UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Estudio de herramientas de detección de imágenes con aplicación de ejemplo

Presentado por Bryan Reinoso Cevallos
en Universidad de Burgos — Julio de 2015
Tutores: Dr. José Francisco Díez Pastor,
Dr. César I. García Osorio

Resumen

Descriptores

Abstract

Keywords

Índice general

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

Este proyecto tiene como objetivo el estudio teórico de herramientas de *machine learning*. Las herramientas que se quieren estudiar tendrán como funcionalidad el tratamiento de imágenes y la extracción de una predicción a partir de estas.

Como un segundo objetivo, se pretende crear una pequeña aplicación de prueba. Que usará una o varias herramientas de las estudiadas para mostrar el funcionamiento de las mismas.

La aplicación de soporte para los usuarios será desarrollada en Android y constará de una interfaz y funcionamiento orientada al uso por personas con dificultades de visión, además de guiará al usuario a través de la aplicación con un asistente de voz que irá diciendo al usuario qué debe hacer y en qué punto de la aplicación se encuentra.

La aplicación tendrá como objetivo inicial mandar una foto tomada por el usuario a un servidor que la procesará y extraerá una predicción que le dirá al usuario lo que se puede observar en la imagen que él ha elegido. La predicción será una frase que describa la imagen y está será leída por la aplicación.

Por otra parte en el lado del servidor usaremos algoritmos de *machine learning* para el procesado de la imagen y la extracción de la predicción. El servidor recibirá una petición de tipo POST y cargará la imagen, la cuál será procesada y en respuesta devolverá una frase en español que describa la imagen.

El principal objetivo del proyecto es conseguir crear un prototipo de esta aplicación que tenga un correcto funcionamiento y que sirva como base para desarrollar sobre él una aplicación más optimizada y de aún mejor funcionamiento.

2. OBJETIVOS DEL PROYECTO

2.1 Objetivos funcionales

El principal objetivo del proyecto es tener un prototipo de la aplicación que funcione de manera adecuada, para esto tenemos que tener en cuenta tanto los objetivos del lado del servidor como del lado del cliente.

2.1.1 Objetivos funcionales en el cliente Android

El principal objetivo que podemos encontrarnos en el lado del cliente es el disponer de un prototipo de aplicación Android con la que el usuario pueda mandar de manera intuitiva una imagen al servidor y recibir de él la predicción esperada. Para que esto funcione de manera adecuada se han propuesto una serie de requisitos que deben cumplirse:

- Para poder enviar la imagen al servidor y recibir de él la predicción, se debe establecer una conexión con él desde la aplicación Android y hacer un correcta petición de tipo POST, la cual subirá la imagen al servidor para ser procesada . Para la realización de esto se procederá al estudio de la documentación Android, buscando la manera más sencilla y adecuada para nuestro caso de uso.
- Para que la persona que lo utilice no tenga por qué saber la estructura de la interfaz gráfica de la aplicación y poder usarla sin problemas, nos apoyaremos sobre los eventos ontouch de Android, evitando de esta manera la dependencia de la interfaz gráfica para que la aplicación pueda llegar a ser usada por personas con dificultades visuales.
- Para ofrecer una guía fácil y útil para el usuario a lo largo de su experiencia usando la aplicación, usaremos la librería Text2Speech para ir guiando al usuario con instrucciones en voz alta, explicándole lo que debe hacer en cada momento. Finalmente le devolveremos la predicción y se le dirá cuáles son sus opciones.
- Para poder tomar la imagen se requerirá de un dispositivo con cámara de fotos y se tendrá que establecer los permisos sobre la aplicación para el uso de la misma.

2.1.2 Objetivos funcionales en el servidor

En el lado del servidor tenemos un objetivo claro y es el de recibir una imagen a través de una petición POST desde un cliente, procesarla y, finalmente, devolver la predicción en español. Para conseguir todo esto tenemos una serie de requisitos y objetivos que tenemos que cumplir:

- Para poder recibir las peticiones y mandar respuestas tenemos que tener un servidor con la capacidad de gestionar este tipo de operaciones, para ello se deberá proceder al estudio de las herramientas disponibles de predicción y buscar un *framework* de programación de servicios web, que se adapte mejor a nuestro caso de uso.

- Para procesar la imagen, que es un objetivo principal en el servidor. Se estudiará el funcionamiento de distintas herramientas de *deep learning*, eligiendo las que más interesantes resulten y que se puedan usar en el servidor.
- Finalmente, la predicción, que hayamos obtenido tras el procesado de la imagen, será devuelta como respuesta a la petición POST.

2.2 Objetivos a nivel teórico

Este proyecto conlleva una carga bastante grande en cuanto al estudio teórico de las herramientas que se van a proceder a usar y, con ello, una serie de objetivos a nivel teórico a tener en cuenta.

Como primer y objetivo básico del proyecto a nivel teórico sería el correcto estudio y entendimiento de las herramientas y protocolos se se van a usar. Esto implica un estudio bastante concienzudo de toda la información que aporta cada herramienta estudiada y su funcionamiento interno. El alumno debería ser capaz, al finalizar el proyecto, de aportar una explicación coherente y suficientemente razonada de cómo y por qué funcionan las herramientas estudiadas.

En segundo lugar se pretende que el alumno sea capaz de interpretar y entender artículos de investigación que estén relacionados con este tema. Estos artículos deberán ser posteriormente debidamente explicados en la documentación del proyecto. Aunque el entendimiento de estos artículos no será profundo, sí será lo suficientemente estricto como para poder realizar una presentación del funcionamiento de las herramientas.

En último lugar se pedirá una comparación crítica de las herramientas estudiadas y por qué unas son mejores o se entiende que funcionan mejor que las otras. O hacer una comparación de cómo está construida cada una y lo que diferencia sus arquitecturas.

3. CONCEPTOS TEÓRICOS

En este apartado se profundizará en los conceptos teóricos con los que se ha trabajado a lo largo de todo el proyecto.

3.1 Conceptos teóricos en el lado del Servidor

En este apartado se pretende explicar todos los conceptos teóricos que se utilizan en el lado del servidor, tanto directamente usados por el alumno como los que se usan en proyectos o librerías de apoyo que se usan en el proyecto.

3.1.1 Machine Learning

El *Machine Learing* o también conocido como aprendizaje computacional, entre otros nombres, es una rama de la inteligencia artificial que pretende conseguir el objetivo de que los computadores puedan aprender de manera automática. De forma más general se podría decir que en el *machine learning* se pretende crear programas informáticos capaces de generalizar comportamientos a partir de una información no estructurara que suele estar suministrada en forma de ejemplos.

El *machine learning* tiene muchísimas aplicaciones y es ahora mismo un campo de la inteligencia artificial que se encuentra en estudio y del que se suceden bastantes proyectos.

■ Tipos de algoritmos

Podemos dividir los algoritmos en tres tipos, aunque puede haber más pero los más importantes o con los que se puede clasificar a todos los algoritmos o programas que nos encontremos:

- **Aprendizaje Supervisado:** Este tipo de algoritmos crea una función que relaciona las entradas al sistema con las salidas del mismo. Aquí tendremos dos tipos de problemas a resolver, los cuáles son:
  - Clasificación: El problema de clasificación se trata de que a través de un conjunto de datos, entrenamos nuestro modelo para que este devuelva como resultado una clase que clasifique al valor de entrada. Internamente estamos creando una función que nos devolverá la clase perteneciente a cada ejemplo con el menor error posible. Ver 3.1 [1][**?**].

    Ejemplo: Tenemos un conjunto de datos con una estructura del tipo (altitud, presiónALTA, BAJA). El modelo será entrenado para recibir un valor cualquiera de altitud, y este devolverá o bien, clase0 = Presión baja o, clase1= Presión alta. Lo que el modelo está preparado para devolver es una clase.

[1] https://en.wikipedia.org/?title=Machine_learning

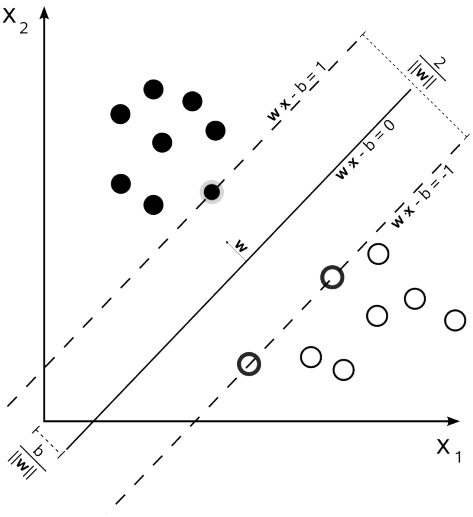

Figura 3.1.: Ejemplo de clasificación.

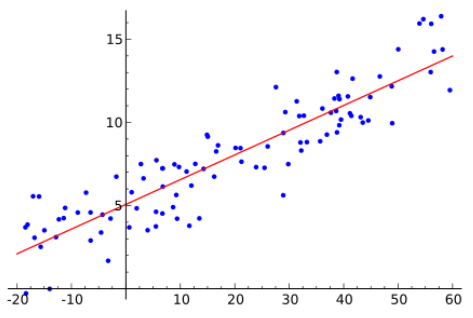

Figura 3.2.: Ejemplo de regresión lineal.

- Regresión: Este problema es bastante similar al de clasificación, pudiendo llegar a considerarse al de clasificación como un tipo de regresión. La principal diferencia de este con el de clasificación es que no esperamos una clase, sino un valor devuelto por la función construida, donde dicho valor será la predicción correspondiente al ejemplo. Internamente también creamos una función, pero esta está diseñada para devolver un valor numérico intentando predecir el estado del ejemplo, en función de los parámetros de entrada. Osea, la regresión nos devolverá un número como resultado, mientras que la clasificación devolverá una variable categórica.Ver imagen 3.2[2].

  Ejemplo: Tenemos un conjunto de datos del tipo (Altitud,Presión), en este caso tanto la altitud como la presión toman valores numéricos reales. El modelo se preparará para buscar una función que se ajuste mejor a los datos de entrenamiento. Ante un ejemplo para predecir, el modelo devolverá un número real, que será el valor esperado de la presión a esa altitud y no una clase.

- **Aprendizaje no Supervisado:** En estos tipos de algoritmo se lleva a cabo el modelado con una serie ejemplos que tan sólo constan con sus valores de entrada, mientras que el sistema desconoce a qué clase o qué salida debiera surgir por cada ejemplo. Esto obliga al algoritmo a tener la capacidad de reconocimiento de patrones y ser capaz de diferenciar entre los ejemplos y dividirlos en grupos, en el que la salida será el grupo al que pertenece cada ejemplo.

- **Aprendizaje semisupervisado:** Este tipo de algoritmos son una combinación de los dos anteriores, tiene en cuenta tanto los ejemplos etiquetados como los no etiquetados.

Dentro de este tipo de algoritmos, que son los que serán usados en el proyecto; podemos identificar los más significativos y relacionados con el proyecto:

- Máquinas de vectores de soporte: Se trata de un conjunto de algoritmos de aprendizaje supervisad, que son capaces de predecir la clase de un ejemplo nuevo que entre en el modelo. Aunque este tipo de algoritmos pueden ser usados tanto para regresión como para clasificación.

  En lo que se basan estos algoritmos es en que una máquina de vectores de soporte constituye un hiperplano, el cuál posee una dimensión muy elevada. Dentro de ese hiperplano se producen divisiones resultantes del entrenamiento del modelo, las cuáles cuanto mejor dividan el espacio, mejor será la clasificación[3].

- Árboles: Los árboles de decisión son unos de los primeros métodos del *machine learning*. Estos árboles están compuestos por una serie de nodos internos de decisión y unos nodos hojas, que se corresponden con la predicción o el resultado del modelo.[4].

- Redes neuronales: Es un conjunto de algoritmos de aprendizaje tanto supervisado como no supervisado, los cuales usan el concepto biológico de la neurona y de las interconexiones neuronales para crear un modelo de neurona artificial y las redes neuronales. La neurona artificial tendrá una serie de entradas, que son procesadas por la función de activación de la neurona y devuelve una salida que será el parte o el resultado del modelo o bien podría ser la entrada a otra neurona, formando con ello grandes redes den neuronas artificiales.

[2] https://es.wikipedia.org/wiki/Regresión_lineal
[3] https://es.wikipedia.org/wiki/Máquinas_de_vectores_de_soporte
[4] https://regularizer.wordpress.com/2014/11/18/deep-learning-with-decision-trees/

3.1.2 Deep Learning

Se trata de un conjunto de algoritmos cuyo objetivo es intentar modelar abstracciones de alto nivel sobre los datos, usando para ello arquitecturas compuestas. Estas arquitecturas son de transformaciones no lineales y múltiples.

La definición de aprendizaje profundo o *machine learning* no está muy clara, ya que existe más de una definición. Por norma general, hace referencia a algoritmos centrados en el aprendizaje de manera automática. Aún teniendo este punto en común, podemos encontrar diferentes algoritmos cuyas características los puede clasificar de la siguiente manera:

- Usar un conjunto de capas en forma de cascada, o puestas de manera consecutiva una tras de otra, lo que significa que la salida de una capa será la entrada de la capa posterior. Los algoritmos de este tipo se enmarcan dentro del aprendizaje supervisado como en el no supervisado, esto implica la necesidad de que algunos algoritmos tengan la capacidad de detectar patrones.
- estar basados en el aprendizaje (no supervisado) de múltiples niveles de características o representaciones de datos. Las características de más alto nivel se derivan de las características de nivel inferior para formar una representación jerárquica [**?**].
- aprender múltiples niveles de representación que corresponden con diferentes niveles de abstracción. Estos niveles forman una jerarquía de conceptos [**?**].

Los algoritmos de aprendizaje profundo contrastan con los algoritmos de aprendizaje poco profundo por el número de transformaciones aplicadas a la señal mientras se propaga desde la capa de entrada a la capa de salida. Cada una de estas transformaciones incluye parámetros que se pueden entrenar como pesos y umbrales. No existe un estándar de facto para el número de transformaciones (o capas) que convierte a un algoritmo en profundo, pero la mayoría de investigadores en el campo considera que aprendizaje profundo implica más de dos transformaciones intermedias.

3.1.3 Servicio Web

Se trata de una tecnología que nos permite el correcto intercambio de datos entre distintas aplicaciones, para esto usa una serie de protocolos y estándares. El objetivo principal al usar este tipo de tecnología es conseguir que aplicaciones que trabajan en con distintos software, distinto idioma de programación, distinta localización e incluso con distinta plataforma de instalación; puedan comunicarse de manera adecuada y correcta consiguiendo que el paso de datos de una a otra sea posible no sólo de manera adecuada sino, también, correcta. Para conseguir este objetivo, los servicios web utilizan estándares abiertos, osea, que es accesible para todos.[5]

Entre estos estándares los más relevantes con el proyecto y su desarrollo han sido:

- XML: El formato de este tipo de documentos lo ha llevado a ser muy usado, incluso llega a ser la base para la definición de otros estándares.
- SOAP: Protocolos para establecer el intercambio de datos entre distintas aplicaciones. Hace referencia al tipo de dato.
- WSDL: Lenguaje para los servicios web con el que establecemos la comunicación con estos, en el se especifican los datos del servidor y los servicios que este ofrece, determinando los

[5] https://es.wikipedia.org/wiki/Servicio_web

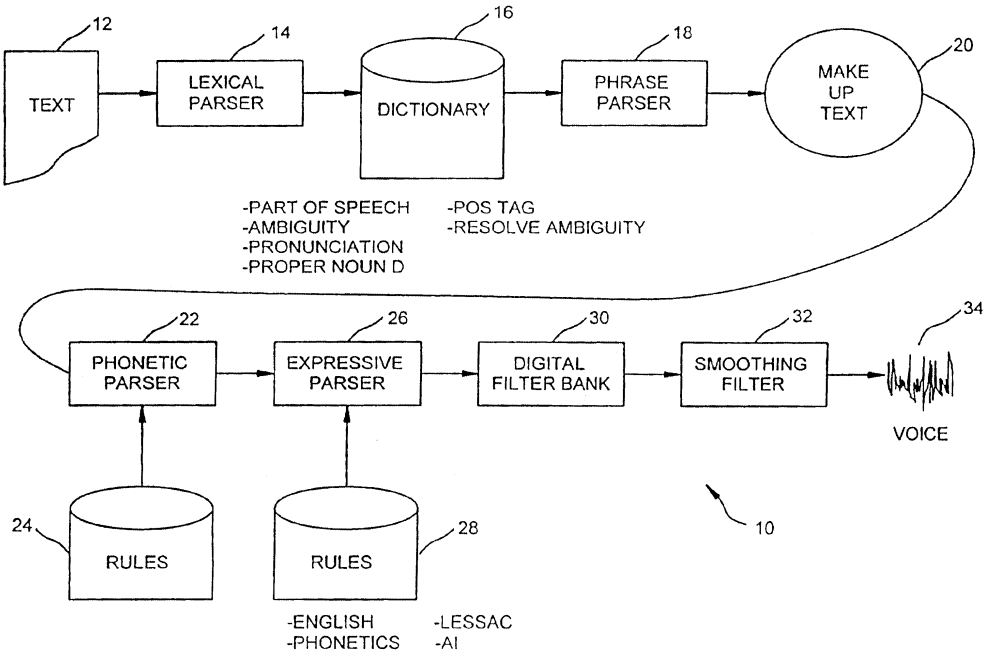

Figura 3.3.: Gráfico de librería *Text to Speech*

tipos de datos de respuesta y petición, los cuáles están establecidos en la especificación SOAP. Este lenguaje esta basado en XML.

