Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Detección de logos en imágenes mediante Redes Convolucionales

---

Marc González Fernández

Director: Brais Cancela
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 3 de juliol de 2017

Abstract

This memory will talk about an application for Android and ios devices that will look for events or nearby places that will offer discounts and/or rewards interacting either click or taking photos of a certain logo.

The project is separated in two parts. The first part is about how the app and the server have been developed, explaining its features and the technologies that have been used. This document will make an introduction on this, but will be focused on the second part, image recognition through Convolutional Neural Networks. We tested two different networks, a custom network advised by our tutor to this specific situation, and another net based on a pyramidal structure called AlexNet.

We have prepared a dataset to build the networks. Our dataset contains 11 classes with an average of 500 pictures per class. The networks have been configured with different parameters and we'll compare the results obtained to choose one. We'll also explain the different techniques tested to improve predictions.

Finally, we'll talk about limitations, improvements and extensions that the app could have, such as new events or to improve the current dataset.

Resumen

Esta memoria hablará sobre una aplicación para dispositivos Android y ios que buscará eventos o lugares cercanos que te ofrecerán descuentos y/o recompensas interactuando de alguna forma, ya sea mediante clic o tomando fotografías de un logo concreto.

El proyecto está dividido en dos partes. Una primera parte tratará sobre el desarrollo de la aplicación y el servidor, mostrando las diferentes funcionalidades que ofrece y explicando las tecnologías usadas. Este documento hará una introducción sobre esta parte, pero se centrará en la segunda parte, la detección de logos, que se realizará mediante redes neuronales convolucionales. Para ello se han probado dos redes diferentes, una primera red personalizada con ayuda del tutor para esta situación concreta y una segunda red piramidal llamada AlexNet.

Para construir las redes se ha preparado un dataset con 11 clases y una media de 500 imágenes cada una. Se configurarán las redes con diferentes parámetros y se contrastarán los resultados para ver como se ha tomado la decisión. También se explicarán las diferentes técnicas probadas para mejorar los resultados de la detección de logos.

Finalmente, se hablará sobre las limitaciones, futuras mejoras y ampliaciones que puede tener la aplicación, como nuevos tipos de eventos o ampliar el dataset.

Resum

Aquesta memòria parlarà sobre una aplicació per a dispositius Android i aios que cercarà esdeveniments o llocs propers on t'oferiran descomptes i/o recompenses interactuant d'alguna forma, mitjançant clic o fent fotografies de logos concrets.

El projecte està dividit en dues parts. Una primera part tractarà sobre el desenvolupament de l'aplicació i el servidor, mostrant les diferents funcionalitats que ofereix i explicant les tecnologies utilitzades. Aquet document introduirà la primera part, però es centrarà en la segona part, detecció de logos, que es realitzarà mitjançant xarxes neuronals convolucionals. Per aconseguir-ho s'han provat dues xarxes diferents, una primera xarxa personalitzada amb ajut del tutor per aquesta situació concreta i una segona xarxa piramidal anomenada AlexNet.

Per construir les xarxes s'ha preparat un dataset amb 11 classes i una Mitjana de 500 imatges cadascuna. Es configuraran les xarxes amb diferents paràmetres i es contrastaran els resultats per veure com s'ha pres la decisió. També s'explicaran les diferents tècniques provades per millorar els resultats de la detecció de logos.

Finalment, parlarem sobre les limitacions, millores i ampliacions que podria tindre l'aplicació, com nous tipus d'esdeveniments o una ampliació del dataset.

Contenido

Lista de figuras

1. Introducción

¿Alguna vez has estado pensando en un plan para el fin de semana y no se te ha ocurrido nada?

¿Te has enterado de un evento el día después al verlo en el periódico?

¿Te has desplazado a un lugar que no conoces y no sabes en que restaurante comer?

Son situaciones en las que te has podido encontrar varias veces. Pero estoy convencido de que ahora mismo tienes un ordenador o algún tipo de dispositivo a tu alcance. ¿No sería más fácil que este dispositivo pudiera planificar tu fin de semana o te sugiriera un lugar dónde comer?

En este proyecto se ha desarrollado una aplicación móvil que te informará de eventos o lugares cercanos. Además, podrás conseguir descuentos u otros tipos de recompensas.

La idea inicial era hacer una aplicación donde el usuario pudiera obtener recompensas al asistir a eventos, como puede ser un concierto, realizando una foto a un logo en concreto que sería indicado por la aplicación.

Una situación curiosa que pasó fue al asistir a un evento de League of Legends. Los organizadores comenzaron a repartir regalos lanzándolos a la grada. Pero esto puede llegar a ser injusto ya que no se reparte a todos los sectores por igual y se pensó en la posibilidad de que todo el mundo tuviera las mismas posibilidades de conseguir el regalo. De allí surgió la idea de, si se diera la recompensa a los 30 primeros que tomaran una foto de algo concreto en el recinto del evento sería más justo e incentivaría a interactuar con otras personas y el entorno, siendo motivacional y obteniendo un mejor “feedback” de tus clientes.

Pero, a medida que avanzaba, surgieron ideas como las paradas, un lugar donde al acercarte puedas obtener descuentos, sustituyendo a los repartidores de “flyers”.

El proyecto está dividido en dos partes, una primera donde se explicará las características de la aplicación, las funcionalidades que ofrece y su diseño y, una segunda parte, de cómo la aplicación es capaz de reconocer los logos.

En este documento se hará un breve resumen de la primera parte para conocer las posibilidades que ofrece la aplicación y que tecnologías se han usado para llevarla a cabo, y luego se centrará en la segunda parte, la detección de logos, donde se explicará cómo se han usado redes neuronales convolucionales para conseguir el reconocimiento del logo y que procedimiento sigue desde que se toma la foto hasta que se recibe una respuesta de la predicción.

2. Objetivos y motivaciones

La mayor motivación es aprender cosas nuevas y un proyecto así te permite conocer las nuevas tecnologías con el apoyo del tutor y docentes del centro que tienen experiencia en este ámbito. El aprendizaje debe acompañar a una persona durante toda su carrera ya que no se conoce límite en la tecnología y siempre hay que estar abierto a nuevos conocimientos.

El objetivo del proyecto es realizar una aplicación compatible con dispositivos Android y i-Phone capaz de reconocer logos. Esto se conseguirá gracias a las redes neuronales convolucionales (CNN).

A parte, esta aplicación tendrá otras funcionalidades como buscar eventos y conseguir recompensas y otros objetivos a futuro ya que tan solo se ha puesto un primer ladrillo que te ofrece un montón de posibilidades.

