegación de la aplicación

Además de verificar el funcionamiento de la aplicación con imágenes obtenidas del sistema de archivos, también se realizaron pruebas en dispositivos físicos utilizando la cámara del mismo para obtener las imágenes.

Derivado de estas pruebas fue posible observar que algunos aspectos influyen de manera considerable en la exactitud con la que se realiza la clasificación de las imágenes obtenidas.

El aspecto más importante a tener en cuenta es que, como se mencionó anteriormente, las imágenes utilizadas son en general muy similares entre sí por lo que para que la aplicación realice la clasificación de manera correcta es necesario que la imagen obtenida muestre la mano de forma clara y sobre todo que la señal realizada

sea lo más apegada posible al estándar, de otro modo, pequeñas variaciones pueden confundir al sistema y generar un resultado incorrecto.

Al igual que se muestra en la tabla de confusión, los signos que resultaron más complicados de reconocer para la aplicación fueron aquellos que tienen mayor similitud entre sí, como es el caso de la letra “o” y el numero “0”.

El ambiente en el cual se captura la imagen también puede influenciar el resultado de la clasificación ya que si el fondo contiene elementos que puedan ser interpretados por la aplicación como contornos correspondientes a manos o dedos el resultado de la clasificación podría no ser el idóneo.

Finalmente, otra variación que puede confundir al sistema es el uso de elementos que alteren la forma de la mano, como puede ser el caso de guantes o de joyería demasiado ostentosa.

Capítulo 6

CONCLUSIONES

En el presente trabajo se ha desarrollado una herramienta cuya finalidad es la de facilitar la comunicación para las personas que padecen de pérdida de audición incapacitante permitiendo que cualquier persona sea capaz de entender el significado de algunos de los signos comúnmente utilizados en los lenguajes de señas.

La herramienta creada consiste en una aplicación móvil para el sistema operativo Android, tomando ventaja de que la gran mayoría de las personas cuentan con un dispositivo móvil y que de estos la mayoría cuenta con este sistema operativo además de que estos dispositivos cuentan con los todos los elementos de hardware que requiere la aplicación para su correcto funcionamiento.

Para realizar el análisis de las imágenes se utiliza una red neuronal profunda ya que este tipo de sistemas han demostrado buenos resultados para este tipo de tareas, sin embargo, se observa que debido a la complejidad de la red neuronal utilizada se requiere de una gran cantidad de tiempo, recursos de hardware y de datos para su correcto entrenamiento.

El problema anterior se resuelve utilizando la técnica de transferencia de aprendizaje la cual permite tomar una red neuronal previamente entrenada con una gran cantidad de datos y modificarla de tal manera que sea capaz de trabajar con un nuevo conjunto de datos.

También fue posible apreciar cómo a pesar de que los dispositivos móviles normalmente cuentan con una limitada capacidad de procesamiento estos son capaces de utilizar una red neuronal previamente entrenada para realizar la clasificación de imágenes.

Lo anterior es gracias a que si bien es cierto que una red neuronal requiere de una cantidad considerable de recursos de hardware para su entrenamiento, el modelo generado al concluir el proceso no tiene requerimientos demasiado altos para su utilización.

Otro factor a considerar durante la creación de la aplicación móvil es que el sistema operativo Android se encuentra presente en una gran cantidad de dispositivos móviles los cuales en ocasiones cuentan con muy diversas configuraciones de hardware, es decir, diferentes capacidades procesamiento, diferentes tamaños de pantalla, algunos elementos de hardware no se encuentran disponibles en todos los dispositivos, etc.

Además, se debe considerar la fragmentación de mercado que existe para este sistema operativo, todo esto puede provocar que la creación de una aplicación se complique, por ello es necesario conocer a qué tipo de dispositivo está dirigida la

aplicación y establecer dentro de los parámetros de la misma que requerimientos mínimos se esperan del dispositivo móvil de modo que la aplicación no sea instalada en aquellos dispositivos que no cuenten con los requerimientos para ejecutarla.

Finalmente, la herramienta creada sirve para mostrar la manera de entrenar e integrar un sistema de inteligencia artificial (red neuronal profunda) con una aplicación móvil sin necesidad de utilizar un servidor remoto para realizar el análisis de los datos.

Capítulo 7

LIMITACIONES DEL SISTEMA

La herramienta creada tiene como finalidad la interpretación de algunos de los signos utilizados en el lenguaje de señas para lo cual requiere de obtener imágenes individuales para su posterior análisis.

Lo anterior implica que la herramienta no es capaz de interpretar aquellos signos que requieren de una serie de movimientos.

Esta limitante aplica también para algunos caracteres utilizados en el lenguaje de señas, particularmente las letras J y Z, las cuales no son posibles de detectar correctamente con esta aplicación.

Otra limitante del sistema es que al requerir de imágenes individuales se requiere capturar carácter por carácter para poder obtener una frase o palabra, si bien la aplicación cuenta con un espacio en el cual se muestran los caracteres que han sido reconocidos recientemente esto sigue siendo una limitante que complica el uso de la herramienta.

Por otra parte, la herramienta requiere que las imágenes capturadas sean lo más precisas posible ya que si presentan una variación considerable con respecto a los signos que se utilizaron como ejemplo para el entrenamiento de la red neuronal esta producirá resultados inexactos.