- REST: Protocolo que usa Http para establecer la conexión con el servidor y que, gracias a los distintos tipos de peticiones que posee, puede realizar las distintas operaciones en función de lo servicios que aporte el servicio web.

3.2 Conceptos teóricos en el lado del Cliente

3.2.1 Funcionamiento de las librerías Text2Speech

Se tiene la intención de usar una librería para que el dispositivo lea las frases deseadas en voz alta, para esto se usa una librería del tipo *text to speech*. Este tipo de librerías lo que hace es procesar los datos que se quieren leer y convertirlo en un clip de audio, en el que podemos escuchar una voz que lee lo deseado.

Para entender el funcionamiento de este tipo de librerías se ayuda de un pequeño gráfico (3.3)[6], además se procede a poner una serie de pasos a través de los cuáles tienen que pasar los datos para llegar al clip final:

- En primer lugar se almacena el texto en la memoria del dispositivo en el que se va a procesar con esta librería.

[6] https://www.google.com/patents/US6865533

- Se procede a aplicar una serie de reglas de análisis léxico para convertir el texto en un conjunto de componentes de pluralidad.
- Se asocia la información de pronunciado y de significado a esta serie de componentes.
- El analizador fonético enmarca el texto usando las reglas de análisis fonético.
- Guardamos el conjunto de sonidos en memoria, cada sonido guardado es asociado con alguna información de pronunciación.
- Se recogen los sonidos asociados a los componentes para generar una fila o conjunto de sonidos que se asocian con el analizador fonético y con las reglas de análisis expresivo para generar la salida final.

3.3 Conceptos teóricos asociados a los artículos de investigación

En apartados posteriores en la documentación (ver 5) se explicará de manera más o menos detallada los artículos de investigación asociados a las herramientas que se han utilizado, dentro de estos artículos tenemos conceptos teóricos algo avanzados, que se considera oportuno explicar en esta apartado para el posterior entendimiento de dichos artículos.

3.3.1 Convolutional Neural Networks o Redes Neuronales Convolucionales

En este apartado de los conceptos teóricos se procederá a explicar las Redes Neuronales Convolucionales[7], las cuáles son muy importantes debido a que en el tratamiento de imágenes es una herramienta básica para el procesado de las mismas.

Las Redes Neuronales Convolucionales, CNNs a partir de ahora, son muy similares a las Redes Neuronales Multicapa. Poseen varias capas conectadas totalmente entre si, están formadas por un conjunto de neuronas artificiales, poseen pesos y bias que cambian en el entrenamiento de la red. Cada neurona recibe algunas entradas y obtiene el producto escalar de estas. La red entera posee una función objetivo. También posee una función de pérdida y se aplican todos los pasos normales que posee una Red Neuronal Multicapa común.

Parece que las CNNs no tienen nada nuevo en principio, pero la diferencia está en que la CNN lo que hace es dar por hecho que el input de la red es una imagen. Dar por hecho que la entrada de la red es una imagen nos aporta grandes beneficios, puesto que podemos configurar el resto de la red para tratar exclusivamente con este tipo de dato.

■ Visión Global de la arquitectura de las CNNs

Las CNNs toma ventaja de que el dato de entrada son imágenes y no otra cosa, lo que hace que la arquitectura de este tipo de redes sean restringidas de una manera más sensata. En particular, a diferencia de una red neuronal normal, las CNNs tienes sus neuronas organizadas en capas que conforman tres dimensiones:Altura, Anchura y Profundidad (la profundidad hace referencia al número de canales que tiene la imagen de entrada, pudiendo ser inclusive uno).

[7] http://cs231n.github.io/convolutional-networks/

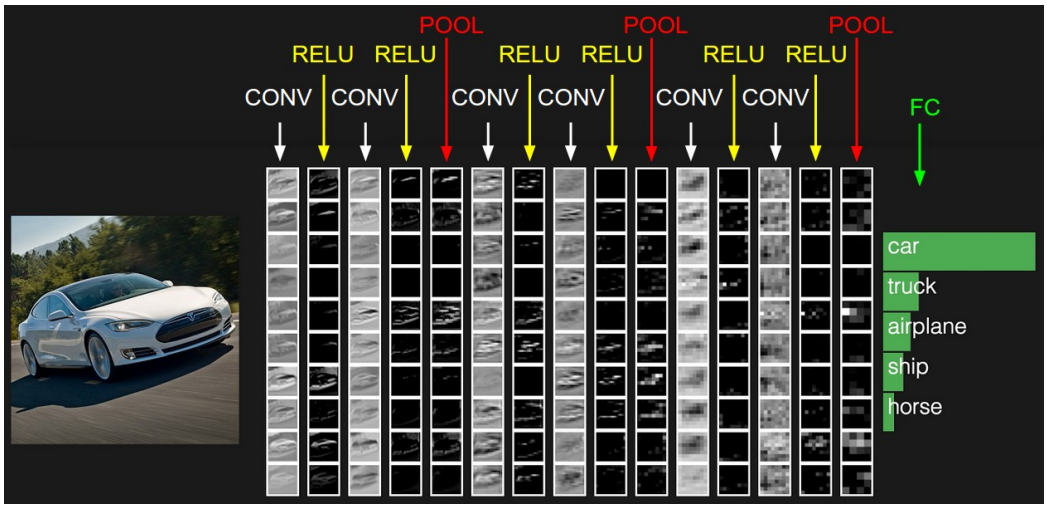

Figura 3.4.: Gráfico de una CNN [?]

■ Ejemplo de una CNN

Cada capa de una CNNN transforma un volumen de activaciones (la entrada de la capa, que pasa a través de las neuronas y su función de activación) es transformado en otro volumen con el uso de una función diferencial. Las CNNs tienen tres tipos principales de capas: **Capa Convolucional, Capa de puesta en común o de *Pooling* y Capa Completamente Conectada**. Si juntamos estas tras capas, entonces formamos una CNN completa.

Vamos a ver un pequeño ejemplo, aunque se detallará la explicación de las capas más adelante. Este ejemplo será para la clasificación de una y imagen y tiene una arquitectura [INPUT-CONV-RELU-POOL-FC] y unas 10 clases. La explicación de las capas será:

- **INPUT[32X32X3]:** Esta capa es la de entrada y contendrá los píxeles de la imagen que se vaya a procesar, la imagen tendrá una resolución de 32 de altura por 32 de anchura; además podemos ver que tendrá 3 canales, que podrían ser RGB.
- **CONV:** Esta capa computará las salidas de cada neurona que están conectadas a regiones locales de la entrada, cada computación realiza un producto escalar entre los pesos y la región a la que están conectadas en el volumen de entrada. El resultado dará una matriz de tamaño [32X32X12].
- **RELU:** Esta capa aplicará una función de activación, que podría ser como la función max(0,x) con umbral cero. Esta capa no afectará al tamaño de la matriz resultante.
- **POOL:** Esta capa hará un cómputo que reducirá la resolución de la matriz, lo que deja la matriz con tamaño [16X16X12].
- **FC:** (*fully-connected* o completamente conectada) Esta capa los resultados de clases, lo que provoca una matriz de tamaño [1X1X10], donde cada uno de los 10 números se corresponde con el resultado para cada clase.

Por lo tanto, la CNN transforma la imagen original en una matriz de resultados para cada clase, que contiene la probabilidad de que la imagen de entrada pertenezca a esa clase. Para verlo de manera gráfica podemos ver la imagen 3.4.

3.3.2 Recurrent Neural Networks o Redes Neuronales Recurrentes

En el siguiente apartado se procede a explicar las Redes Neuronales Recurrentes [8], RNN a partir de ahora. Esta clase de redes se usan en varios de los artículos que se procederá a explicar en el apartado de Estado del Arte (5).

Las RNNs se consideran un caso especial de redes neuronales también, esto se debe a que en general las redes neuronales tienen un conexión entre las capas de tal manera que la comunicación entre capas se hace sólo hacia delante. Por otro lado, las RNNs permiten conexiones recurrentes entre las distintas capas, esto significa que la información ya no viaja sólo en dirección hacia delante sino que existe una propagación de esta información hacia atrás (*back-propagation*). El hecho de permitir que exista este tipo de conexiones entre las distintas capas de la red neuronal añade un elemento temporal a la red, entonces esta puede predecir eventos adelante en el tiempo.

Cuando hablamos de una RNN, entonces estamos hablando de un tipo de red cuyo dato de entrada será siempre un vector, osea una secuencia de datos. La salida de esta red es también un vector de datos. En principio pudiera llegar a parecer extraño que tanto los datos de entrada como los de salida sean tratados como vectores o secuencias de datos, pero esto nos permite que la red procese los datos de una manera secuencial.

■ Visión global de la arquitectura de las RNNs

La arquitectura de las RNNs no tiene diferencia, en cuanto a capas se refiere, con las redes neuronales normales. La redes recurrentes poseen las mismas características que estas, estas tienen una función de activación dentro de la neurona, tienen capa de entrada, capas ocultas y capa de salida, poseen una serie de pesos que van cambiando en función del entrenamiento de la red.

La diferencia de una RNN respecto de una red neuronal común o básica se encuentra en el modo en el que están conectadas las neuronas con respecto al resto de neuronas. En las redes básicas las neuronas están conectadas solamente con las capas posteriores, lo que significa que la propagación de la información se hace únicamente hacia a delante sin que exista ningún tipo de retroalimentación. En cambio, las neuronas de una RNN posee también conexiones recurrentes, las conexiones recurrentes pueden ser de tres tipos, se puede ver de forma gráfica en la imagen 3.5:

- **1.-** Una conexión de una neurona de una capa A con otra neurona de la misma capa A
- **2.-** Una conexión de una neurona de una capa A con otra neurona de otra capa B, pero que se encuentra en un instante de tiempo anterior a esta, osea, se produce una propagación hacia atrás de los datos.
- **3.-** Una conexión de una neurona de cualquier capa con ella misma, una conexión a ella misma.

El hecho de añadir todos estas posibles conexiones también provoca que se añadan más pesos para formalizar la red y se necesite el uso de más memoria, además de que los pesos de las capas

[8] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

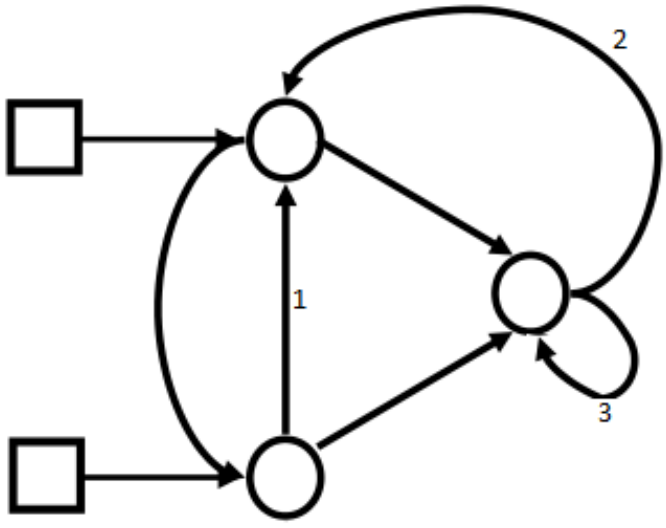

Figura 3.5.: Gráfico de conexiones recurrentes en una RNN

en una conexión recurrente son guardados en un estado intermedio al cuál acceden las neuronas objetivo de la conexión recurrente para producir predicciones temporales.

4. TÉCNICAS Y HERRAMIENTAS

En este apartado se indicarán las técnicas y herramientas utilizadas durante la realización del proyecto.

4.1 Técnicas de desarrollo

En esta sección se indicarán las técnicas de desarrollo utilizadas a lo largo del proyecto.

4.1.1 Metodología de desarrollo ágil

Para llevar a cabo este proyecto hemos seguido una metodología ágil de desarrollo de proyectos.

El uso de una metodología ágil de desarrollo viene justificada por el hecho de que los datos respecto a los últimos años, la inmensa mayoría de los proyectos informáticos que se desarrollaron con una metodología clásica de gestión de proyectos han fracasado o no se han llegado a terminar. Ante este problema se ha ideado una nueva e innovadora metodología de gestión de proyectos, esta es llamada metodología de desarrollo ágil.

La metodología de desarrollo ágil se tiene como objetivo el evitar el fracaso de los proyectos, para ello se pretende que los proyectos sigan un nuevo método de desarrollo llamado *Scrum*. Las principales características del *Scrum* son:

- Adopta una estrategia de desarrollo incremental en contra partida a la ejecución completa del producto que se da en la gestión clásica de proyectos.
- Basa la calidad del resultado más en el conocimiento de las personas en equipos autoorganizados, que en la calidad de los procesos empleados a lo largo del proyecto.
- En el desarrollo ágil podemos observar un claro solapamiento de las fases de desarrollo, mientras que en la gestión clásica estas se producen de manera secuencial o en cascada.

Al usarse la metodología de desarrollo *Scrum* podemos dividir el proceso en varias partes:

- En primer lugar el desarrollo ha sido guiado a través de iteraciones, que han sido delimitadas como *Sprints*. Estos *Spints* tienen una duración de entre 1 o 2 semanas.
- En cada *Sprint* se propone una serie de tareas y objetivos a completar antes de la finalización del mismo.
- Al final de cada *Sprint* se ha realizado una reunión en la que se establecía los objetivos que se han cumplido en ese *Sprint*, esto ha sido reflejado en la planificación del proyecto

como *Retrospective Meeting*; seguidamente se proponía los nuevos objetivos para el siguiente *Sprint*, esto esta representado en la planificación del proyecto como *Sprint planning*.

Creemos que gracias al uso de este tipo de metodología el proyecto tiene mayores posibilidades de terminar de manera exitosa.

4.1.2 Desarrollo Software con control de versiones

El control de versiones en desarrollo software se puede describir como la gestión de los cambios que se producen en nuestro software o en la configuración del mismo. Entendiéndose como versión el estado en el que se encuentra el software en un determinado momento.

En el proyecto se ha usado el control de versiones sobre el software a programar por parte del alumno. Para realizar el control de versiones se ha creado un repositorio público en GitHub[1].

En GitHub[1] las versiones se van determinando a través de *commits*. Lo *commits* son una actualización del estado actual del proyecto, estas actualizaciones llevan un título y un comentario para poder identificar los cambios del software y qué documentos han sido añadidos en qué *commit*.