Aunque el resultado final no es el esperado debido a que no se dispone del hardware necesario y un banco de fotos suficiente para ser más precisos en la predicción, se intentará hacer énfasis en las decisiones tomadas durante el transcurso del proyecto y cómo se podría mejorar el resultado obtenido.

3. Desarrollo

Para empezar, se hará una breve introducción sobre las partes que forman la aplicación y, seguidamente, nos centraremos en el funcionamiento y las herramientas usadas para la detección de logos.

3.1. ¿En qué consiste el proyecto?

Como se ha mencionado, se trata de una aplicación para dispositivos Android y i-Phone que su función principal es participar en eventos y ganar recompensas tomando fotografías de los logos solicitados.

Esta aplicación se ha dividido en tres partes:

1) Front-end nativo usando Ionic.
2) Back-end con Django (python) + REST
3) Clasificación de la imagen.

3.1.1. Front-end

Se trata de la capa presentación con la que el usuario interactuará. Para esta capa se ha utilizado Ionic que consiste en un lenguaje nativo para poder compilar un mismo código fuente tanto en Android como ios. En un principio se miró de usar Brython debido a su sinergia con el back-end (se trata de un framework de Python) pero finalmente nos decantamos por Ionic, ya que tiene mucho más potencial siendo uno de los frameworks híbridos más importantes del momento y con una base y compatibilidad con Cordova.

Para este proyecto hemos elegido usar la versión 1 de Ionic que trabaja con AngularJS 1, un framework MVC de JavaScript para desarrollar Web front-end, ya que, aunque sea más complicado debido a su poca información (tanto en web como el soporte que ofrece cuando hay un error, que nos notifica con mensajes muy genéricos), nos sentíamos más cómodos para realizar un proyecto en poco tiempo que aprendiendo un nuevo lenguaje, TypeScript, incluida en AngularJS 2 (usado por Ionic 2).

La aplicación inicialmente mostrará una ventana de Login (ver Fig. 3.1) que comprobará en el back-end si el usuario está registrado o no.

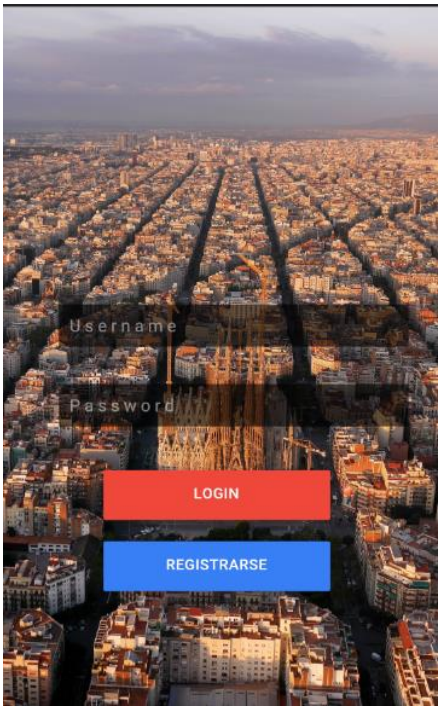


Figura 3.1. Ventana de Login

También tiene su correspondiente registro y una vez logeado mostrará la pantalla inicial (ver Fig. 3.2) que consiste en un mapa indicando tu posición y la de todos los eventos cercanos. También hay un menú en la esquina superior derecha que mostrará opciones como ver/modificar tu perfil, ver las recompensas obtenidas o cerrar sesión.

En el mapa se visualizarán dos tipos de eventos:

Evento parada: Al clicar sobre el icono obtendrás una recompensa.

Evento de búsqueda: Al clicar podrás participar y te pedirá tomar una foto de un logo concreto.

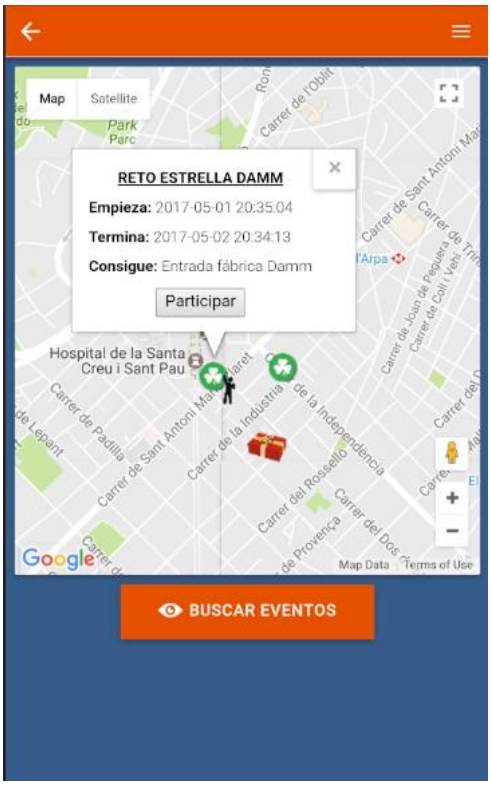


Figura 3.2. Ventana inicial

También hay una ventana donde se mostrarán todos los eventos y los podrás filtrar por fecha y localidad para planificar tu día.

Como se puede ver, se trata de una aplicación fácil de usar que permite al usuario poder realizar planes tanto a largo plazo como a corto plazo y con el incentivo de ganar recompensas con su uso.

Todavía es una aplicación en desarrollo con un abanico enorme de posibilidades con las que mejorarla. Más adelante se hablará de posibles ampliaciones a nivel de usabilidad y funcionalidad.

3.1.2. Back-end

En esta sección se hablará sobre cómo se gestionan los datos entre el usuario y la base de datos.

El back-end se trata de un servidor Django, framework de Python, con API REST que dependiendo la llamada que reciba, realizará unas consultas a la base de datos y presentará el resultado en formato JSON para que lo recoja el cliente.