Para mejorar la herramienta creada en el presente trabajo se propone que el sistema tenga la capacidad de analizar videos cortos, esto con la finalidad de que se pueda reconocer un mayor número de caracteres.

Una tecnología que ha mostrado buenos resultados en el análisis de videos y de otros datos en los que se requiere tener en cuenta la secuencia de los mismos son las redes neuronales recurrentes, particularmente aquellas que utilizan la arquitectura LSTM (Long Short-Term Memory).

Capítulo 8

FUENTES Y REFERENCIAS BIBLIOGRÁFICAS

[1] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, “Rethinking the Inception Architecture for Computer Vision”. arXiv:1512.00567v3. 2015.

[2] Barczak, A.L.C., Reyes, N.H., Abastillas, M., Piccio, A., Susnjak, T., “A new 2D static hand gesture colour image dataset for ASL gestures”, Research Letters in the Information and Mathematical Sciences, 15, 12-20. 2011.

[3] Deafness and hearing loss. (2017). World Health Organization. Recuperado el 2 de Abril de 2017, de http://www.who.int/mediacentre/factsheets/fs300/en/

[4] Sign Language. (2017). WFD. Recuperado el 2 de Abril de 2017, de https://wfdeaf.org/human-rights/crpd/sign-language/

[5] CS231n Convolutional Neural Networks for Visual Recognition. (2017). Cs231n.github.io. Recuperado el 15 de Abril de 2017, de http://cs231n.github.io/transfer-learning/

[6] Computer Basics: Mobile Devices. (2017). GCFLearnFree. Recuperado el 7 de Abril de 2017, de https://www.gcflearnfree.org/computerbasics/mobile-devices/1/

[7] Topic: Smartphones (2017). Statista. Recuperado el 8 de Abril de 2017, de https://www.statista.com/topics/840/smartphones/

[8] Topic: Tablets (2017). Statista. Recuperado el 8 de Abril de 2017, de https://www.statista.com/topics/841/tablets/

[9] Topic: App stores (2017). Statista. Recuperado el 8 de Abril de 2017, de https://www.statista.com/topics/1729/app-stores/

[10] Why are people buying fewer tablets? The Guardian. Recuperado el 8 de Abril de 2017, de https://www.theguardian.com/technology/2015/feb/03/why-are-people-buying-fewer-tablets

[11] IDC: Smartphone OS Market Share. (2017). IDC. Recuperado el 12 de Abril de 2017, de http://www.idc.com/promo/smartphone-market-share/os

[12] Dashboards. (2017). Android Developers. Recuperado el 12 de Abril de 2017, de https://developer.android.com/about/dashboards/index.html

[13] Platform Architecture (2017). Android Developers. Recuperado el 13 de Abril de 2017, de https://developer.android.com/guide/platform/index.html

[14] Tensorflow demystified. (2017). Chatbot's Life. Recuperado el 16 de Abril de 2017, de https://chatbotslife.com/tensorflow-demystified-80987184faf7

[15] TensorFlow and deep learning, without a PhD. (2017). Codelabs.Developers. Recuperado el 16 de Abril de 2017, de https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/

[16] nightly-android [Jenkins]. (2017). Tensorflow. Recuperado el 22 de Abril de 2017, de https://ci.tensorflow.org/view/Nightly/job/nightly-android/

[17] Using Transfer Learning to Classify Images with TensorFlow. (2017). Medium. Recuperado el 17 de Abril de 2017, de https://medium.com/@st553/using-transfer-learning-to-classify-images-with-tensorflow-b0f3142b9366

[18] Tensorflow models. (2017). GitHub. Recuperado el 20 de Abril de 2017, de https://github.com/tensorflow/models/tree/master/inception

[19] Installation Guide Linux CUDA Toolkit Documentation. (2017). Nvidia. Recuperado el 20 de Abril de 2017, de http://docs.nvidia.com/cuda/cuda-installation-guide-linux/#axzz4VZnqTJ2A

[20] TensorFlow: How to freeze a model and serve it with a python API. (2017). Metaflow. Recuperado el 20 de Abril de 2017, de https://blog.metaflow.fr/tensorflow-how-to-freeze-a-model-and-serve-it-with-a-python-api-d4f3596b3adc

[21] How to Retrain Inception's Final Layer for New Categories. (2017). TensorFlow. Recuperado el 20 de Abril de 2017, de https://www.tensorflow.org/tutorials/image_retraining

[22] Tensorflow Android. (2017). GitHub. Recuperado el 20 de Abril de 2017, de https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android

[23] Build a Material Design App with the Android Design Support Library. (2017). Codelabs.Developers. Recuperado el 15 de Abril de 2017, de https://codelabs.developers.google.com/codelabs/material-design-style

[24] Essentials of Machine Learning Algorithms (with Python and R Codes). Analytics Vidhya. Recuperado el 15 de Abril de 2017, de https://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms

[25] Machine Learning. Coursera. Recuperado el 25 de Febrero de 2017, de https://www.coursera.org/learn/machine-learning/home/welcome

[26] Neural Networks. OpenNN. Recuperado el 12 de Abril de 2017, de http://www.opennn.net/

[27] Artificial Neural Networks Architectures And Applications. Intechopen. Recuperado el 12 de Abril de 2017, de https://www.intechopen.com/books/artificial-neural-networks-architectures-and-applications/applications-of-artificial-neural-networks-in-chemical-problems