En el proyecto la asiduidad de los *commits* no sigue ningún procedimiento estricto, sino que cuando se determinará que se ha producido un cambio en el software lo suficientemente importante se realizaría un *commit* con un comentario descriptivo del cambio que se ha producido en el software.

Lo interesante del uso de este tipo de metodología junto con la herramienta es que se pueden extraer gráficos y datos que nos aportan información bastante representativa de cómo ha ido evolucionando el proyecto en el aspecto software. También nos asegura que los cambios realizados no se pierden en caso de que la máquina falle o en caso de habernos equivocado al hacer un *commit* este puede ser deshecho. En conclusión el uso de control de versiones aporta muchas ventajas y datos con los que posteriormente podremos analizar más en profundidad la evolución del proyecto y usarlo como *feedback* para posteriores proyectos que se vayan a realizar.

4.2 Herramientas utilizadas

En este apartado se mostrará las distintas herramientas utilizadas para el desarrollo del proyecto.

4.2.1 Gestor de Tareas: VersionOne

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- **PivotalTracker**[2]

[1] https://github.com/
[2] http://www.pivotaltracker.com/

- **FogBugz**[3]
- **VersionOne**[4]

Se ha optado por la herramienta VersionOne[4], que ofrece unas condiciones notablemente mejores a las otras en su versión gratuita y además resulta bastante intuitiva y fácil de usar.

Con esta herramienta nos encargaremos de generar los Sprints del proyecto y se gestionará las tareas dentro de cada uno. Esta herramienta es usada en el desarrollo ágil, que es el tipo de desarrollo que se llevará acabo en el proyecto, además de que con la herramienta podemos, posteriormente, generar una serie de gráficos e informes que nos serán de gran ayuda a la hora de documentar el proyecto y de gestionar el avance del mismo.

4.2.2 Gestor de Versiones: GitHub

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- **GitHub**[1]
- **Bitbucket**[5]
- **Sourcefroge**[6]

Finalmente se decidió que se iba a usar la herramienta GitHub[1] porque se tenía experiencia previa en el uso de la misma, ofrece unas condiciones bastante razonables en su versión gratuita y se puede hace un buen seguimiento del proyecto con ella. Además de que es bastante sencilla la generación de commits en Windows porque dispone de una aplicación con la que lo commits se realizan de manera sencilla. Se puede ver de una manera bastante clara cómo ha ido avanzando el proyecto y se puede extraer unos gráficos muy útiles a la hora de documentar y determinar cómo ha avanzado el proyecto. Además al estar en su versión gratuita los miembros del jurado y cualquier persona que desee el acceso al proyecto sólo necesitará el link del mismo para poder verlo y comprobar la asiduidad de los commits y cómo ha sido la evolución del proyecto.

4.2.3 IDE de desarrollo: Android Studio

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- **Eclipse**[7]
- **Android Studio**[8]

La elección de Android Studio[8] ha sido porque no sólo es una herramienta exclusivamente dedicada a aplicaciones Android, sino que resultaba más prometedora que Eclipse; la cuál pensamos

[3] https://www.fogcreek.com/FogBugz/
[4] http://www.versionone.com/
[5] https://bitbucket.org/
[6] http://sourceforge.net/
[7] https://eclipse.org/
[8] http://developer.android.com/sdk/index.html

que puede quedar obsoleta para este tipo de aplicaciones. También vemos que Android Studio[8] ofrece un gestor de instalación de paquetes de *kit* de desarrollo y *plugins*, lo cuál puede resultar muy útil a la hora de la configuración del entorno de desarrollo. Además en Android Studio[8] no puede sólo usarse máquinas virtuales de móviles, sino que puedes conectar un dispositivo móvil al ordenador e ir ejecutando tu aplicación sobre el contando con un debugger y un logcat para errores, lo cual facilita en gran medida la programación.

4.2.4 Herramienta de Generación de Documentación: TEXMaker

Esta herramienta ha sido elegida por su interfaz gráfica, la cual es muy intuitiva y fácil de usar. Además de que es muy fácil de instalar y tiene ayuda en todo momento, pues autocompleta las etiquetas que quieras usar y las referencias. Además su consola de errores ayuda en gran medida a la hora de encontrar errores en tus documentos. También permite el uso de proyectos con más de un documento de tipo te, dándonos la opción de marcar uno de ellos como documento maestro.

Por debajo esta herramienta esta usando LATEX, que es un sistema para preparar documentos de manera sencilla y fácil. El objetivo de este sistema es que nosotros nos concentremos en escribir el contenido de manera adecuada y que no se nos olvide ningún detalle, dejando a LATEXque se encargue de optimizar el documento para que tenga la presentación más óptima y adecuada.

En conclusión, sin duda este tipo de sistema y herramienta es uno de los más adecuados para el trabajo que vamos a realizar.

4.2.5 Herramientas de Deep Learning: NeuralTalk

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- **Lib CCV**[9]
- **Overfeat**[10]
- **Deep Belief SDK**[11]

Esta herramienta ha sido elegida porque venía en el mismo idioma en el que se iba a programar el servidor, además de que su aplicación en nuestro proyecto era mucho mas práctico, pues devuelve una frase descriptiva de lo que en una imagen hay. Lamentablemente su funcionamiento e instalación son algo enrevesadas y requiere de una configuración que depende mucho de la máquina en la que se trabaje y de la estructura de directorios que esta tenga, pero una vez configurada su funcionamiento es el esperado.

Aunque cabe destacar que se uso la herramienta DeepBeliefSDK[11] en gran parte del proyecto, puesto que sus pruebas fueron correctas y funcionaba de manera bastante adecuada. Pero finalmente desistimos de su uso como herramienta principal debido a que al intentar combinarlo con la herramienta GSOAP[12], acabo no teniendo un correcto funcionamiento debido a que el módulo de

[9] http://libccv.org/post/with-a-sub-10-image-classifier-a-decent-face-detector-here-comes-ccv-0.7/
[10] http://cs.stanford.edu/people/karpathy/rcnn/
[11] https://github.com/jetpacapp/DeepBeliefSDK
[12] http://www.cs.fsu.edu/~engelen/soap.html

GSOAP[12] para Apache no era bueno y su documentación bastante floja. Esto está explicado de forma mucho más detallada en el apartado de Aspectos Relevantes (6.1.2).

4.2.6 Herramientas de Deep Learning: Caffe

Esta herramienta es usada en nuestro proyecto debido a que la herramienta NeuralTalk tiene dependencia de este proyecto para extraer características de las imágenes y luego poder realizar una predicción con ellas. Además este proyecto está bien estructurado y en si se sigue correctamente su documentación es relativamente fácil de instalar. Caffe es un arquitectura para el *deep learning* que está programada en C++ puro y en CUDA, pero también se puede usar en línea de comandos en un sistema Unix y cuenta con *wrappers* para Python y MATLAB. Sus principales características que lo convierten en un proyecto que destaca sobre los demás son:

- Es una arquitectura que funciona de manera especialmente rápida.
- Su código está bien testeado.
- Tiene buenas herramientas, modelos de referencia, demos y repositorios.
- Posibilidad de ejecutarlo tanto en CPU como en GPU

■ Anatomía de un modelo Caffe

La *deep networks* o redes profundas son modelos compositivos que se representand e forma natural como un conjunto o una colección de capas interconectadas que trabajan sobre fragmentos de datos. Caffe define una red capa a capa en su propio esquema de modelo. La red define el modelo entero desde abajo hasta arriba a partir de unos datos de entrada. Como los datos y sus derivados fluyen a través de la red en el *Forward and Backward*, Caffe guarda, comunica y manipula la información como burbujas: la burbuja es una matriz estándar y una interfaz de memoria unificada para el *framework*. La capa que sigue es tratada como la base tanto del modelo como de la computación. La red se considera como un conjunto de capas y de sus intercionexiones. Los detalles dentro de cada burbuja describen como la información será guardada y cómo esta será enviada a través de las distintas capas y redes.

La forma en que se resuelve la predicción es configurada a parte para desacoplar el modelo de la optimización del mismo.

4.2.7 Herramientas de programación: MATLAB

Tuvimos que instalar MATLAB porque NeuralTalk usa un script de MATLAB para poder preparar las imágenes para extraerles las características, además se usa el wrapper de caffe para MATLAB.

4.2.8 Herramientas de desarrollo de servidores: Flask

Se ha estudiado entre varias posibles herramientas, entre ellas están:

- **Axis2/C**[13]
- **GSOAP**[12]
- **Tomcat**[14]

Se ha escogido esta herramienta en concreto porque sobre todas las demás su funcionamiento era muy inmediato y además se escribe en Python, que es un idioma muy versátil y fácil de usar. El hecho de que esta herramienta tenga un funcionamiento y programación tan sencilla la hace una herramienta que, a nuestro parecer, destaca sobre el resto y es interesante trabajar con ella. Además tiene una documentación sencilla, repleta de ejemplos y explicaciones para poder realizar tu servidor. Sus ejemplos son muy útiles y funcionan a la primera, sin tener que configurar casi nada. Tiene una forma de establecer los URLs de forma muy sencilla y es fácil estructurar tu servidor con esta API. A pesar de ser una API tan fácil de usar tiene una gran potencia y se pueden construir con ella servidores bastante grandes y fácilmente escalables, por tanto se convierte en la herramienta perfecta para la programación de nuestro servidor.

4.2.9 Librería Text to Speech para Android

Android nos ofrece una librería de específica para este tipo de aplicaciones. En nuestra aplicación cliente hemos usado la librería *Text to Speech* que nos ofrece Android para realizar la lectura de mensajes para el cliente.

El funcionamiento de este tipo de librerías es parecido, se trata de un problema de procesadores de lenguajes. Primero se analiza la entrada a través de una serie de reglas para acabar asociando estas a unos sonidos de salida, estos acabarán constituyendo el clip de audio con la frase esperada. Esto se puede ver explicado de forma más detallado en el apartado de Conceptos Teóricos (ver 3.2.1).

4.2.10 Librería de Apache para conexiones Http para Android

Para establecer la conexión con el Servidor se ha usado la librería de Apache, que nos permite establecer una conexión con un servidor, a través del protocolo HTTP, de manera muy sencilla.

Se ha usado este tipo de librería debido a que el servicio web que se va ha programar usa el estándar REST para aportar los servicios y realizar el intercambio de datos con los distintos clientes que se conecten a este. Este tipo de conexión y la explicación de servicio web se puede ver más detallada en el apartado de conceptos teóricos (ver 3.1.3).

Está librería tiene una serie de elementos que se utilizan dentro de la aplicación Android para que todo funcione de manera correcta, los elementos son los siguientes:

- HttpClient: Este objeto se usa para establecer el cliente que ejecutará la petición. Es un objeto que te representa como cliente que se conecta al servidor.
- HttPost: Este objeto está hecho para determinar el tipo de operación que se va a solicitar al servicio web, en este caso es del tipo POST. Este objeto llevará asociada una cabecera con la que se establece la conexión.

[13] http://axis.apache.org/axis2/c/core/
[14] http://tomcat.apache.org/

- MultipartEntityBuilder: Este objeto maneja los datos que queremos enviar con nuestra petición, en nuestro caso será una imagen. Después de haberle añadido los datos a este objeto, podemos crear a partir de él un objeto de tipo HttpEntity.
- HttpEntity: Este objeto es una entidad que lo que realmente tiene en su interior es la parte de la cabecera de la petición donde van definidos los datos, osea los metadatos de la cabecera de una petición HTTP.
- HttpResponse En este objeto recibimos la respuesta que el servidor nos da tras ejecutar la operación que hayamos definido.

Como nota final cabe destacar que el procesamiento de todo esto deber hacerse en un hilo separado del hilo principal de ejecución de la aplicación debido a que los estándares de Android así lo establecen.

4.2.11 Librería de traducción de texto

En el lado del servidor se procede al uso de una librería de traducción porque la herramientas nos devuelven las cadenas sen Inglés, pero nosotros las queremos en Español.

En un principio se optó por intentar usar la librería de Google para la traducción de texto, pero nos encontramos con el inconveniente de que esta era de pago. Entonces, ante la búsqueda de una solución, se procedió a usar un proceso algo mas rudimentario pero igualmente válido. Como no se tenía acceso a una API de traducción se usaron los conceptos aprendidos de servicio web y de sus peticiones para simular una conexión a la página de traducción de Google y recibir de ella la cadena traducida.

El resultado se obtiene en un tipo de dato json pero este es fácilmente convertible a texto y de ahí se extrae la cadena ya traducida.

5. ESTADO DEL ARTE

En este apartado se procede a la presentación de una serie de artículos que han conformado el estudio teórico que conlleva este proyecto. el cuál tiene un gran peso en el mismo; pues el estudio de cada una de las herramientas que se han tenido en cuenta tiene por detrás un concienzudo estudio por parte del alumno, el cuál ha tenido que comprender el funcionamiento de las mismas a través de este estudio.

Para realizar la explicación de los artículos, estos serán traducidos de manera resumida, ordenada y eliminando los aspectos que sean demasiado avanzados o que no se consideren del todo necesaria su explicación en este proyecto.

5.1 Artículo del DeepBelief SDK

El artículo de investigación que se va a explicar es el que se ha usado para implementar la herramienta *Deep Belief SDK*[1] y se puede encontrar con el nombre de *ImageNet Classification with Deep Convolutional Neural Networks*[7]. Lo siguiente pretende ser un resumen explicativo del artículo, dónde se tratarán los temas más interesantes del mismo.

5.1.1 Introducción

Los enfoques actuales sobre el reconocimiento de imágenes ha hecho esencial el uso de técnicas de *machinne learning* para la resolución de este tipo de problemas. Para mejorar los rendimientos que actualmente se pueden encontrar, nosotros podemos recolectar conjuntos de datos más grandes, entrenar modelos más potentes y usar mejores técnicas para evitar el sobreajuste de nuestro modelo. Hasta hace poco se contaban con conjuntos de datos (*datasets*) relativamente pequeños, del orden de diez mil imágenes. Las tareas de reconocimiento simples pueden ser resueltas lo suficientemente bien con *datasets* de este tamaño. Pero ahora nos encontramos con tareas más complejas y tenemos la posibilidad de trabajar con *datasets* bastante más grandes. El *datasets* nuevo más largo está incluido en LabelMe[10] , el cuál consiste en un conjunto de imágenes de alta resolución con sus predicciones (*labels*) y clasificadas en más de veintidós mil categorías.

Para poder entrenar tal cantidad de datos es necesario un modelo lo suficientemente grande y capaz de predecir esa cantidad de categorías. De todas formas, la inmensa complejidad del reconocimiento de objetos significa que este problema no pude ser especificado incluso por un *dataset* tan largo como lo es ImageNet. Esto significa que tenemos que contar con conocimiento a priori para compensar todo los datos que no tenemos dentro del *dataset*. Un modelo que encaje en este tipo de descripción es, sin lugar a dudas, una Red Neuronal Convolucional (CNN)(ver sección 3.3). Su capacidad puede ser controlada variando su amplitud y su profundidad y, además, crean fuertes y, en su mayoría, supuestos correctos sobre la naturaleza de las imágenes.

[1] https://github.com/jetpacapp/DeepBeliefSDK

A pesar de las cualidades tan atractivas de las CNNs, estas no trabajan bien con imágenes de alta resolución porque resulta demasiado cara este tipo de ampliación. Afortunadamente, las GPUs (tarjetas gráficas) actuales vienen con una implementación altamente optimizada de convolución 2D, que son lo suficientemente potentes para facilitar el entrenamiento de CNNs particularmente grandes.

La res está entrenada con un subconjunto de datos de ImageNet usados en las competiciones ILSVRC-2010 y ILSVRC-2012[1] y se han logrado resultados lo mejores resultado con bastante diferencia de los mejores reportados en este conjunto de datos. Se escribió una implementaciónn altamente optimizada para GPU de convolución 2D y todo el resto de operaciones que internamente se necesitan con las CNNs. La red contiene algunas características inusuales que mejoran el rendimiento y reducen el tiempo de entrenamiento. El tamaño de la red convierte al sobreajuste en un problema bastante serio, para solucionar esto se usan métodos para prevenir el sobreajuste. La red final contiene cinco CNNs y tres capas totalmente conectadas, esta profundidad parece ser importante ya que si se aumenta o disminuye el número de CNNs, entonces el rendimiento se reduce.