También se ha creado una interfaz de administrador para poder gestionar usuarios y eventos de forma rápida:

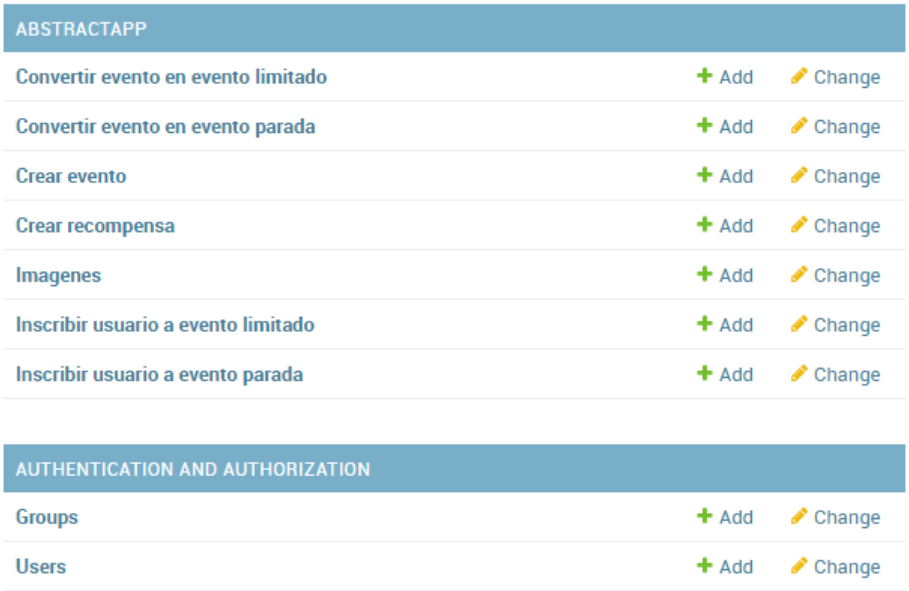


Figura 3.3. Ventana de administrador

Por ahora solo podrán acceder administradores, pero en un futuro se podría plantear la opción de permitir un acceso limitado a clientes de forma que puedan personalizar sus eventos.

Otra particularidad de Django es su estructura de directorios que te da facilidades para entender y desarrollar aplicaciones más rápidamente.

Por defecto separa el código de forma que tengas un fichero urls.py encargado de llamar al método correspondiente a la dirección url que ha recibido, views.py

que se encargará de la lógica y realizará las consultas necesarias a la base de datos y, el fichero models.py, donde generas las tablas de la base de datos.

Además de esos ficheros, también hay otros opcionales como admin.py para personalizar el menú de administrador o serializers.py para serializar datos.

Cada vez que modifiques las tablas de la base de datos (models.py), hay que realizar los comandos **makemigrations** y **migrate** para actualizarla.

Django también te ofrece la herramienta ORM, una herramienta muy útil para realizar consultas a la base de datos ya que, en vez de escribir *SELECT * FROM…* puedes tratarlos como objetos y escribir:

NombreTabla.objects.filter(CampoTabla = "ValorCampo")

Una forma más fácil de interpretar, más segura y que puede ahorrarte muchas líneas de código.

3.1.3. Clasificación de la imagen

Por último, una introducción sobre el procedimiento realizado para poder clasificar una imagen entre las clases de logos que hay.

Puede dividirse en tres partes:

- Primero hubo un proceso de selección de imágenes, llamado scraping, que consiste en unos scripts para obtener imágenes de los logos que queremos reconocer y formar un dataset (más adelante hablaremos del concepto dataset).

- Por otra parte, se han construido redes neuronales, necesarias para la detección de logos, usando Lasagne, un framework de Python para construir y entrenar estas redes neuronales en Theano. Theano es una librería de Python para calcular expresiones matemáticas de arrays multidimensionales más eficientemente usando la GPU.

  También se estuvo mirando la posibilidad de usar Keras, se trata de otro framework de Python, pero entonces daba muchos problemas de compatibilidad con Windows y Python 3. Actualmente ha mejorado mucho en este aspecto y podría ser una buena opción para futuros proyectos.

- Por último, tenemos la conexión con el servidor Django, que cada vez que le pasemos una imagen nos devolverá la respuesta que dependerá de si la imagen es la que buscábamos o no.

3.2. Recorrido de la imagen.

Hasta ahora se han definido por encima las diferentes capas que forman la aplicación, pero ¿cómo funcionan en conjunto?

Vamos a entrar más en detalle sobre el recorrido que realiza una imagen desde que es tomada hasta que devuelve el resultado. La siguiente imagen (Fig 3.4) muestra el recorrido de forma esquematizada.

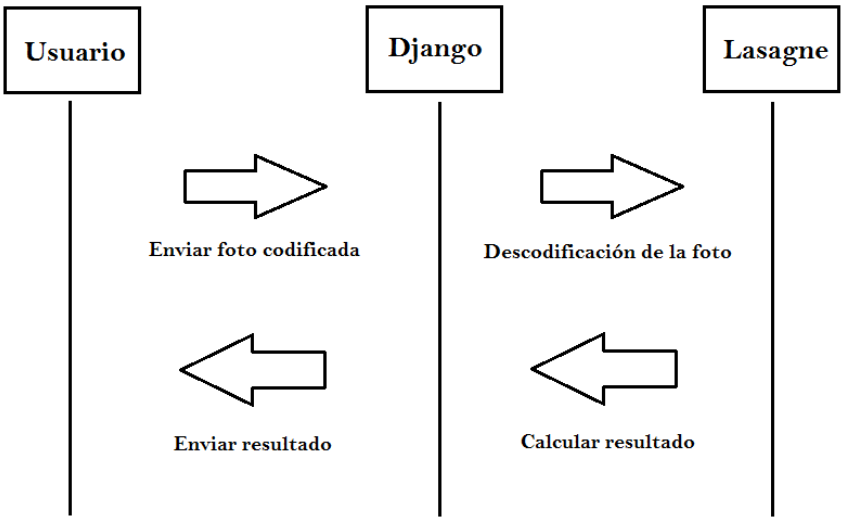


Figura 3.4. Esquema del recorrido de una imagen

Una vez el usuario toma una fotografía, ésta se envía codificada en Base64 al servidor Django mediante una petición POST. Django comprobará que el usuario sea correcto a partir de un token y comprobará que este usuario no haya ganado anteriormente el evento por seguridad.

Entonces, descodificará la imagen y llamará un script que se encargará de comprobar si esta imagen corresponde con la pedida en el evento y devolverá una respuesta true o false dependiendo del resultado.

En caso de ser la foto que se buscaba, el servidor modificará en la base de datos la información referente a la participación en el evento y la recompensa obtenida por el usuario.

Finalmente, devolverá el resultado obtenido a la aplicación del usuario y ésta mostrará un mensaje en función de si ha ganado o no.

3.3. Detección de logos a partir de una imagen.

En esta sección se hablará del procedimiento realizado para que cuando enviemos una imagen al servidor, éste sepa identificar si en ella está el logo buscado. Este proceso se logrará gracias al Deep Learning y, en concreto, se hablará de las redes neuronales convolucionales (CNN), que actualmente es el mejor mecanismo para el reconocimiento de imagen.

3.3.1. Elaborar un dataset

Los dataset son una pieza fundamental y no es más que un conjunto de datos para entrenar nuestro modelo. En nuestro caso, por ejemplo, nuestro dataset está formado por imágenes y sus correspondientes labels que definen a la clase que pertenecen.