Al fin y al cabo, el tamaño de la red está limitado por la cantidad de memoria disponible en las GPUs actuales y por la cantidad de tiempo de entrenamiento que estamos dispuestos a tolerar. La red tarda de cinco a seis días para ser entrenada sobre dos tarjetas gráficas GTX 580 3GB. Todos los experimentos apuntan a que los resultados pueden ser mejorados con la mejora de las GPUs, haciéndolas más rápidas, y con *datasets* más grandes.

5.1.2 El dataset o conjunto de datos

Se usa el un subconjunto del *dataset* ImageNet, que contiene sobre los quince millones de imágenes de alta resolución y están clasificadas con alrededor de veintidós mil categorías. El subconjunto que se usa es el que se utiliza en la competición llamada *ImageNet Large-Scale Visual Recognition Challenge*(ILSVRC), y este contiene uno coma dos millones de imágenes de entrenamiento, cincuenta mil imágenes de validación y ciento cincuenta mil imágenes de *test* o prueba.

ILSVRC-2010 es la única versión del ILSVRC que tiene disponibles los *labels* de las imágenes, por lo tanto este es el conjunto de datos sobre el que se han realizado la mayoría de los experimentos. eN ImageNet es posible mostrar los errores de dos formas: top-1 y top-5, dónde top-5 es el errror sobre las imágenes de test en el cual la predicción correcta no se encuentra dentro de las cinco clases más probables consideradas por el modelo.

ImageNet consiste en un conjunto de imágenes, cuya resolución es variable. Sin embargo, para el modelo, se necesitan imágenes con una resolución continua, osea, que todas las imágenes tengan la misma resolución (ver sección 3.3). Por lo tanto, se ha procedido a cambiar la resolución de las imágenes a una resolución común, 256X256. Las imágenes no se preprocesan de ninguna otra manera, excepto para extraer la actividad principal sobre el conjunto de entrenamiento a partir de cada pixel, por lo tanto, las imágenes son tratadas con los valores de sus filas RGB, se trabaja con los tres canales de color.

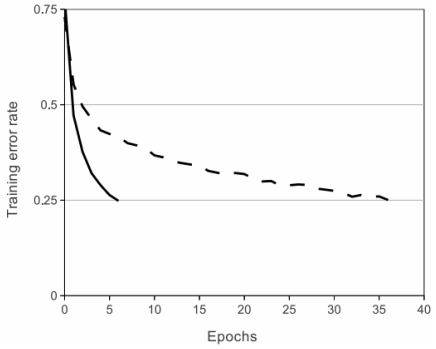

Figura 5.1.: Gráfico demostración de Saturación vs No Saturación

5.1.3 Arquitectura

Contiene en total ocho capas de aprendizaje, de las cuales son cinco convolucionales y tres son capas totalmente conectadas (*fully-connected*).

■ ReLU de no linealidad

El método normal para modelar la salida de una neurona como función aplicada sobre la entrada,

$$salida = f(entrada) \tag{5.1}$$

usando su función de activación, es con la función tangente.

$$salida = tangente(entrada) \tag{5.2}$$

En terminos de tiempo de entrenamiento con gradiente descendiente, estas saturaciones no lineales son mucho más lentas que si no usáramos saturamiento, con la función máximo.

$$salida = maximo(0, entrada) \tag{5.3}$$

Nos referimos a las neuronas con esta no linearidad como *Rectified Linear Units* (ReLUs), tal y cómo vemos en Nair and Hinton[8]. Las Redes Neuronales Convolucionales Profundas se entrenan en un tiempo considerablemente menor que las que usan la función tangente. Esto se demuestra en la imagen 5.1, dónde se ve que entrenando una red pequeña, se necesita un número de iteraciones menor para llegar al 25 % de error de entrenamiento si no usamos el modelo con neuronas con saturación.

Este trabajo no es el primero en considerar el uso de de modelos de neuronas diferentes a los tradicionales en las CNNs. Pero el objetivo de estos otros trabajos era distinto y el principal objetivo de este conjunto de datos es prevenir el sobreajuste, el objetivo era distinto al de hacer que la red se entrene de manera más rápida, lo que se pretende en este trabajo con sus ReLUs. El aprendizaje rápido tiene una buena influencia sobre el rendimiento sobre el entrenamiento de grandes modelos con grandes conjuntos de datos.

■ Entrenamiento en múltiples GPUs

El uso de una tarjeta gráfica GTX 580 que tiene sólo 3GB de memoria, limita el tamaño máximo de las redes que pueden ser entrenadas sobre esta. Si añadimos el problema de que con uno coma dos millones de ejemplos de entrenamiento son suficientes para que la red resultante sea demasiado grande como para que esta pueda ser entrenada sobre una sola GPU. Por lo tanto, el modelo ha sido entrenado sobre dos GPUs. Las GPUs actuales están bien preparadas para la paralelización, puesto que estas pueden acceder a la memoria de otra sin necesidad de pasar por la memoria principal del sistema. La paralelización usada en el modelo, básicamente entrena la mitad de neuronas en cada GPU, pero estaas solo pueden comunicarse con capas específicas de la otra GPU. Esto significa que si tenemos una capa 2 que se comunica con la capa 3 en su totalidad, estando estas dos en la misma capa, y tenemos una capa 4, esta se podrá conectar sólo con algunas neuronas de la capa 3, la capa 4 se encuentra en otra GPU. Elegir el patrón de comunicación es un problema para la validación cruzada pero hacerlo permite modificar la cantidad de comunicación hasta que esta equivalga una fracción aceptable de cantidad de computo.

Como resultado la arquitectura que se obtiene es una arquitectura similar la CNN "columnar" empleada por Cireşan [2],la cual tenía en su estructura capa opcional de preprocesado de imagen, capa convolucional, capa de tipo *Max-Pooling* y una capa de clasificación; pero la diferencia es que las columnas de este modelo no son independientes. Esto reduce el error top-1 en 1,7 % y el error top-5 en 1,2 %.

■ Respuesta local a la Normalización

Lo más destacable de este apartado es que como se utilizan neuronas de tipo ReLU, la normalización no es necesaria para evitar el saturamiento de las neuronas.

■ Overlapping Pooling o Puesta en común superpuesta

El trabajo de las capas de *pooling* o puesta en común, es la de resumir las salidas de los grupos de neuronas vecinos que le corresponde. Tradicionalmente los vecindarios de neuronas al ser resumidos no se superponían. Para que quede más claro, una capa de *pooling* está formada por una cuadrícula de unidades de *pooling* que están separadas entre si por un número X de píxeles, y cada resumen realizado se hace sobre un vecindario de tamaño Y * Y. Si la separación entre unidades de *pooling* es igual que Y, entonces la capa de puesta en común es la tradicional; pero si nos encontramos con que la separación es menor que el valor Y, entonces las unidades de *pooling* se superponen al resumir vecindarios y así obtenemos una capa de puesta en común superpuesta(ver 5.2.

En conclusión se obtiene que el error en top-1 y top-5 se reduce en 0.4 % y 0.3 %, respectivamente. Por eso esta red usa un valor de dos para X y de tres para Y.

■ Arquitectura

Como se ha dicho antes la arquitectura de la red está formada por ocho capas, de las cuáles las cinco primeras con convolucionales y las tres restantes son tres capas completamente conectadas. La salida de la última capa completamente conectada es una matriz de 1000 datos, que se corresponde con las 1000 *labels* de clase que tenemos.

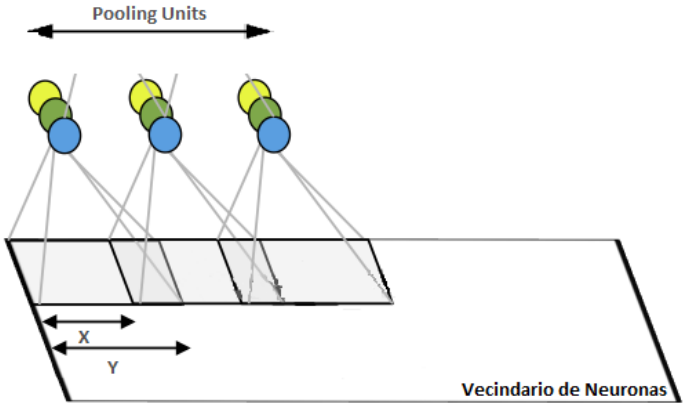

Figura 5.2.: Overlapping Pooling

Las capas dos, cuatro y cinco convolucionales están únicamente conectadas con la capa anterior, la cual está situada en la misma GPU. La capa 3, por el contrario, está completamente conectada con la segunda capa. Capas de normalización están después de la primera y segunda capa convolucional. Capas de *pooling* están después de las de normalización y después de la quina capa convolucional. La ReLU, función de no linealidad, se aplica en todas las capas, tanto convolucionales como las completamente conectadas.

■ Reduciendo el sobreajuste

Debido a la cantidad de parámetros de la red neuronal, unos sesenta millones, y a la cantidad de clases que contiene el conjunto de datos, se hace imposible entrenar una red sin poder tener en cuenta la probabilidad de que se produzca un sobreajuste.

Para evitar el sobreajuste de la red hemos usado dos métodos:

- El primero es aumentar el conjunto de datos original, esto se hace creando nuevas imágenes a partir de las imágenes originales pero conservando su etiqueta. Las dos formas para hacer esto que se ha usado en este artículos son:

  1- Hacer traslaciones en la imagen y reflexiones horizontales a través de los cuáles obtenemos una nueva imagen.

  2- Se ha aumentado la intensidad de los canales RGB de la imagen para obtener otra imagen que sea distinta para el modelo.

- Se usa un nuevo método llamado *dropout* el que consiste en poner a cero la salida de las neuronas de la capa oculta con una probabilidad de 0.5. Las neuronas que sean abandonadas en este proceso, entonces no participaran en la propagación hacia atrás.

■ Detalles del aprendizaje

Se inicializan los pesos de cada capa a partir de una distribución Gaussiana con una derivación estándar de 0.01. Los bias de las neuronas en las capas dos, cuatro y cinco con una constante de 1.

Esta inicialización lo que hace es acelerar el proceso de aprendizaje en los pasos tempranos daando a las ReLUs entradas positivas. En el resto de capas los bias se inicializan cero.

Se ha usado el ratio de aprendizaje igual para todas las capas y este es ajustado de forma manual. La heurística que se ha seguido es que el ratio de aprendizaje es dividido entre diez cuanto el ratio del error de validación deja de mejorar con el ratio de aprendizaje actual. El ratio de entrenamiento se inicializa a 0.01. LA red ha sido entrenada en 90 ciclos con un conjunto de imágenes de 1.2 millones de tamaño, el cuál tomó 5 o 6 días en finalizar sobre las dos tarjetas gráficas ya mencionadas.

■ Resultados

El resultado sobre el dataset ILSVRC-2010 ha conseguido, sobre el conjunto de test, unos ratio de error top-1 y top-5 de 37,5 % y 17 %. Teniendo en cuenta que en la competición de 2010 se consiguió el mejor rendimiento como un 47,1 % y de 28.2 %, se puede decir que este modelo tiene bastante mejor resultados.

■ Conclusión

Las conclusiones más importantes que se pueden sacar del artículo es que una red neuronal convolucional profunda, siempre que sea lo suficientemente grande, es capaz de romper los récords sobre los resultados ya existentes. También es interesante el hecho de que si alguna de las capas convolucionales es quitada o añadida a este modelo, entonces su error ratio se incrementa; por alguna razón la profundidad de la red es importante a la hora de mejorar los ratio de error.

5.2 Artículo del NeuralTalk

En este apartado se procederá a resumir el artículo asociado a la herramienta NeuralTalk[6].

5.2.1 Introducción

Una vistazo rápido es suficiente para el humano para poder extraer una inmensa cantidad de detalles de la escena que este presenciando[5].Sin embargo está tarea es muy compleja para nuestros modelos de reconocimiento visual. Hasta ahora los esfuerzos en el reconocimiento de imágenes ha sido el de etiquetar imágenes a través de un conjunto fijo de categorías, el cuál ha progresado bastante. Sin embargo, estos métodos tienen un vocabulario muy restrictivo en comparación con el vocabulario descriptivo que tiene el ser humano.

Algunos pioneros se acercan a resolver el reto de generar descripciones de imágenes. Sin embargo, a estos modelos usualmente tienen ciertas deficiencias que les impide alcanzar una gran variedad en el reconocimiento. Por otra parte, el tema central de estos trabajos ha sido reducir la complejidad de las imagines en una sola sentencia.

Este trabajo tiene el objetivo de generar descripciones complejas a partir de una imagen. El principal reto de este trabajo es diseñar un modelo lo suficientemente rico para a la vez razonar sobre el contenido de la imagen y su representación en lenguaje natural. El segundo reto es el de encontrar un *dataset* lo suficientemente grande sobre el que trabajar.

La idea central es que podemos aprovechar estos grandes conjuntos de datos mediante el tratamiento de las frases como etiquetas débiles, en la que los segmentos contiguos de palabras corresponden a algunos en particular, pero la ubicación es desconocida en la imagen. Para esto, la contribución es doble:

- Se desarrolla un modelo de red neuronal que es capaz de relacionar los segmentos de frases con la región de la imagen que esta describiendo.
- Se introduce una arquitectura de red neuronal recurrente multimodal que es capaz de generar una descripcion en texto a partir de una imagen que toma como entrada.

5.2.2 El modelo

El objetivo final es generar descripciones de regiones de imágenes. Durante el entrenamiento, la entrada del modelo es un conjunto de imágenes y sus correspondientes *labels*, que son frases en lenguaje natural. En primer lugar se presenta un modelo que asocia fragmentos de oraciones a regiones de la imagen. A continuación, se tratan estas asociaciones como datos de entrenamiento para una segunda red, que aprende a generar los fragmentos de las oraciones.

■ Aprendiendo a relacionar datos visuales con datos de lenguaje

El modelo de alineación necesita una entrada de un conjunto de imágenes con sus respectivas frases descriptivas. La idea principal de este modelo es que las personas puede escribir frases referentes a la imagen, pero no sabemos a que parte de la imagen se refieren estas. Asumimos, entonces, que existe una relación entre la frase y el objeto que se describe de la imagen; por tanto intentamos encontrar estas relaciones para posteriormente aprender a generar estos fragmentos de imágenes a los que las oraciones se refieren.

Primero creamos una red neuronal que asocie las palabras con las regiones de la imagen. Entonces, el siguiente objetivo sera relacionar semánticamente estas palabras.

■ Representando imágenes

Observamos que las frases hacen referencias frecuentes a los atributos de los objetos que están describiendo. Así, siguiendo el método de Girshick[4] para la detección de objetos en todas las imágenes se usa una Red Neuronal Convolucional. La CNN es pre-entrenada con el *dataset* de ImageNet[3], y afinada sobre las 200 clases del ImageNet Detection Challenge[9].

■ Representando frases

Para establecer las relaciones inter-modales, sería conveniente representar las palabras de las frases en el mismo espacio dimensional que ocupan en la región de la imagen. El enfoque más simple podría ser proyectar cada palabra en este espacio. Debido a que lo anterior presenta ciertos defectos, se podría usar una extensión de este método, que sería el uso de bigramas de palabras o el uso de relaciones de dependencia. Sin embargo, esto sigue imponiendo un tamaño máximo arbitrario de la imagen y requiere el uso de Árboles de Dependencia, que debe ser entrenada con texto no relacionado.

Para conseguir este propósito en el modelo de este trabajo se usa una Red Neuronal Recurrente Bidireccional (ver sección 3.3.2) para computar las representaciones de las palabras.

- Objetivo de alineamiento:

  Se ha descrito las transformaciones que asocian a cada imagen con la frase en conjunto de vectores en común, dentro de un mismo espacio dimensional. Como la extracción y la predicción se realizan sobre la imagen y la frase entera, se creó una puntuación imagen-frase como una función de las puntuaciones región-palabra individuales. Intuitivamente, una predicción del tipo imagen-frase, tendrá una puntuación elevada si sus puntuaciones región-palabra son elevadas, osea se relacionan bien y tienen buen soporte.

- Decodificando segmentos de texto alineados a imágenes:

  Lo que se pretende es generar secuencias de palabras que estén asociadas a una imagen, no una palabra suelta asociada a la misma; además estas deben estar relacionadas semánticamente entre ellas por lo tanto se meten en una "caja" para posteriormente generar la secuencia con esa relación semántica exigida.

Red Neuronal Recurrente Multimodal para la generación de descripciones

En esta sección se toma en cuenta que la entrada sera un conjunto de imágenes y sus correspondientes descripciones. Estas pueden ser imágenes y su frase descriptiva o regiones y fragmentos de texto, como se ha comentado anteriormente. El desafío clave es diseñar un modelo que pueda sacar como predicción a través de una imagen, que consiste en una secuencia variable (la frase que describa la imagen). En anteriores trabajos basados en RNN, esto se conseguía obteniendo una probabilidad de cuál sería la siguiente palabra en una secuencia, teniendo disponible la palabra actual y el contexto anterior (dado por las conexiones recurrentes). En este trabajo se usa una pequeña extensión de estas redes.

- Entrenando la RNN:

  LA RNN está entrenada para predecir la siguiente palabra usando la palabra actual y el contexto en el tiempo anterior. Se condiciona las predicciones de la RNN a través de la interacción con su *bias* en los primeros pasos de la predicción. El entrenamiento funciona de la siguiente manera: Se empieza con una palabra especial para determinar el comienzo de la predicción, y la primera predicción obtenida será la primera palabra de la frase a obtener. La siguiente palabra se traduce tomando la palabra actual como la recién predicha y el contexto anterior, esperando que el modelo prediga la siguiente palabra como la segunda; y así sucesivamente. Cuando se llega a la última palabra, la palabra predicha será una palabra reservada para determinar el fin de la predicción.

- Testeando la RNN:

  Para predecir una frase, se computa la representación de la imagen y se empieza con la palabra reservada que determina el comienzo de la predicción. Se muestra una palabra de la distribución, que se toma como palabra actual y se procede a predecir la siguiente palabra; esto se repite hasta obtener la palabra reservada que determina el fin de la frase.

■ Conclusiones

Se ha creado un modelo capaz de generar frases a partir de imágenes, pero tiene la gran limitación de que estas sólo pueden tener una resolución fija. Por último, hay que tener en cuenta que este modelo en realidad consiste en dos modelos, uno que preprocesa las imágenes para extraer las palabras asociadas a regiones de la imagen y el segundo es la RNN que predice las frases.

5.3 Artículo del Arctic Caption

5.4 Artículo del Google

6. ASPECTOS RELEVANTES DEL DESARROLLO DEL PROYECTO

En este apartado se introducen los aspectos más relevantes del proyecto.

6.1 Dificultades encontradas

Durante el desarrollo del proyecto nos hemos encontrado varias dificultades que han hecho que el proyecto se retrase considerablemente y su avance no haya sido ni fácil ni rápido.

6.1.1 Dificultades con DeepBeliefSDK

En principio se intentó su uso directamente en Android, pero se encontró que fue excesivamente difícil para el alumno usarlo en Android. Aunque viene un ejemplo en Android, este se intentó hacer funcionar a través de la herramienta Android Studio, pero no funcionaba correctamente el ejemplo al intentarlo.

Se intentó en un principio, siguiendo los ejemplos aportados por el autor, hacer una aplicación lo más sencilla posible para un dispositivo Android. No se obtuvo un resultado satisfactorio, pues no se consiguió que se compilara de manera correcta la aplicación. Entonces se procedió al intento de compilar el ejemplo ofrecido en el proyecto, el cuál resulto que tampoco compilaba y era bastante más complejo que la aplicación de prueba inicial como para poder arreglar los errores.

En conclusión vimos que la ejecución en Android de esta aplicación iba a dar muchos problemas y determinamos que se programaría un servidor para que el cliente subiera ahí la foto y la librería se ejecutará en el lado del servidor. Cabe destacar que después de un par de intentos de instalación del proyecto, los ejemplos, que estaban en la máquina Linux donde se alojaría el servidor, funcionaron de manera adecuada.

6.1.2 Dificultades con GSOAP y Apache

Este fallo es el que más ha retrasado al proyecto y ha supuesto una dificultad enorme a la hora de llevar a cabo el mismo.

Empezamos con que para usar GSOAP[1] se tuvo que estudiar una serie de cosas para poder adquirir los conocimientos necesarios para usar la herramienta, dichos conocimientos serán listados aquí:

[1] http://www.cs.fsu.edu/~engelen/soap.html

- **XML:**[2] Se tuvo que coger un nivel adecuado en el uso de XML ya que la herramienta GSOAP se basa en el uso de este tipo de archivos como medio de comunicación en las distintas peticiones y respuestas que procesa. Además el XML también es necesario para comprender el funcionamiento de SOAP y de WSDL, los cuales son completamente necesarios para entender el funcionamiento de la herramienta GSOAP.
- **SOAP:**[3] Esta especificación se tuvo que estudiar para comprender el funcionamiento de la herramienta GSOAP y en qué se basaba su funcionamiento, entender el por qué debía funcionar la herramienta y como se realiza la comunicación gracias a ella. Aunque el SOAP no es usado directamente cuando usas GSOAP es necesario conocer esta especificación ya que GSOAP sí que usa WSDL, para el cual tenemos que tener un conocimiento básico, al menos, de SOAP para poder usarlo.
- **WSDL:**[4] Esta otra especificación sí que se usa directamente en la herramienta GSOAP y básicamente con ella vertebras toda la aplicación que vas a hacer, de hecho tienes dos opciones:

  La primera es usar un archivo WSDL donde especificas las operaciones que el servidor va a realizar, después con la herramienta GSOAP generas todos los stubs y documentos necesarios para hacer tu aplicación.

  La segunda sería a través de un documento de tipo .h o una cabecera de C. Con el cuál también generas los stubs y documentos necesarios para programar tu servidor, entre dichos documentos se econtrará un archivo WSDL que contendrá la especificación de las operaciones que hay dentro del fichero cabecera que hayas usado. Pero incluso en esta opción necesitas entender SOAP y WSDL porque a través de comentarios tienes que especificar características que irán directamente al fichero WSDL, y que serán necesarios para el correcto funcionamiento del servidor.

Una vez se ha estudiado lo anterior se paso al estudio de la documentación de la herramienta GSOAP, además de el intento de hacer que funcionen sus ejemplos.Cuando se consiguió que funcionarán sus ejemplos se paso a la programación de un servidor propio, una vez se programo y se hicieron las pruebas de que estaba bien programado se procedió a intentar que este funcionará desde un cliente GSOAP. Para que funcionará con el cliente GSOAP se hizo una investigación de cómo hacer que el servidor funcionará en localhost y, siguiendo la recomendación que en la documentación de GSOAP encontramos, se instaló Apache y se inteto usar el módulo de Apache para su funcionamiento con GSOAP.

Como conclusión sacamos que, tras una larga investigación y mucho tiempo dedicada a esta herramienta, esta herramienta no tenia la documentación suficiente como para poder hacerla funcionar con Apache y, a pesar de haberlo intentado muchas veces, no conseguimos que el servidor GSOAP programado por nosotros devolviera alguna vez un resultado coherente al cliente. Finalmente desechamos la opción de trabajar con esta herramienta y le dimos un giro al proyecto con el que esperábamos tener avances más rápidos y mejores, optamos por la programación de un servidor en Flask.

[2] http://www.w3schools.com/xml/
[3] http://www.cs.fsu.edu/~engelen/soap.html
[4] http://www.w3schools.com/webservices/ws_wsdl_documents.asp

6.1.3 Dificultades en la instalación de Herramientas

En este proyecto nos encontramos con que la instalación de las herramientas que se van a usar en el lado del servidor conllevan una carga de trabajo bastante grande. Un ejemplo claro es, que descartamos la herramienta Caffe por la cantidad de dependencias que esta poseía y su complejidad y, finalmente, la herramienta elegida, NeuralTalk, dependía a su vez de Caffe. Por tanto, la instalación de NeuralTalk conllevaba la instalación de Caffe y su cantidad de dependencias.

■ Instalando Caffe

En un primer intento de instalar Caffe, se fueron instalando una a una las dependencias de este según íbamos encontrando los errores en instalación. Esto llevo mucho tiempo ya que cada dependencia tenía a su vez su manera de instalarse y no siempre era de manera directa y limpia, un claro ejemplo fue open-cv, el cuál instalamos de manera manual pero a su vez requería otra instalación y la instalación final resultó no se adecuada.

El problema de la instalación de open-cv llevo a que la instalación completa de Caffe fallara. Lo que nos obligo a limpiar la máquina de toda dependencia de Caffe instalada y empezar desde cero. Afortunadamente nos topamos con un tutorial de cómo instalar PyCaffe en una máquina virtual, el cuál era lo suficientemente útil como para que nos sirva a nosotros; que realmente necesitábamos instalar MaCaffe (el *wrappper* de Caffe para MATLAB)[5].

Gracias a que seguimos esos pasos se resolvió el error de instalación del open-cv, cabe destacar que llegar a este punto lleva un proceso que puede superar fácilmente la hora por la cantidad de dependencias que se deben instalar aquí. El siguiente problema a tener en cuenta es que debíamos instalar MATLAB, por suerte se contaba con una licencia que la Universidad de Burgos había facilitado a los alumnos, de modo que la instalación de esta herramienta no fue realmente muy compleja, aunque puede tardar sobre otra media hora o incluso más.

Una vez se ha pasado el calvario de la instalación de las dependencias llega el proceso de configurar el makefile de Caffe para instalarlo definitivamente en tu máquina. El tutorial anteriormente mencionado nos serviría para algunos aspectos pero la definición de MATLAB la teníamos que hacer nosotros. Se debía definir el directorio en el que se encontraba el ejecutable de MATLAB, o eso ponía en la documentación. La definición del directorio tal y como ponía en la documentación falló, y se tuvo que intentar con distintos directorios hasta que se llego a la solución correcta. Hay que tener en cuenta que cada intento rondaba los 15 minutos porque había que limpiar la anterior instalación del makefile y rehacer todas las comprobaciones del mismo.

Una vez se ha superado todos estos contratiempos, se puede decir que tienes Caffe instalado en tu máquina y que está listo para ser usado. La dificultad de esto es que no hay una guía paso a paso de todo lo que hay que hacer y, en muchos casos, tienes que investigar por tu cuenta cómo instalar cada cosa, lo que conlleva una pérdida de tiempo enorme.

6.1.4 Instalando NeutalTalk

La instalación de NeuralTalk lleva consigo la carga de instalar Caffe por detrás. Pero una vez instalado no puedes realizar ningún tipo de prueba debido a que no tienes entrenada la red. El

[5] https://github.com/BVLC/caffe/wiki/Ubuntu-14.04-VirtualBox-VM

entrenamiento de la red es inmediato porque la documentación es clara en cuanto a esto, pero el entrenamiento de la red puede tardar bastantes horas.

Durante el proyecto se intento entrenar la red por nuestra parte, pero las horas que esto conllevaban podías llegar a sumar días. De modo que se tuvo que buscar alguna red pre-entrenada. Por suerte la solución también fue encontrada en la documentación de la herramienta y se descargo una red pre-entrenada[6] para proceder a hacer pruebas con la herramienta.

6.1.5 Dificultades con NeuralTalk

El trabajo que se ahorro en la instalación de NeuralTalk lo tuvimos que invertir en hacer que este funcione de manera que nosotros queremos. Puesto que NeuralTalk inicialmente sólo viene predispuesto para ser probado con las imágenes que ya tiene y devuelve un fichero HTML como resultado. No había la opción de trabajar sobre imágenes propias.

Mientras que la documentación de instalación de NeuralTalk era sencilla, la documentación para la configuración de este era muy precaria y sólo se podía encontrar a modo de comentario en los ficheros del proyecto. Además de que no encontrabas todo documentado en un sólo fichero, sino que un fichero documentaba una parte y hacía referencia a documentación que se encontraba en un comentario de otro fichero. Esto hizo que el proceso de configuración de NeuralTalk fuera lento y pesado.

En primer lugar se leyó la documentación del fichero que hacía las predicciones. En el ponía que se usaba scripts de MATLAB para la preparación de las imágenes, este script cogía el nombre de las imágenes de un fichero de texto llamado tasks.txt. El primer problema es que el script dependía de la cómo estaba estructurado los ficheros de NeuralTalk, osea que teníamos que adaptarlo a nuestro caso de uso; para ello se tuvo que leer el código y diferenciar dónde debíamos y qué debíamos cambiar para que se adaptara a nuestro caso de eso. Este proceso fue lento y pesado, porque los comentarios no eran lo suficientemente concisos como para poder interpretarlo de manera inmediata, sino que se tuvo que entender el código casi en su totalidad para poder entender qué se debía modificar y por qué. Además de que tuvimos que añadirle líneas para que encontrara a Caffe dentro de nuestro sistema, sino, no funcionaba. En este script nos encontramos además que la definición del objeto de Caffe está mal hecha o no funcionaba en nuestra máquina, de modo que se tuvo que recurrir a la documentación de Caffe. Otra vez se tuvo que indagar en los comentarios dentro del código, porque esto no estaba explicado dentro de la documentación que se encontró, y Caffe es un proyecto más extenso y complejo por lo que llevo mucho tiempo llegar a la solución.

Resuelto el problema anteriormente descrito se procede a la programación del servidor, este tiene que:

- Escribir en el fichero tasks.txt el nombre de la imagen a procesar: Esto fue sencillo, ya que trabajar con ficheros en Python es bastante fácil.
- Ejecutar el script de MATLAB: El script de matlab no predecía, sino que extrae características de la imagen para poder usarla con NeuralTalk. El problema que surgió aquí es que MATLAB nos daba un error de licencia ya que el servidor se ejecutada como superusuario, pero MATLAB detectaba la licencia a nombre del usuario normal. Además se tuvo que investigar la secuencia de comandos correcta para que el script se ejecutara, una vez más la documentación no decía exactamente cómo se debía ejecutar este.

[6] http://cs.stanford.edu/people/karpathy/neuraltalk/

Finalmente cuando se descubrió cómo ejecutar el script, se tuvo que buscar una solución al problema con la licencia. Se intento cambiar la licencia o reinstalar MATLAB, pero no dio resultado. Finalmente se opto a hacer un script bash y ejecutarlo desde el servidor usando el nombre de usuario que sí aceptaba MATLAB en su licencia.

- Seguidamente se tenía que coger el fichero HTML que devolvía NeuralTalk y procesarlo para convertir todos esos datos en la cadena que queríamos, esto no llevo demasiado tiempo porque con la función *split* de cadenas de texto y la facilidad de Python para trabajar con ficheros, se convirtió en una tarea bastante sencilla.

Finalmente se consiguió que integrar así el servidor con NeuralTalk, pero la carga de trabajo que esto llevo fue bastante grande.

6.1.6 Dificultades con Android

Con Android encontramos los problemas más destacables a la hora de realizar la conexión con el servidor y enviarle los datos. Estos son fácilmente numerables por lo que vamos a tratarlos por puntos:

■ Usando la librería de Apache para HTTP

El primer problema de todos es que para usar esta librería era necesario instalarla de manera externa y en la documentación de Apache o Android no se encontraba la solución de manera clara. Se tuvo que investigar durante bastante tiempo hasta dar la solución en un foro de Internet[7].

Después se procedió a programar la conexión y se hizo con la documentación que se encontró de manera más usual, pero esta resultó estar obsoleta y, por lo tanto, no funcionaba al compilar y ejecutar la aplicación. Se tuvo que buscar información de manera más exhaustiva para dar con la solución correcta y que no se encuentra obsoleta.