Para conseguir imágenes que formen el dataset se investigó sobre “Scraping”. El scraping se refiere a la acción de extraer información de un sitio Web. En nuestro caso sería aplicado para imágenes. Consiste en scripts o programas que descargan grandes cantidades de imágenes a partir de un URL. La mayor parte de nuestro dataset está formado por imágenes extraídas con esta técnica. Después de investigar, la solución más rápida para conseguir una base de imágenes para el proyecto fue usando la aplicación "Bulk Image Downloader” (Fig. 3.5).

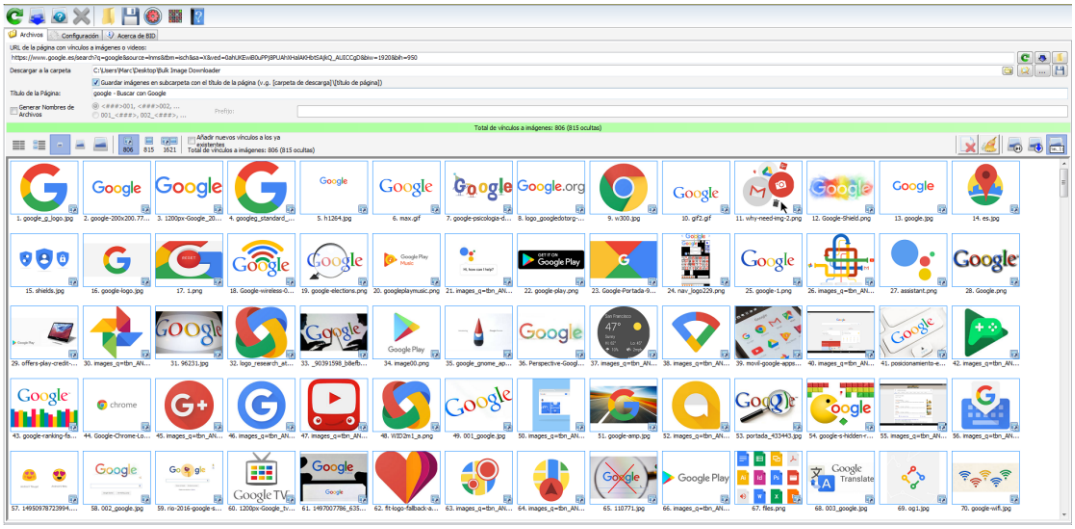


Figura 3.5. Bulk Image Downloader buscando imágenes de Google

Esta aplicación consiste en pasarle una URL y te encuentra imágenes que contenga dicha página. Tiene uso gratuito limitado con el que solo puedes descargar imágenes de 100 en 100 pero para empezar ya es suficiente.

Otro método para conseguir imágenes fue el tradicional, hacer fotos tú mismo. Aunque obviamente esta técnica solo sirve para complementar, ya que para realizar miles de fotos se necesita invertir mucho tiempo.

Muchas imágenes también se tuvieron que recortar debido a que el logo a proporción ocupaba una parte muy pequeña de la imagen y para el entrenamiento es necesario que al menos un 60% de la imagen contenga el logo ya que es lo que queremos que reconozca.

El dataset contiene aproximadamente 6000 imágenes (11 clases con ~500 imágenes cada una excepto random que contiene algunas más).

Las clases elegidas para el proyecto han sido Apple, Carrefour, Crunch, Damm, KFC, Lays, M&M's, Pepsi, Snickers, Twix y una última clase llamada "Random" que contiene imágenes aleatorias consiguiendo una dispersión que hará que está clase sea elegida en fotografías que no tengan relación con los logos.

Pero ¿Cómo se representa una imagen en el dataset?
Dentro del dataset almacenamos el valor de cada píxel. Trabajamos con imágenes de tamaño 80x80 con 3 canales (RGB), por lo tanto, cada imagen ocupa 80x80x3 posiciones o píxeles. Para lograr que las imágenes cumplan estas condiciones se ha diseñado un script que las recorra y cambie sus dimensiones a 80x80 además de convertirlas en RGB en caso de no serlo.

Esta parte fue una de las que requirió más tiempo ya que tener que construir tu propio dataset desde cero requiere muchas horas de dedicación para seleccionar imágenes específicas y retocarlas, incluso muchas descartarlas, por su calidad o dimensiones muy pequeñas.

3.3.2. Construir una red convolucional

Bien, ya sabemos que es un dataset pero ¿cuál es su función?

El dataset tan solo es un conjunto de imágenes, pero tiene más importancia de la que parece. A partir de estas imágenes podremos construir nuestra red convolucional que en un futuro predecirán si una fotografía contiene el logo que buscábamos.

Antes de empezar a explicar cómo funciona nuestra red, hacer un último inciso sobre el dataset. Nuestro dataset estará dividido aleatoriamente en dos conjuntos: Training set y Test set.

El Training set contendrá un 85% del total de las imágenes del dataset. Este conjunto se usará para entrenar nuestra red de la que ahora hablaremos.
El 15% restante de las imágenes corresponden al Test set. Su función será evaluar la precisión de nuestra red. Es importante separar bien estos dos conjuntos ya que, si tuvieran imágenes repetidas, manipularían el resultado final, y carece de sentido usar las mismas imágenes con que se ha entrenado el modelo para evaluar su eficacia.

Dicho esto, vamos a adentrarnos en las redes neuronales.

Las redes neuronales, intentan simular el comportamiento de las neuronas. En nuestro caso, la red recibirá como entrada una de las imágenes del Training set e irá abstrayendo características relevantes de la imagen que ayuden a predecir si la fotografía contiene este logo. Para seguir con la explicación sería interesante ver la siguiente imagen:

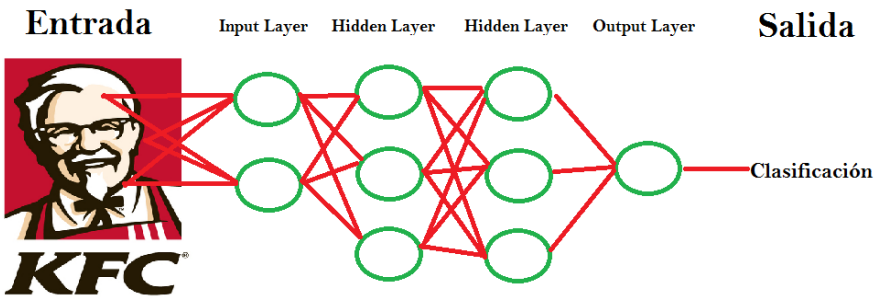


Figura 3.6. Ejemplo de red neuronal

La figura 3.6 representa una red neuronal. Como se ha comentado, la red recibe una imagen como entrada y durante el recorrido se pasa por varias capas o “layers”. Cada capa tiene su función y al final del recorrido tenemos el resultado, que se trata de nuestra predicción. En cada capa también encontramos “neuronas”, cada una de ellas se trata de una característica o descriptor.

Pero el ejemplo mostrado tiene un problema. Una imagen tiene tres dimensiones: altura, anchura y canales. Si queremos que cada neurona contenga cada píxel de la imagen, generaría una cantidad de parámetros imposible de tratar. Suponiendo que es una imagen 80x80, ya serían 6400 parámetros por neurona. Fácilmente podríamos llegar a millones de parámetros en solo la primera capa.

Para solventarlo usaremos un tipo de redes específico, las CNN o redes neuronales convolucionales.

A diferencia de las redes convencionales, en las CNN cada neurona se responsabilizará de una pequeña región NxN. Estas regiones por lo general no serán muy grandes sino perderemos muchos detalles al aplicar la convolución.

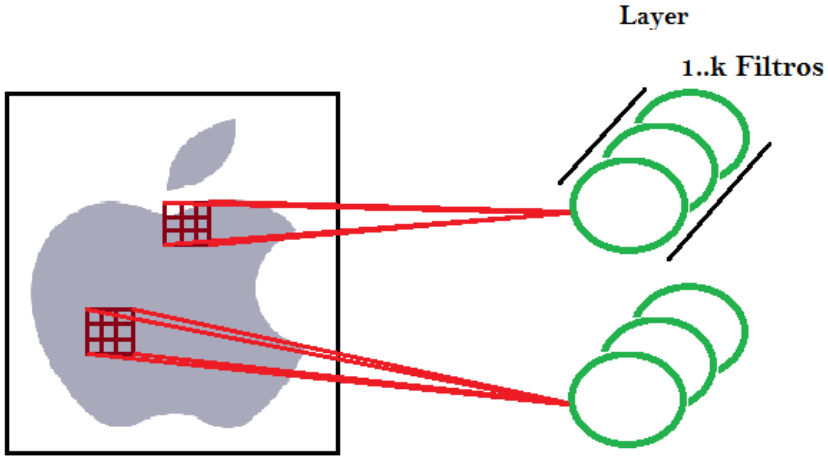


Figura 3.7. Input de las neuronas

Como se muestra en la Figura 3.7, cada neurona se encargará de un píxel y los de su alrededor, formando una región 3x3. Ese mismo píxel tendrá tantas neuronas asociadas como filtros de características tengamos.

Las capas tienen una función de entrada que realiza el producto escalar del vector de pesos por el vector de entrada y en la salida una función de activación que introduce no linealidades. Las capas tienen cada vez descriptores más complejos de forma que la primera capa podría detectar los contornos de la imagen y la última capa ser capaz de detectar formas que permitan clasificar correctamente la imagen.

Un tipo de capa importante en estas redes son las de convolución. La convolución consiste en mezclar la entrada de la neurona con un filtro de características. A continuación, se mostrará un ejemplo de convolución.

Suponiendo que tenemos la siguiente imagen (en forma matricial de los valores de cada canal):

Matriz 5x5

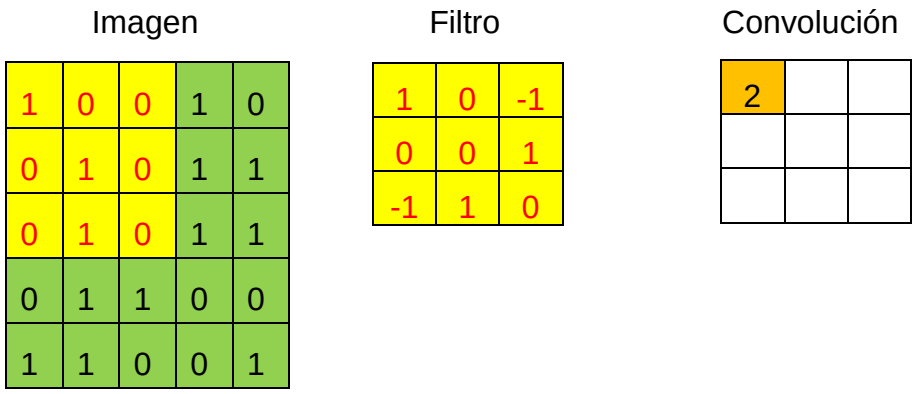
1	0	0	1	0
0	1	0	1	1
0	1	0	1	1
0	1	1	0	0
1	1	0	0	1

Y el siguiente filtro de características:

Matriz 3x3

1	0	-1
0	0	1
-1	1	0

Suponiendo que tenemos esta entrada y filtro, desplazaremos el filtro sobre la ventana de entrada multiplicando cada valor por el sobrepuesto y sumando los resultados:



Aplicando el filtro sobre esa primera región obtenemos 2. Si desplazamos el filtro sobre la imagen tendremos como resultado la siguiente matriz 3x3:

2	-1	2
1	1	-1
1	-1	-1

Además, también intervienen algunas variables que hemos ignorado y pueden variar el resultado:

- **Bias.** Es una constante que se añade para extender una capa. En algunos casos es necesario como puede ser un conversor entre Celsius y Fahrenheit, ya que multiplicando no podemos convertir 0ºC en 32ºF y, sumándole 1, ya se podría realizar. Este valor se suma a cada valor de la matriz final obtenida en la convolución.
- **Stride.** Siempre es mejor usar un stride de 1, pero se puede usar uno mayor para reducir los parámetros. Este valor es el desplazamiento del filtro sobre la matriz de entrada. Si el stride es 2, significa que cada iteración se saltará un valor y, por tanto, reduce la matriz final.
- **Zero-padding.** El padding también es importante. En nuestro ejemplo no hemos usado padding y por lo tanto se ha reducido el tamaño de la matriz original. A veces nos interesará mantener las dimensiones y la forma de conseguirlo es añadiendo ceros alrededor de la matriz entrada.

Para calcular el tamaño del volumen de la salida se aplica la siguiente fórmula:

(W – F + 2P) / S + 1

donde W es el tamaño de entrada, F es el tamaño del filtro, P es el Padding y S el Stride. Aplicando la fórmula para el ejemplo anterior, sería:

(5 – 3 + 2·0) / 1 + 1 = 3

El resultado es 3 que coincide con el tamaño espacial obtenido.

Ahora que ya sabemos un poco más de cómo funcionan las redes convolucionales, vamos a hablar del Framework utilizado y que opciones nos ofrece.