Una vez se ha cambiado el código nos encontramos con el problema de que nuestra imagen daba incompatibilidades con el tipo de dato que se tenía que definir en la petición de tipo POST, de manera que se recurrió otra vez a una búsqueda bastante exhaustiva hasta dar con la solución de cómo se tenía que tratar la imagen para mandarla dentro de una petición POST a un servidor.

Finalmente descubrimos que, debido a estándares establecidos por Android, lo que habíamos programado no se podía ejecutar porque esto no debía hacerse en el hilo principal de ejecución, sino que debía ser lanzado en un hilo diferente. De manera que se tuvo que reestructurar todo el código para adaptarlo a esta especificación, una vez más se tuvo que realizar una investigación de cómo se realizaba esto y por qué.

■ Localizando la imagen

Después de haber programado la conexión de manera correcta, la petición hacia el servidor se realizaba de manera incorrecta devolviendo un código de error. Descubrimos el origen de este error ayudándonos de los mensajes de error que devolvían las Excepciones que se producían.

[7] http://stackoverflow.com/questions/28538078/java-lang-nosuchfielderror-org-apache-http-message-basicheadervalueformatter-in

Descubrimos que el error se encontraba al pasar los datos de la imagen desde un hilo a otro, de manera que tuvimos que investigar la manera de conseguir solucionar esto. Resulta que este tipo de problemas es bastante complejo de solucionar porque al intercambiar los datos de un hilo a otro se pierde la dirección en la que se encuentra la imagen. Para solucionar esto se ha puesto la imagen en memoria compartida y se crea la imagen con la dirección de la misma, para extraer esta dirección se tiene que crear una función que extraiga la dirección de la imagen. Al trabajar con la dirección y no con la cabecera de la imagen, no se pierde la imagen durante el proceso.

6.1.7 Dificultades con Librería de Traducción en el servidor

El problema que se encontró es que el API de traducción de Google resulto ser de pago, por lo tanto no era del todo adecuado para su uso en este proyecto. Pero se encontró una solución alternativa.

La solución alternativa consistía en simular una conexión a través de navegador e la página de traducción de Google y recibir como respuesta la cadena ya traducida. Esta solución es bastante interesante ya que usa conceptos de los servicios web y más concretamente de los servicios web con una especificación de tipo REST.

Se mandará una petición a la página de Google a través de un url, como sabemos la dificultad está en cómo decir a la página qué queremos traducir y de qué a qué idioma lo queremos traducir. Pues bien, esto se hace definiendo un url. En este url se tiene que hacer uso de la construcción de url, porque en el url es donde van definido los parámetros que se pasan a la página web. Una vez se construye el url, se procede a lanzar la petición con lo parámetros establecidos.

Finalmente, se recibe una respuesta que es recibida en un dato de tipo json, este tipo de dato es fácilmente convertido a un dato de cadena de caracteres en Python y, finalmente, se extrae de ese conjunto de caracteres la cadena que contiene la traducción del texto.

7. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

En este apartado se presentarán las conclusiones y las posibles líneas de futuro que saquemos del proyecto.

7.1 Conclusiones del Proyecto

7.2 Lineas de trabajo Futura

I. PLAN DEL PROYECTO SOFTWARE

Este apartado está dedicado al plan de proyecto software donde se comenta todo el proceso de planificación del proyecto.

I.1 Introducción

La planificación del proyecto se lleva a cabo de forma ágil, con la metodología SCRUM. Y la plataforma sobre la que se ha trabajado para realizar la planificación es VersionOne. Y la planificación se ha dividido en *sprints* o iteraciones, las cuáles tienen la mayoría una duración de una semana, aunque por problemas algunas han durado más.

I.1.1 Problemas encontrados

Debido a que las primeras dos iteraciones se realizan de forma simulada, osea después del tiempo previsto, por eso no se pueden sacar sus gráficos *burn-down*. Además de que al día 30 de uso del VersionOne se caducó la prueba y perdí todos los datos del proyecto, pero contactando con el soporte de VersionOne estos me reabrieron la prueba por una semana para extraer los datos necesarios, pero debido a la tardanza en su respuesta hay un *sprint* que dura 3 semanas y no podemos extraer su gráfico *burn-down* tampoco.

I.2 Planificación temporal del proyecto

En este apartado se presentan los distintos *sprints* y las tareas de cada uno.

I.2.1 Sprint 1:23 de Diciembre al 12 de Febrero

Este *sprint* es de una duración bastante mayor al resto porque se empezó con mucha antelación y su objetivo era el de documentarse. Se estudió los distintos aspectos del proyecto y se tomo contacto con el mismo.

Este *sprint* es importante porque es donde se asientan las bases del proyecto y se empieza a tomar contacto con el proyecto. Sin este *sprint* se hubiera empezado sin ninguna base y el comienzo del proyecto hubiera sido más tardío y , por tanto, su avance más lento.

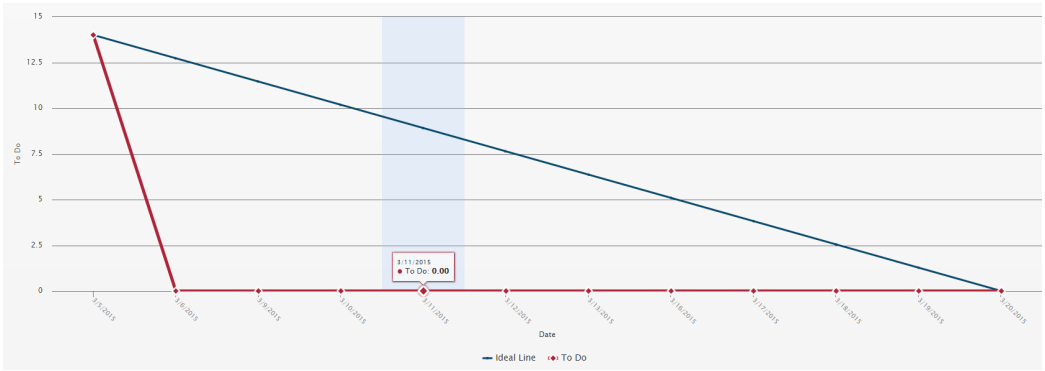

Figura I.1.: Burn-down del sprint 3

I.2.2 Sprint 2:12 de Febrero al 26 de Febrero

Esta iteración se dedicó exclusivamente a la decisión de las herramientas que se van a usar en el proyecto y a la familiarización con las mismas.

Aquí se decidió:

- Gestor de versiones: GitHub
- Gestor de Tareas: VersionOne
- Entorno de desarrollo: Android Studio
- Herramienta de ofimática: LaTeXcon su editor TeXMaker

I.2.3 Sprint 3:26 de Febrero al 6 de Marzo

En este *sprint* se instalaron y se crearon cuentas de las herramientas del anterior *sprint*. Se dividió en distintas tareas:

- Crear cuenta VersionOne
- Crear cuenta en GitHub
- Instalar Android Studio
- Instalar librería DeepBeliefSDK

Además se procedió a invitar a los tutores a VersionOne, realizar el primer commit en GitHub, probar ejemplos de Android y del DeepBeliefSDK y, finalmente, se generó la documentación asociada a este *sprint*.

Podemos ver el gráfico *burn-down* de este sprint en la imagen I.1.

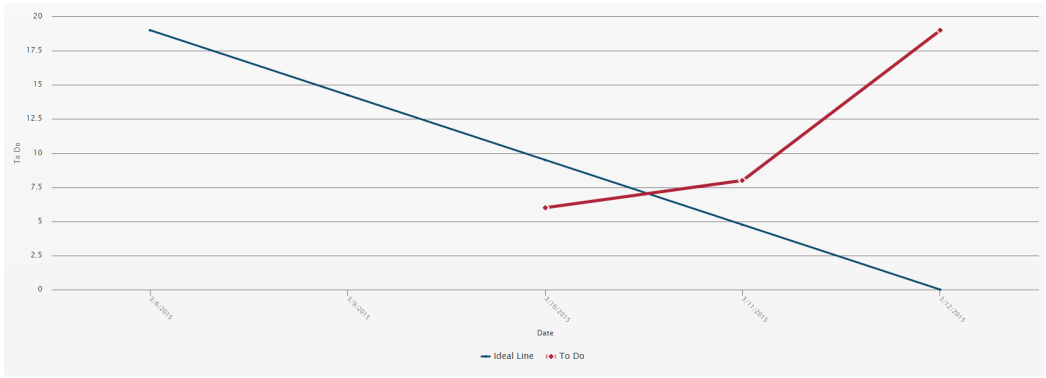

Figura I.2.: Burn-down del sprint 4

I.2.4 Sprint 4:6 de Marzo al 13 de Marzo

En esta iteración se profundiza, sobretodo, en la utilización de la librería DeepBeliefSDK, se probarán ejemplos y trabajarán sobre ellos. Además se generará la documentación asociada al *sprint*.

Se identifican las tareas:

- Consulta de la documentación de Android: se pasa a avanzar en la programación de Android y se empiezan a realizar las primeras aplicaciones de prueba con ayuda de la documentación de Android.
- Instalación Linux: Se procede a instalar un Linux con su distribución Ubuntu en una máquina virtual sobre la que trabajar.
- Impresiones de Pantalla: Se hacen capturas de pantalla del proceso de instalación de las herramientas para la futura documentación en la sección: Manual del programador (II)

Este fue el primer *sprint* en el que añadimos el *Retrospective Meeting* y el *Sprint Planning* al VersionOne, pese a que sí que habíamos hecho estas reuniones.

En el *Retrospective Meeting* se determino que:

- Se estudió el funcionamiento de la librería DeepBeliefSDK
- Se hicieron las impresiones de pantalla para la documentación
- Se hicieron más pruebas con Android.
- Se envió invitaciones a los profesores a VersionOne y Github

El *Sprint Planning* simplemente sirvió para determinar las tareas a realizar en esta iteración, las cuáles ya han sido comentadas antes.

Además esta iteración viene acompañada con un gráfico *burn-down*s el cuál puede verse en la imagen I.2

I.2.5 Sprint 5:13 de Marzo al 27 de Marzo

Este *sprint* está dedicado en su mayoría al aprendizaje de GSOAP, que es un *framework* para poder programar servidores en código C y C++. El estudió de esta herramienta llevó bastante tiempo debido a que su documentación, no solo estaba puramente en inglés, sino que además no era lo suficientemente precisa como para alguien que estuviera empezando a programar servicios WEB.

En el *Sprint Planning* se propuso como objetivos a tener en el siguiente *sprint*:

- Evaluar si usar Axiss2/c o GSOAP para el servidor
- Continuar con la documentación
- Ejemplos de Android

En este *sprint* podemos identificar las siguientes tareas que finalmente se definieron en la herramienta VersionOne:

- Pruebas en Linux: Se hacen pruebas de la librería GSOAP en la máquina virtual de Linux, en su distribución Ubuntu.
- Pruebas en Android: Se hacen pruebas de distintos códigos y aplicaciones en Android que posteriormente nos puedan servir
- Primera versión del Cliente: Se genera la primera versión del cliente Android para el proyecto. Esta sólo es una interfaz sencilla que toma una foto con el móvil y la guarda.
- Estudio de WSDL y SOAP: Se estudia estas dos especificaciones de XML, junto con un pequeño repaso de XML. Estas dos especificaciones son totalmente necesarias para el desarrollo de la aplicación, aunque al final puede que no se lleguen a usar directamente, el conocimiento de las mismas es requisito indispensable para conocer cómo trabaja el servidor internamente.
- Descargar materiales: Para el estudio y trabajo con la librería GSOAP es necesario una serie de archivos, esta tarea es en la que nos descargamos todos los necesarios.
- Primera versión del fichero WSDL: Un versión inicial de un fichero WSDL con el que generar archivos *stubs* con la herramienta GSOAP. Este fichero contiene las especificaciones de los servicios que dará el servidor al cliente y qué tipo de datos admite y devuelve.
- Generar documentación:Como en todas las iteraciones se procura generar la documentación asociada.

En el *Retrospective Meeting* se determina qué tareas del *sprint* han sido terminadas, cuáles no y el avance de la iteración. En este caso se identificó lo siguiente:

- Se decidió usar GSOAP porque era más sencillo.
- Se estudió GSOAP y se consiguió implementar un cliente en Linux de un ejemplo básico de calculadora.

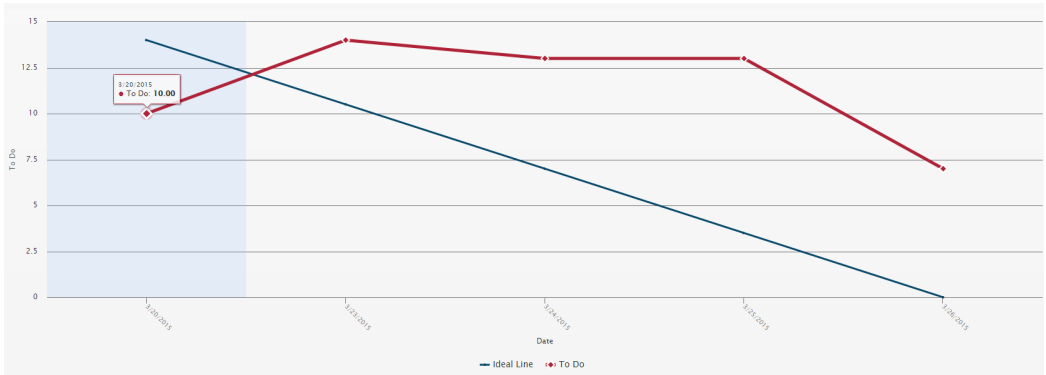

Figura I.3.: Burn-down del sprint 5

- Se continuó con ejemplos de Android creando el proyecto del cliente en su primera versión, aunque este no hacía nada más que mostrar una imagen por pantalla.

Esta iteración también tiene un gráfico de tipo *burn-down* asociado y se puede ver en la imagen I.3

I.2.6 Sprint 6: 28 de Marzo al 22 de Abril

En esta iteración se procedió a crear prototipos, tanto de cliente como de servidor, del proyecto para poder empezar con el desarrollo software del mismo.

En la planificación dentro del *Sprint Planning* se determinó:

- Tratar de implementar el ejemplo básico de servidor, con el que podamos hacer las primeras pruebas.
- Tratar de hacer el cliente en Android, que se conectara al ejemplo básico de servido. Con esto empezaremos a establecer cómo se realizará la conexión entre cliente y servidor.
- Tratar de hacer el prototipo con DeepBeliefSDK para empezar a intentar conectarlo con el servidor.
- Continuar con la documentación.

Esos fueron los objetivos fijados para este *sprint*, que es inusualmente largo debido a la serie de problemas que se encontraron en el mismo y que son explicados en el *Retrospective Meeting*.

Mientras que en VersionOne lo que tenemos definido es:

- Instalar Apache y demás herramientas: Nos encontramos con que el servidor de GSOAP necesitaba de Apache para poder ejecutarlo en localhost y así establecer la conexión con el cliente Android, así que procedimos a la instalación de las herramientas necesarias.
- Construcción del servidor: Se montó el servidor con un ejemplo de calculadora básico y se comprobó con las propias herramientas de GSOAP, que este estaba bien definido.