Como ya se ha comentado, se ha decidido usar Lasagne

Lasagne agiliza mucho el trabajo de construir una red ya que trae implementadas las capas principales y algunas funciones importantes:

- **InputLayer.** Capa de entrada. Se le pasa como parámetros los canales de la imagen, la anchura y la altura.

- **Conv2DLayer.** Capa de convolución. Le pasamos la capa anterior, el número de filtros y el tamaño de los filtros. También tiene como parámetros opcionales el stride, el padding y una función no lineal que, en nuestro caso, nos interesa la ReLU.

- **MaxPool2DLayer.** Capa MaxPooling. Necesita la capa anterior y el tamaño de la región de sub-muestro. También se le puede pasar un stride y un padding.

- **LocalResponseNormalization2DLayer.** Capa de normalización. Normaliza los mapas de características. Recibe como parámetro de entrada la capa anterior y se puede modificar los valores alfa y beta de la fórmula.

$$x_i = \frac{x_i}{(k + (\alpha \sum_j x_j^2))^\beta}$$

Figura 3.8. Fórmula de normalización

- **DenseLayer.** Capa Fully-Connected. Recibe la capa anterior y el número de unidades. Si se trata de la última capa, el número de unidades será la cantidad de clases que entrenamos. También puede recibir una función no lineal que para una capa final nos interesa la Softmax.

La red que usaremos está formada de nueve capas:

- **Capa de entrada:** Capa que recibe una imagen de dimensiones 80x80x3.

- **Capa Conv+ReLU (x3):** Capa que realiza las convoluciones seguido de una función rectificadora, ReLU, que consiste en convertir todos los valores negativos a cero (Fig. 3.9).

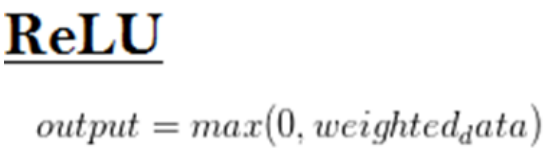


Figura 3.9. Función Lineal Rectificada

- **Capa Max-pooling 2x2 (x3):** Capa de sub-muestreo que se realiza justo después de cada capa convolutiva. Agrupa en parcelas de 2x2 los valores obtenidos y se queda con el mayor valor. De esta forma se va reduciendo la resolución de la imagen, pero manteniendo la información que necesitamos. También se puede hacer con el promedio en vez del máximo valor.

- **Capa Dropout+ReLU:** Capa que consiste en reducir, en nuestro caso a la mitad, nuestra red. Elimina la mitad de los nodos aleatoriamente para evitar overfitting. Overfitting es cuando un modelo se vuelve muy complejo que entorpece la predicción.

- **Capa de salida (FC+Softmax):** Capa que devuelve el resultado de cada clase. FC (Fully-connected) consiste en conectar todas las neuronas de la capa anterior con cada una de las neuronas de esta capa. Se aplica un softmax para normalizar el resultado de la predicción devolviendo valores entre 0 y 1, donde la suma de todos los valores es igual a 1.

La salida de nuestra red siempre consistirá de un vector con tantos valores como clases hay (con clases nos referimos a las diferentes soluciones que puede interpretar nuestra red, en nuestro caso, los logos que ha aprendido), donde la suma de todos los valores equivale a 1 (fórmula Softmax mencionada anteriormente). Estos valores corresponden a la probabilidad que hay de que sea esa clase y es importante que la suma de todas las probabilidades sea 1 para poder interpretarla.

Si el resultado nos devuelve para un logo un valor de 0.8 significa que hay un 80% de posibilidades que la imagen contenga ese logo. Pero no hay que olvidar que se trata de una predicción y las predicciones pueden fallar. El objetivo es mejorar la predicción para que tenga la mayor tasa de acierto posible.

3.3.3. Entrenamiento del modelo

Una vez tenemos la estructura del modelo, es momento de entrenarlo.

Primero le pasaremos al modelo las imágenes del Training set junto a sus labels correspondientes, de forma que el modelo encuentre características que coincidan entre imágenes de la misma clase.

Una vez hecho, le pasamos las imágenes del Test set para evaluar la predicción. A partir de su clasificación, sacaremos el porcentaje de acierto que tuvo en cada clase y el acierto global.

Una forma de visualizar los resultados y ver que clases tienen menor acierto y con qué clases se confunden es mediante la matriz de confusión (ver Fig. 3.10).

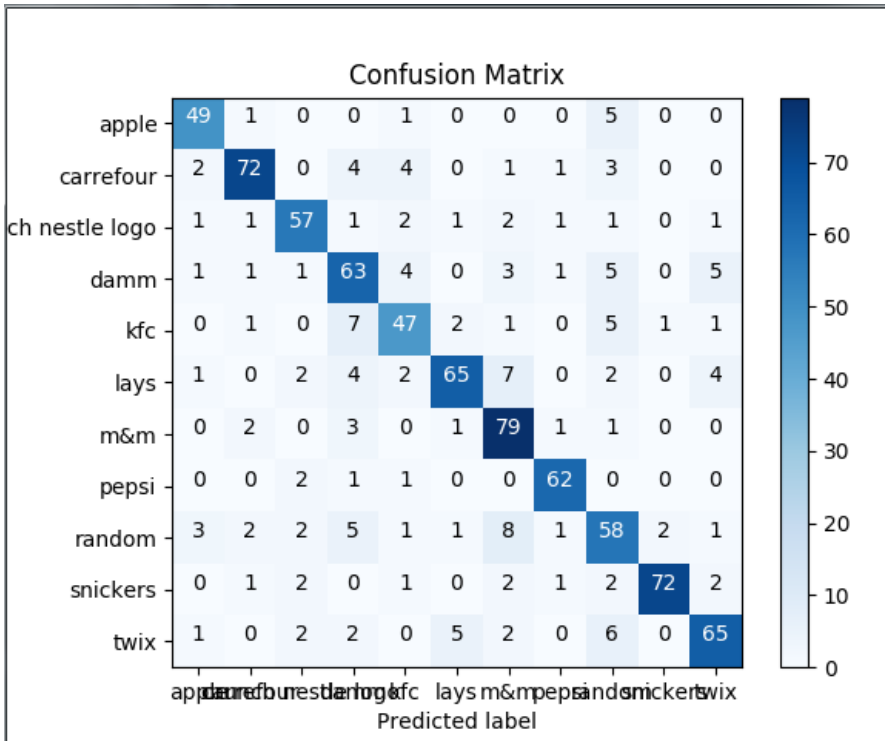


Figura 3.10. Ejemplo de matriz de confusión

La matriz de confusión muestra en las columnas lo que ha predicho y en las filas lo que realmente es, por lo tanto, tendremos en la diagonal todos los aciertos.