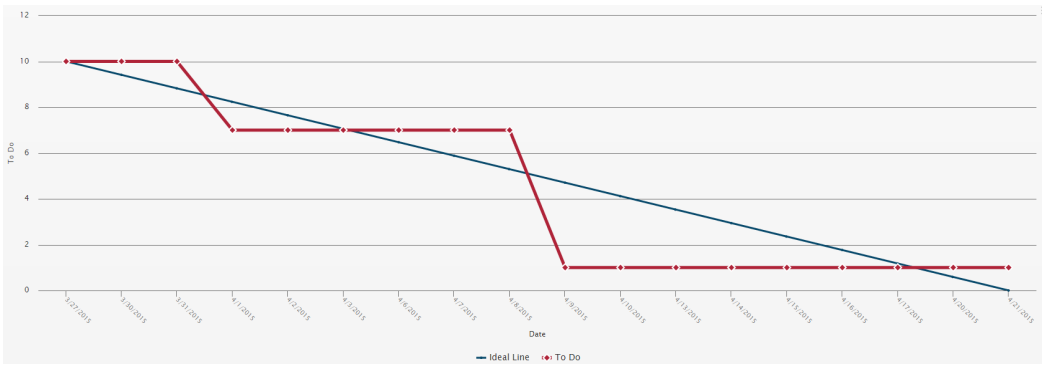

Figura I.4.: Burn-down del sprint 6

- Construcción del cliente: Se montó un cliente GSOAP con el que también se probo el ejemplo básico de servidor, este y el servidor funcionaban de manera adecuada.
- Pruebas con distintos servicios en Linux: Se remontaron varias veces el cliente y el servidor, pero con distintas especificaciones SOAP, para probar varios tipos de datos en el envío.
- Primera versión del Servidor: Después de construir el ejemplo básico se paso a conectar el DeepBeliefSDK con el servidor y este con Apache. Se empezó por probar la conexión con Apache, la cuál resultó un rotundo fracaso.
- Escritura de la documentación: Se genera, como siempre, la documentación asociada a la iteración.

Finalmente tenemos nuestro *Retrospective Meeting*, en el que determinamos lo siguiente:

- No se consiguió conectar GSOAP con Apache
- La documentación para el proceso de conectar Apache y GSOAP es escasa y la poca que hay no nos enseña un procedimiento correcto para hacer funcionar esto.
- Se decide abandonar la programación en C y la librería DeepBeliefSDK
- Conclusión: No se consiguieron los objetivos y se decide tomar un nuevo rumbo con el proyecto.

Podemos observar el gráfico de la imagen I.2, en el que observamos el gráfico *burn-down* del *sprint*.

I.2.7 Sprint 7: 22 de Abril al 27 de Abril

Después del anterior *sprint* se procedió a cambiar totalmente el rumbo del proyecto, para ello se ha tomado la decisión de continuar el proyecto en Python, con una librería nueva y con un *framework* nuevo.

En el *Sprint Planning* se determinó los siguientes objetivos:

- Crear prototipo inicial de servidor con Python y Flask

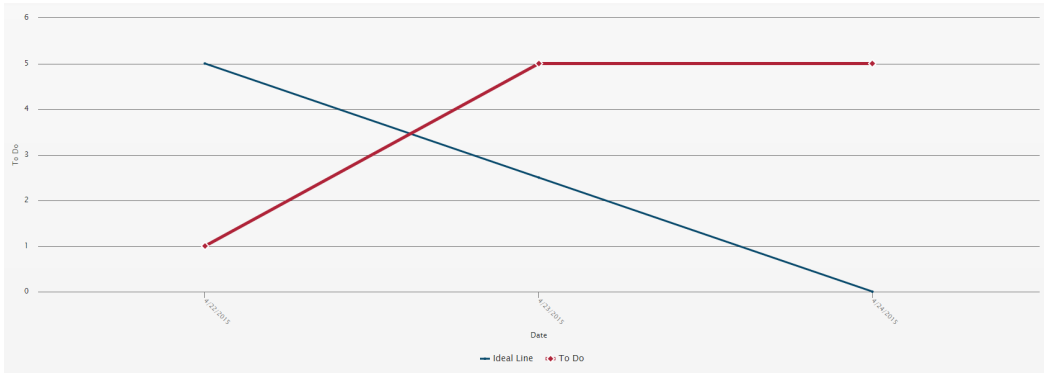

Figura I.5.: Burn-down del sprint 7

- Estudio del funcionamiento de la librería NeuralTalk
- Prototipo del cliente en Android

En VersionOne se definieron una serie de tareas para llevar a cabo estos objetivos, las tareas definidas dentro de la herramienta de gestión de tareas son:

- Instalación de NeuralTalk y sus dependencias: Se procede a la instalación del proyecto de GitHub Neuraltalk, además de empezar a hacer pruebas con él y leer la documentación.
- Instalación de Flask y Python: Para la programación del servidor es necesario la instalación de Flask y de Python.
- Primera versión del servidor: Se procede a la creación de una primera versión del servidor que reciba una imagen y la guarde en el sistema, posteriormente esta será procesada por la herramienta de predicción que se haya elegido.

Una vez se han definido las tareas estas se realizaron a lo largo de la semana, aunque como veremos en ele gráfico *burn-down* producido que al final, debido a problemas a la hora de trabajar con NeuralTalk, se ve aumentado el trabajo por hacer ya que surgen tareas no planificadas, las cuáles pasarán a ser resueltas en el siguiente *sprint*.

Finalmente, en el *Retrospective Meeting* se determinó lo siguiente:

- Se instalaron NeuralTalk y sus dependencias, lo cuál resultó bastante costoso ya que al final este proyecto era dependiente de otro que estudiamos al principio, Caffe.
- Se instaló correctamente Python, Flask y todas sus dependencias.
- Se programo la primera versión del servidor tal y como se había planificado.
- En el trabajo con la herramienta NeuralTalk se vio que esta era más compleja de lo esperado, pues no se podía simplemente predecir la imagen que uno quisiera sino que tendría que modificarse lo que en el proyecto original había para el caso de uso en el que nos encontramos.

Finalmente vamos a ver un gráfico de tipo *burn-down* asociado a esta iteración y generado por la herramienta VersionOne, el gráfico se puede observar en la imagen I.5.

I.2.8 Sprint 8: 27 de Abril al 8 de Mayo

En este *sprint* se procedió a la creación de prototipos de cliente y servidor como principal objetivo. Por lo que en el *Sprint Planning* se determinaron los siguientes objetivos:

- Hacer el prototipo que permita subir la imagen al servidor, lanzar el reconocimiento y devolver la cadena de descripción de la imagen.
- Probar el API de traducción de Google para que la cadena devuelta, en vez de inglés, sea en castellano.
- Probar el Text2Speech para que la aplicación móvil, en vez de sólo mostrar la cadena devuelta, la lea.
- Buscar información sobre bibliotecas de prueba, tanto para el servidor como para el cliente, para empezar a elaborar el conjunto de pruebas de la aplicación.
- Actualizar la documentación con todas las nuevas cosas.
- Comprobar si se puede entrenar la red neuronal con imágenes propias.

Para la realización de estos objetivos se tuvo que determinar una serie de distintas tareas con las que ir trabajando. Esto se hace en VersionOne, en él se definieron las siguientes tareas:

- Instalando Caffe y sus dependencias: Nos encontramos ante la situación de que Neuraltalk usa el *wrapper* para Matlab de Caffe, por lo que debemos instalar Caffe y comprobar que este funciona.
- Entrenamiento de la red: Se comprobó que la red podía ser entrenada con distintos *datasets*, incluido uno propio. Como el entrenamiento completo de la red neuronal podía llegar a tardar varios días en un ordenador normal, se procedió a descargar una red ya entrenada.
- Pruebas con imágenes de ejemplo: Una vez descargada la red entrenada se procedió a probar esta con las imágenes que venían de ejemplo en la librería, comprobando que estas funcionaban pero que la ejecución de la misma sobre imágenes propias conllevaría algunas dificultades.
- Combinación de Neuraltalk con el servidor: Se combina Neuraltalk con el prototipo de servidor ya creado anteriormente. Aquí tenemos que resolver todos los problemas para lanzar la predicción sobre nuestras imágenes propias. Esto está detalladamente explicado en la sección de Aspectos Relevantes (6), más concretamente en la sección de Dificultades con NeuralTalk (6.1.5).
- Añadiendo librería de traducción: Se intentó añadir el API de Google para la traducción de cadenas de texto, pero esta resulto ser de pago y tuvimos que optar por otra solución, la cuál es detallada en la sección de Aspectos Relevantes del Proyecto, más concretamente en el apartado de Dificultades con la librería de Traducción en el servidor (6.1.7).
- Conectando con el servidor : Se modifica el prototipo de cliente Android que se tenía para conectarlo con el nuevo servidor y mandarle una imagen, para después recibir la predicción.
- Estudiando las estructuras de conexión: Como surgió un problema al conectar la aplicación con el servidor, ya que nos devolvía como respuesta que hemos hecho una petición errónea, se procedió a estudiar la forma en qué se conectaba la aplicación con el servidor, aunque aquí no resultó estar el problema de la petición.

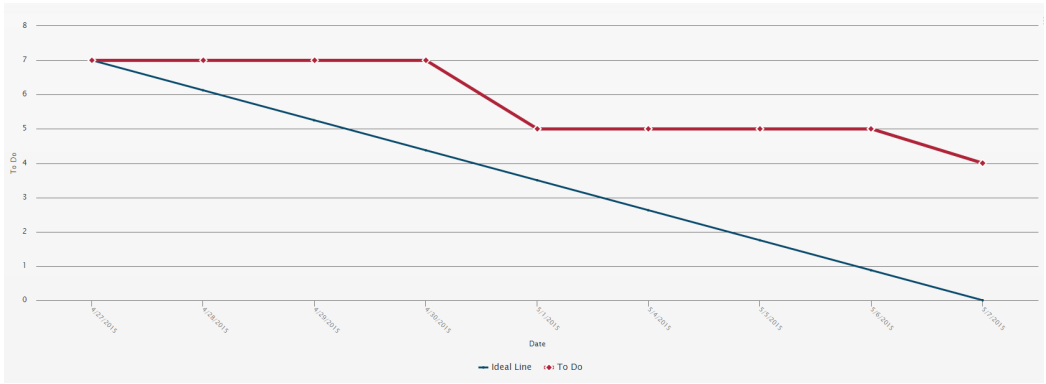

Figura I.6.: Burn-down del sprint 8

- Añadiendo librería de habla: Se usa la librería Text2Speech de Android para hacer que la aplicación de instrucciones guiadas por voz y que la predicción se lea en voz alta.

Como se puede observar hay una cantidad de tareas bastante elevadas y estas conllevan bastante trabajo, por lo que gran carga del desarrollo software se encuentra en este *sprint*. Como Resultado del *sprint* obtenemos lo siguiente en el *Retrospective Meeting*:

- Hacer el prototipo que permita subir la imagen al servidor, lanzar el reconocimiento y devolver la cadena de descripción de la imagen: **Hecho**. A pesar de la cantidad de problemas con esta tarea se logró hacer en esta iteración, pero la carga de trabajo que requirió fue bastante elevado.
- Probar el API de traducción de Google para que la cadena devuelta, en vez de inglés, sea en castellano: **Hecho**. Se tuvo problemas con el API de Google pero se encontró una solución para resolverlo y se añadió correctamente al servidor.
- Probar el Text2Speech para que la aplicación móvil, en vez de sólo mostrar la cadena devuelta, la lea: **Hecho**. Se añadió correctamente esta funcionalidad al prototipo de aplicación.
- Buscar información sobre bibliotecas de prueba, tanto para el servidor como para el cliente, para empezar a elaborar el conjunto de pruebas de la aplicación:**No hecho**. No se buscó información porque no hubo tiempo.
- Actualizar la documentación con todas las nuevas cosas: **No hecho**. No se tuvo tiempo de trabajar sobre la documentación en esta iteración.
- Si te quedara tiempo, intentar ver si se puede entrenar la red neuronal con imágenes propias **No hecho**. Se considera no hecho porque no se llegó a probar esta utilidad a pesar de que pareciera que sí es posible.

De este *sprint* también se tiene un gráfico *burn-down* asociado, el cuál puede verse en la imagen I.6.

I.2.9 Sprint 9: 9 de Mayo al 15 de Mayo

En el anterior *sprint* se tuvo un gran avance en el desarrollo de la aplicación de muestra, pero no se avanzó en el desarrollo de la documentación del proyecto y, por tanto, se procedió a centrarse

en este aspecto en esta iteración. En la reunión de *Sprint Planning* se determinaron los siguientes objetivos:

- Hacer documentación del proyecto: Se ha escrito la documentación pendiente de lo que se ha hecho y comenzado nuevos apartados que aún no se habían escrito.
- Instalar Caffe con octave: Se intentó instalar Caffe con octave porque daba problemas el Caffe, estos están explicados detalladamente en la sección: 6.1.5.
- Probar servidor con GPU: Se pretendía instalar el software desarrollado en una máquina anfitriona para poder ejecutar la predicción en GPU y reducir así su tiempo de ejecución.
- Buscar herramientas de pruebas: Se buscarán herramientas con la que hacer teses sobre la aplicación software para asegurar sus buen funcionamiento.

Las tareas VersionOne y las tareas del *Sprint Planning* coinciden en este caso, así que no se volverán a escribir.

Y, en el *Retrospective Meeting*, se definió que:

- Hacer documentación del Proyecto: **Hecho**. Aunque no se terminó la documentación entera del proyecto, sí que se hizo las partes que se tenían planificadas por hacer.
- Instalar Caffe con Octave: **Fallo**. Se comprobó que no se podía instalar Caffe con Octave porque la instalación misma de MatCaffe, que es el *wrapper* para Matlab de Caffe, necesita del compilador mex de Matlab.
- Buscar Herramientas de prueba: **No hecho**. Debido a que se dedicó mucho tiempo a la documentación y a que la prueba de VersionOne llegó a su fin, esta tarea quedó aplazada para futuros *sprints*.
- Probar servidor con GPU: **No hecho**. No se terminó la instalación del proyecto en una máquina anfitriona porque hubo problemas hardware, se perdieron todos los datos del disco duro y no se pudo trabajar sobre esto en este *sprint*.

Esta iteración no lleva asociado ningún gráfico ya que, debido a que se terminó la prueba en VersionOne, no se pudo actualizar día a día el progreso del *sprint* y, por tanto, no se genera ningún gráfico útil.

I.2.10 Sprint 10: 15 de Mayo al 5 de Junio

En este *sprint* se tuvieron varios problemas, entre ellos que se seguía sin saber por qué no funcionaba VersionOne y se descubrió a mitad de la iteración. Por lo que este *sprint* no dispone de ningún gráfico representativo, ni tampoco de una planificación muy correcta en VersionOne.

En el *Sprint Planning* se determinó que los objetivos de la iteración son:

- Acabar la documentación
- Seguir intentando configurar el proyecto en una máquina anfitriona y hacer pruebas con GPU, si es posible.
- Tratar de recuperar los datos perdidos del VersionOne, poniéndose en contacto con el equipo de soporte de VersionOne.

En VersionOne no tenemos una buena planificación de tareas porque los datos y la instancia de este programa se recuperó a mediados de la iteración, por lo que no fue posible recoger los datos ni el avance de la iteración de manera correcta.

En el *Retrospective Meeting* se determinó lo siguiente:

- Se avanzó bastante en la documentación, sobretodo en la parte de Plan de Proyecto Software (I).
- Se instaló lo máximo posible de la herramienta en anfitrión, pero se detectó la necesidad de modificar el cliente Android y el servidor Flask para facilitar su instalación y portabilidad.
- Se descubrió que VersionOne dejó de funcionar por haber terminado el período de prueba de este, así que se pidió que lo extendieran un poco; consiguiendo dos semanas más. Por lo que se tiene que sacar todos los datos que se pueda de VersionOne y, si es posible, migrar el proyecto a una nueva versión de prueba de este.

II. MANUAL DEL PROGRAMADOR

En esta sección se procederá a la explicación detallada de cómo instalar las herramientas necesarias y qué herramientas son necesarias para trabajar sobre este proyecto.

II.1 Instalación del JDK

La primera, y más esencial de las herramientas, es el JDK de java, que es el set o conjunto de herramientas y librerías para los desarolladores de java.