Si por ejemplo tenemos el siguiente caso:

	Apple	Twix
Apple	17	5
Twix	7	21

Figura 3.11. Ejemplo de matriz de confusión para explicar su funcionamiento

Podemos decir que acertó 17/22 (77%) para Apple y 21/28 (75%) para Twix. Además, gracias a esta matriz, sabemos que 5 imágenes de Apple fueron mal etiquetadas como Twix (23%) y 7 imágenes de Twix fueron mal etiquetadas como Apple (25%).

En global, hemos tenido un 76% de acierto, y con esta matriz sabemos qué clases han fallado más y con qué clases las confundieron.

Volviendo a la Figura 3.10, se puede ver que la clase random tiene un acierto del 69% mientras que m&m tiene un acierto del 90%, así que sabemos que la clase random necesita ser reforzada y que es confundida con frecuencia con las clases m&m y kfc.

4. Implementación

En esta sección se hablará de los requisitos necesarios del proyecto. Durante la explicación ya se habló de algunos frameworks pero volveremos a recordarlos para tener un resumen de todo lo necesario en este proyecto.

4.1. Requisitos Software.

Para el desarrollo del proyecto se ha trabajado en entorno Windows 7, así que las siguientes herramientas son todas compatibles con dicho sistema operativo.

4.1.1. Android Studio

Esencial para programar aplicaciones Android. Aunque nuestra aplicación no es Android puro, este programa trae complementos muy útiles como emuladores para poder probar los cambios en caliente.

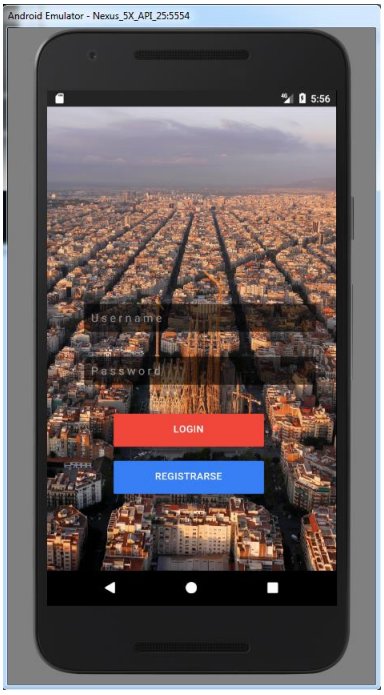


Figura 4.1. Emulador de Android Studio

4.1.2. Python

Como ya se ha comentado, toda la parte servidor está hecho en Python 3.4. Para trabajar en este lenguaje se ha usado la consola WinPython para ejecutar los scripts y el editor de texto Sublime Text 3 para modificarlos.

Estas son algunas librerías importantes que se han añadido (a parte de las comunes como os, numpy…):

- **Theano:** Usado por Lasagne para optimizar cálculos de GPU, a pesar de que nosotros no hemos podido sacarle rentabilidad a esto por no disponer de una buena GPU.

- **Lasagne:** Framework usado para construir y entrenar redes neuronales. Para más información puedes visitar su página oficial pulsando aquí.

- **Django:** Framework usado para desarrollar la parte servidor. Se trata de un framework de desarrollo web Open Source que sigue un patrón MVC.

- **Django rest framework:** Implementa una API REST a Django para poder realizar las operaciones POST y GET que nos ayudarán en el intercambio de datos.

- **Django-cors-headers:** Herramienta de Django que se encarga de las cabeceras del servidor en el compartimiento de recursos.

- **Librerías de imágenes:** También se han añadido librerías como Pillow y misc para trabajar con imágenes.

- **Base64:** Las imágenes se envían al servidor codificadas en Base64. Para poder descodificarlas, se ha usado esta librería.

4.1.3. Ionic

Ionic representa la parte cliente. Se trata de un framework para desarrollar aplicaciones Android y ios usando HTML5, css, angularjs y cordova.

En concreto, se ha usado la versión 1 de Ionic que viene con angular 1 porque ya se comentó que teníamos más experiencia que no con la segunda versión que se basa en TypeScript. Además, se han añadido los siguientes plugins de cordova para ahorrar mucho tiempo:

- **Cordova-plugin-camera:** Plugin que viene con funciones implementadas para acceder a la cámara del móvil.

- **Cordova-plugin-file:** Plugin que viene con funciones implementadas para acceder a los ficheros del dispositivo.

- **Cordova-plugin-whitelist:** Plugin que facilita la gestión del acceso a enlaces externos.

- **Cordova-plugin-geolocation:** Plugin para trabajar con la geolocalización del dispositivo.

- **Cordova-plugin-diagnostic:** Plugin para controlar el estado de las configuraciones de localización, wifi, cámara y bluetooth del dispositivo.

- **Cordova-plugin-compat:** Plugin para dar compatibilidad con versiones antiguas de cordova.

También se han utilizado algunos css gratuitos para mejorar la presentación de la aplicación.

4.1.4. GitHub

Para llevar un orden se ha trabajado con GitHub donde se encuentra tanto el código fuente como las imágenes. Además, se ha ido actualizando una Wiki con los objetivos, las ideas que iban surgiendo y los progresos del proyecto.

El repositorio solo consiste de una rama ya que al ser un proyecto de dos personas y trabajando en secciones diferentes del proyecto no vimos necesidad de crear ramas por tarea ni usar pull-requests.

Repositorio: https://github.com/kazusaki1/TFG

4.2. Redes convolucionales implementadas

Para este proyecto se han probado dos redes diferentes. La primera red fue propuesta por el tutor y ha tenido buenos resultados. Esta red está compuesta por las siguientes capas:

1) Como entrada recibe una imagen de dimensiones 80x80x3

2) La segunda capa aplicamos una convolución de 32 filtros de tamaño 3x3 seguido de una función ReLU para eliminar valores negativos.

3) Luego aplicamos una capa de max-pooling que obtendrá el valor máximo de parcelas 2x2 del resultado obtenido de la convolución.

4) Volvemos a aplicar una convolución, pero de 64 filtros y una función ReLU.

5) Otra capa de max-pooling.

6) De nuevo una convolución de 64 filtros con su correspondiente ReLU.

7) Otra capa de max-pooling.

8) Aplicamos una FC de 1024 unidades y una función de ReLU.

9) Dropout del 50% que pondrá la mitad de los valores de entrada a 0.

10) Otra capa FC de 1024 unidades.