Primero deberemos ir a la página de Oracle en la que descargaremos el jdk, la página debería tener un aspecto más o menos como el de la imagen II.1 En dicha página tendremos que aceptar la licencia y posteriormente descargar el JDK que sirva para nuestra máquina. Una vez hemos descargado dicho archivo, lo ejecutamos. Una vez ejecutado seguimos los siguientes pasos para su instalación.
En la imagen II.2 vemos el primer paso para la instalación del *JDK*, el cual consta de dos pantallas. La primera será solamente una pantalla de bienvenida, por lo que debemos pulsar a *next* o siguiente. La segunda pantalla nos muestra los elementos que van a ser instalados, esto no lo tocamos; además nos muestra también el directorio en el que queremos instalar el *JDK*, esto lo podemos dejar como está o podemos, si queremos, configurarlo en un directorio personal que queramos. Lo único es que habrá que tener cuidado con el *PATH* del equipo para que luego *Android Studio* pueda encontrar la distribución de *JDK* que tengamos en nuestra máquina.

En la imagen II.3 nos encontramos el paso 2 para la instalación de nuestro *JDK*, este también consta de dos pantallas. La primera es el estado de la instalación, veremos una barra que se irá llenando en función de que la instalación vaya avanzando. Finalmente se nos mostrará la pantalla que nos indicará que la instalación se ha realizado correctamente y nos da la opción de pulsar el botón *Next Steps* uqe nos llevará a la *API de Java* o simplemente finalizar la instalación.

II.2 Instalación de Anrdoid Studio

Para la programación del cliente se usará *Android Studio*, una herramienta ideada para programar en *Android* exclusivamente. Lo primero que debemos hacer es dirigirnos a la página web donde descargaremos Android Studio, la cual tendrá un aspecto similar al de la II.4. En ella haremos *click* sobre el botón de descarga (*Download Android Studio*) y procederemos a la instalación ejecutando el fichero que hemos descargado, todo este proceso es para un sistema operativo *Windows*.

En el primer paso de la instalación, que se puede observar en la imagen II.5, nos encontraremos con dos pantallas. La primeara es simplemente una pantalla de bienvenida a la instalación de

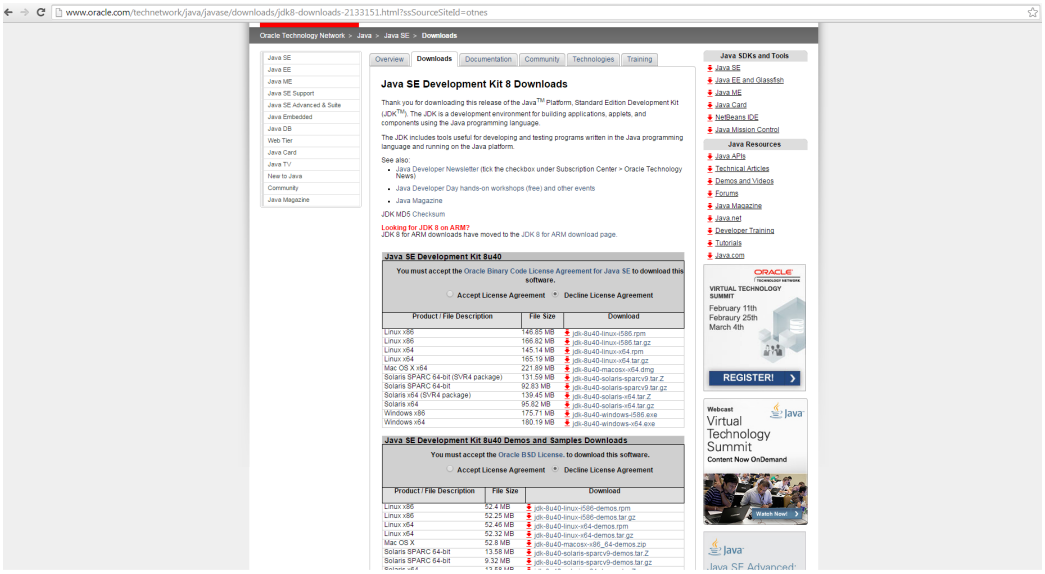

Figura II.1.: Página de descarga del JDK de Java

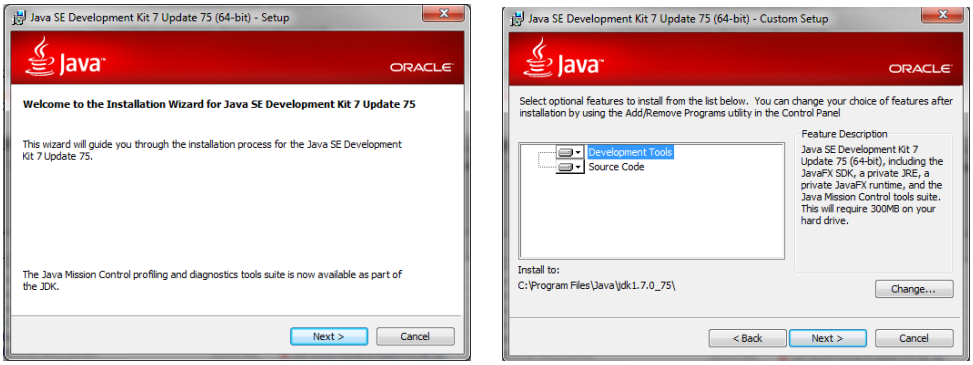

Figura II.2.: Instalación del JDK, paso 1.

Figura II.3.: Instalación del JDK, paso 2.

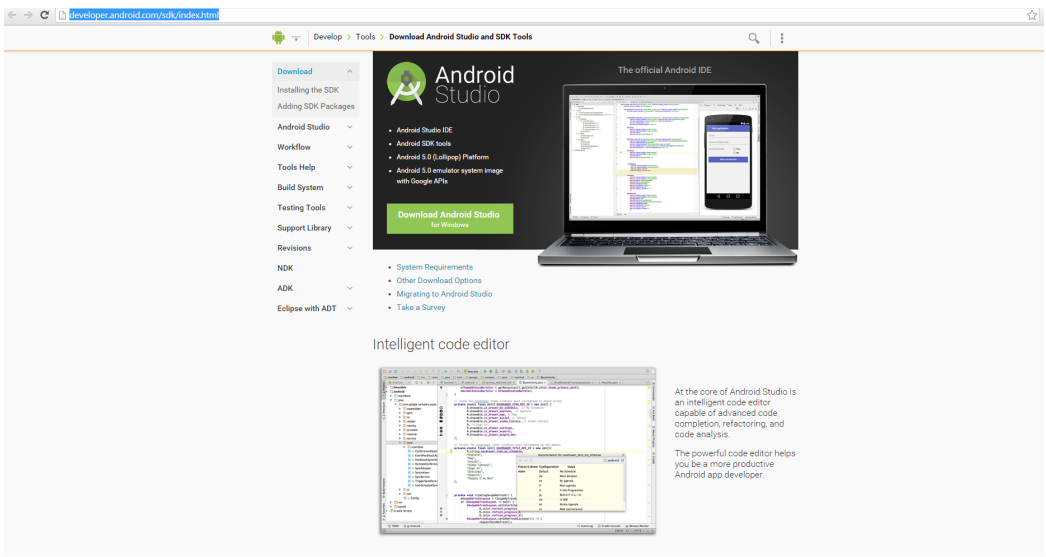

Figura II.4.: Página de descarga de Android Studio

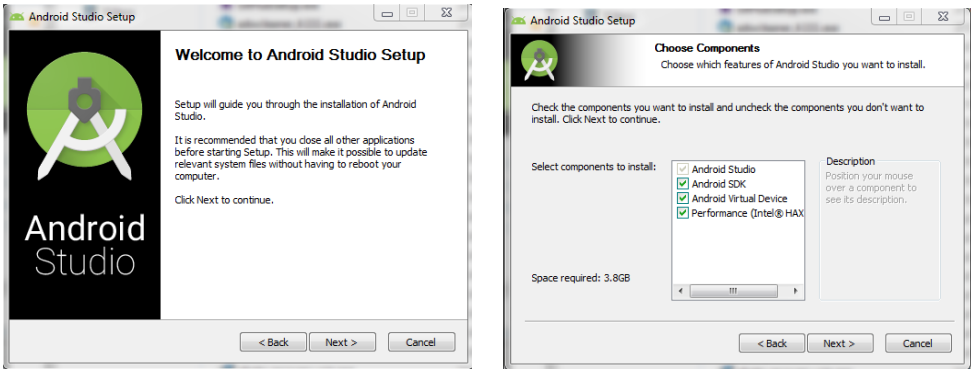

Figura II.5.: Instalación de Android Studio, paso 1.

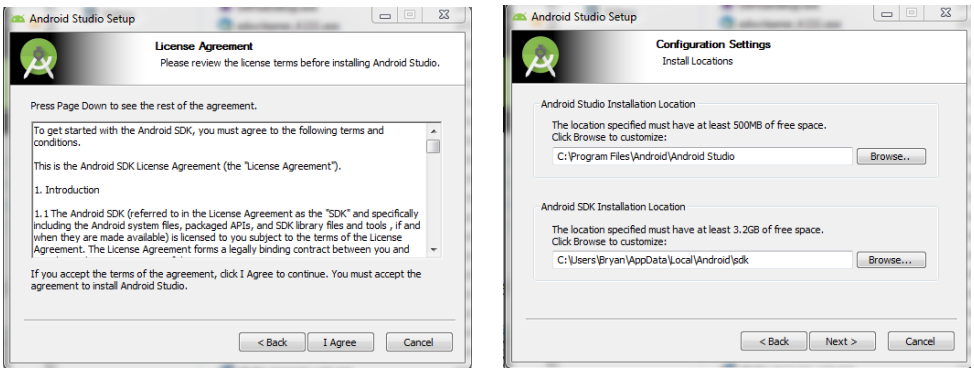

Figura II.6.: Instalación de Android Studio, paso 2.

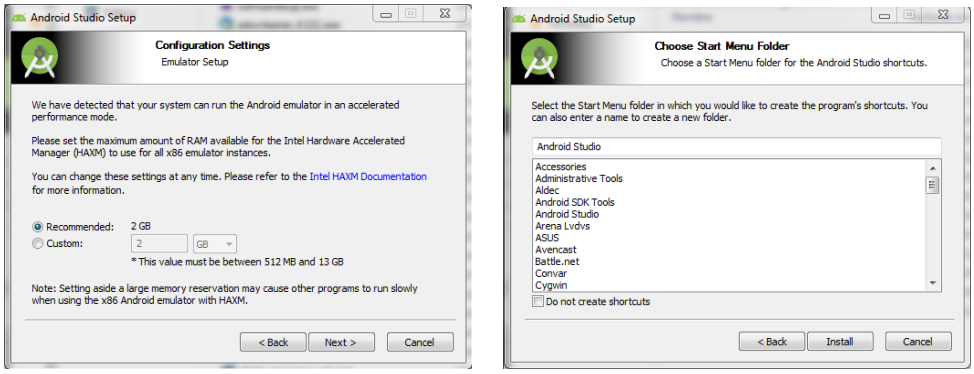

Figura II.7.: Instalación de Android Studio, paso 3.

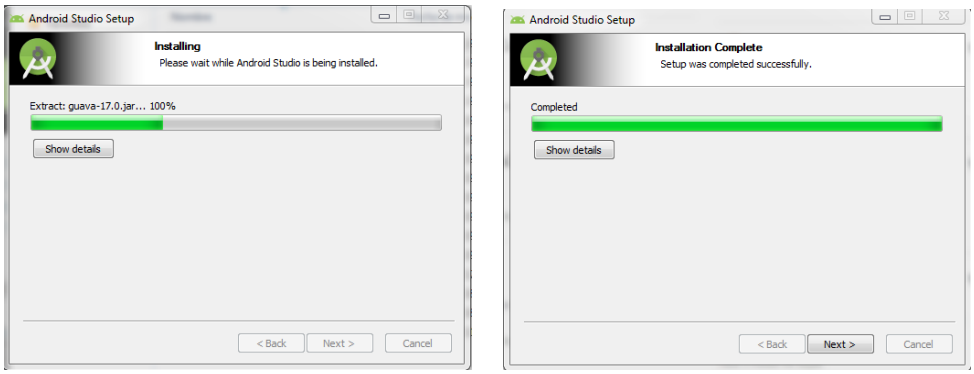

Figura II.8.: Instalación de Android Studio, paso 4.

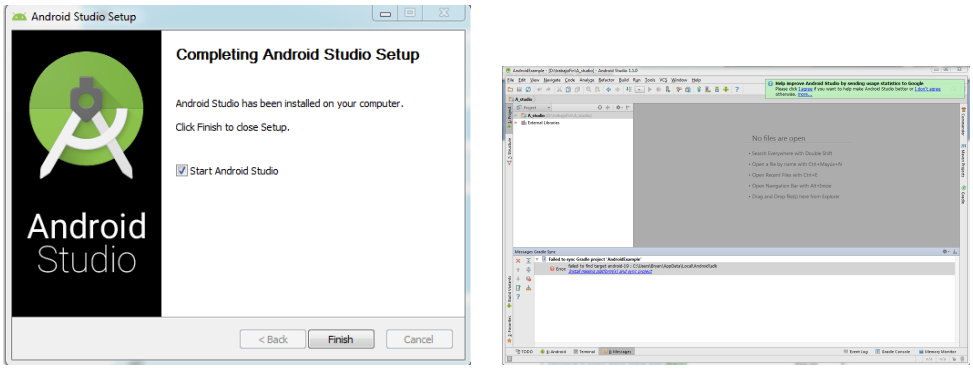

Figura II.9.: Instalación de Android Studio, paso 5.

nuestro *Adroid Studio*, debemos pulsar al botón *next* para empezar nuestra instalación. Entonces pasamos a la siguiente pantalla y nos da la opción de elegir qué elementos instalar, nosotros hemos dejado todos marcados pero puede que algunos no sean obligatoriamente necesarios. Una vez seleccionados los componentes de nuestra instalación pasamos a dar el botón *next* de la pantalla, entonces saltamos al paso 2.

El segundo paso de nuestra instalación lo podemos observar en la imagen II.6. Una vez hemos pasado la selección de componentes en nuestro *Android Studio*, tenemos que aceptar la licencia del software, pasaremos a leer la licencia, si es de nuestro interés, y cuando estemos de acuerdo con ella, pulsaremos el botón *I Agree*. Esto no habrá llevado a la siguiente pantalla de la instalación, en la que podremos elegir los directorios en los que instalaremos nuestros componentes, nosotros los hemos dejado por defecto aunque se pueden personalizar en función de las necesidades del usuario. Una vez establecida la configuración de directorios en la instalación, pulsaremos el botón *next*.

Pasaremos entonces al tercer paso de la instalación, la cual puede ser vista en la imagen II.7. En la primera pantalla de este paso tendremos que elegir la cantidad de memoria *RAM* que asignaremos a nuestro *Android Studio*, podemos dejar la cantidad recomendada o podemos asignarle más cantidad, si disponemos de ella, para que tenga un funcionamiento más fluido. Una vez establecido esto, se pulsara el botón *next*. En nuestra segunda pantalla nos pregunta el directorio en el que se iniciará la aplicación y puedes poner la opción de no crear un acceso directo, nosotros hemos dejado la configuración por defecto. Finalmente pulsamos el botón *Install*.

Ahora en el paso 4, el cual está representado en la imagen II.8, solamente tendremos que observar cómo avanza la instalación, esto se nos irá mostrando a través de una barra de progreso. Una vez se la barra se ha llenado podremos hacer *click* en el botón *Next* para pasar al último paso de nuestra instalación.

En nuestro quinto y último paso, el cual podemos ver en la imagen II.9, se nos informará de que la instalación ha terminado y tendremos la opción de iniciar nuestro *Android Studio* nada más acabar la instalación. Entonces pulsamos al botón *Finish*, se nos abrirá nuestro *Android Studio* y con esto la instalación se considera terminada.

BIBLIOGRAFÍA (LIBROS Y ARTÍCULOS)

[1] Alex Berg, Jia Deng, and L Fei-Fei. Large scale visual recognition challenge 2010, 2010.

[2] Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[5] Asha Iyer, Christof Koch, and Pietro Perona. What do we perceive in a glance of a real-world scene? In *J Vision*.

[6] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[8] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pages 1–42, 2014.

[10] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.