11) Capa FC de 11 unidades (las clases que tenemos) y Softmax para normalizar el resultado.

Nuestra segunda red se trata de una Alexnet, que sigue una estructura piramidal. Está formada por las siguientes capas:

1) Capa de entrada 80x80x3.

2) Capa de convolución con stride de 4, sin padding y 3 filtros de 11x11 además de ReLU.

3) Capa de normalización con n = 5 que son los canales adyacentes.

4) Maxpooling de 3x3 con stride de 2.

5) Convolución con stride de 1, padding de 2 y 48 filtros de 5x5 además de ReLU.

6) Normalización con n = 5.

7) Maxpooling de 3x3 con stride de 2.

8) Convolución con stride de 1, padding de 1 y 256 filtros de 3x3 además de ReLU.

9) Convolución con stride de 1, padding de 1 y 192 filtros de 3x3 además de ReLU.

10) Convolución con stride de 1, padding de 1 y 192 filtros de 3x3 además de ReLU.

11) Maxpooling de 3x3 con stride de 2.

12) Dropout del 50%.

13) FC de 4096 unidades además de ReLU.

14) Dropout del 50%.

15) FC de 4096 unidades además de ReLU.

16) Dropout del 50%.

17) FC de 11 unidades y Softmax.

Más adelante contrastaremos resultados entre las dos redes para ver por qué nos hemos decantado por la primera.

También decir que no se ha podido probar las redes tanto como gustaría para ver que parámetros daban mejores resultados en nuestro caso, ya que al no disponer de una buena GPU se ha tardado días para entrenar cada red. Mientras que por CPU se ha tardado alrededor de 90 segundos para cada iteración, con una de las últimas GPUs del mercado se podría reducir el tiempo a 1s fácilmente.

4.3. Soluciones implementadas

Durante el desarrollo del proyecto, se han implementado algunas soluciones para intentar mejorar los resultados, aunque no todas han dado resultados positivos.

Desde el móvil nos llegan imágenes muy grandes y en las peores condiciones, hay que estar preparado para cualquier tipo de fotografía, tanto desenfocadas, desde diferentes ángulos o incluso imágenes que no están relacionadas con lo que buscamos.

Si bien sabemos que se manda la fotografía tomada al servidor, esta fotografía podría contener el logo, pero solo ocupando una cuarta parte de la imagen. Al intentar reconocerla, puede dar prioridad al fondo de la imagen y devolverte un resultado negativo ya que, aunque contenga el logo hay más coincidencias con otra clase del modelo.

Para ello se ha implementado una ventana que se desliza sobre la fotografía y realiza las predicciones sobre ventanas más pequeñas de la imagen. Primero redimensionamos la imagen a 80x80 y le pasamos tal cual la imagen. En la siguiente iteración le redimensionamos la imagen a 160x160 (siempre desde la imagen entrante que es más grande, si lo hiciéramos de la ya reducida a 80x80 perderíamos muchos detalles y se vería borrosa) y entonces le pasamos ventanas de 80x80 deslizándola 40 posiciones. Es decir, para esta segunda iteración se realizarán 9 predicciones. Se hará lo mismo con la imagen redimensionada a 240x240 y 320x320.

Se ha decidido no hacerlo con imágenes más grandes porque sino el servidor tardaría mucho en responder.

Este es el pseudocódigo que realizamos para recorrer la ventana deslizante:

```
Desde i = 1 hasta 4 hacer
     Leer imagen
     Proporción es igual al max(altura imagen, anchura imagen) / 80
     Redimensionar imagen(altura / proporción * i, anchura / proporción * i)
     Normalizamos imagen
     Desplazamiento ventana = 40
     Tamaño ventana = 80
     Recorremos altura imagen de 40 en 40
          Recorremos anchura imagen de 40 en 40
               Calculamos predicción de la ventana
               Si la predicción coincide con el logo que buscamos
                    Fin
```

Otra mejora implementada fue normalizar las imágenes de entrada, pero el resultado no fue el esperado y, a parte, daba muchos problemas en imágenes que tenían pocas tonalidades de colores, y al normalizarla podía cambiar el color completamente e incluso la imagen, así que este tipo de imágenes tenían que ser ignoradas para el proceso de normalización.

También hubo un problema al trabajar trabajar con imágenes borrosas, tanto recibidas por parte del usuario como cuando las tratamos, ya que al recibir las imágenes necesitamos redimensionarlas a valores más pequeños para poder recorrerlas con una ventana deslizante sin un consumo computacional excesivo.

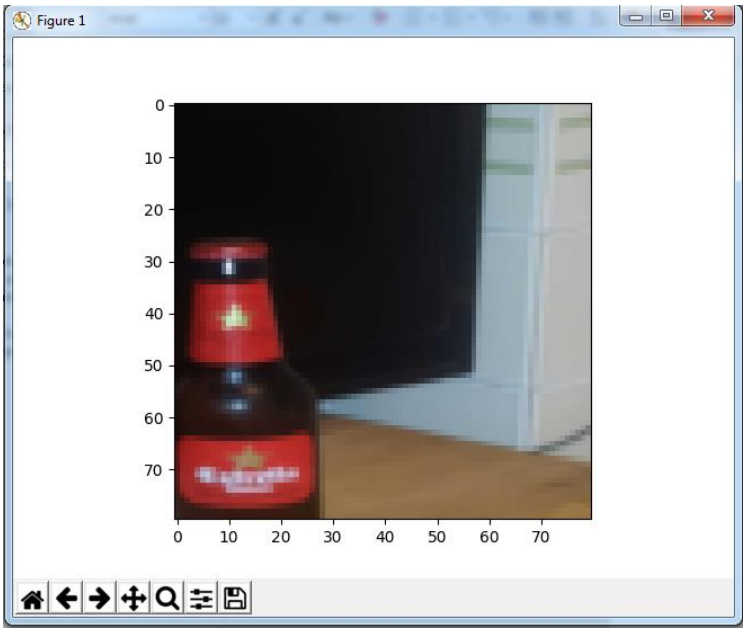


Figura 4.2. Ejemplo de imagen sin normalizar obtenida por una ventana deslizante de 80x80

5. Resultados

A continuación, se mostrarán los resultados obtenidos y se compararán las dos redes construidas para demostrar por qué se seleccionó la primera.

Como ya se ha dicho en varias ocasiones, el modelo todavía no es fiable ya que necesita muchas más imágenes, pero en la Figura 5.2 se puede ver un ejemplo de la misma imagen que obtuvo la ventana deslizante en la Figura 4.2 pero normalizada, suavizándola y mejorando los resultados, y con el resultado de su predicción que acertó con un 99,99%.

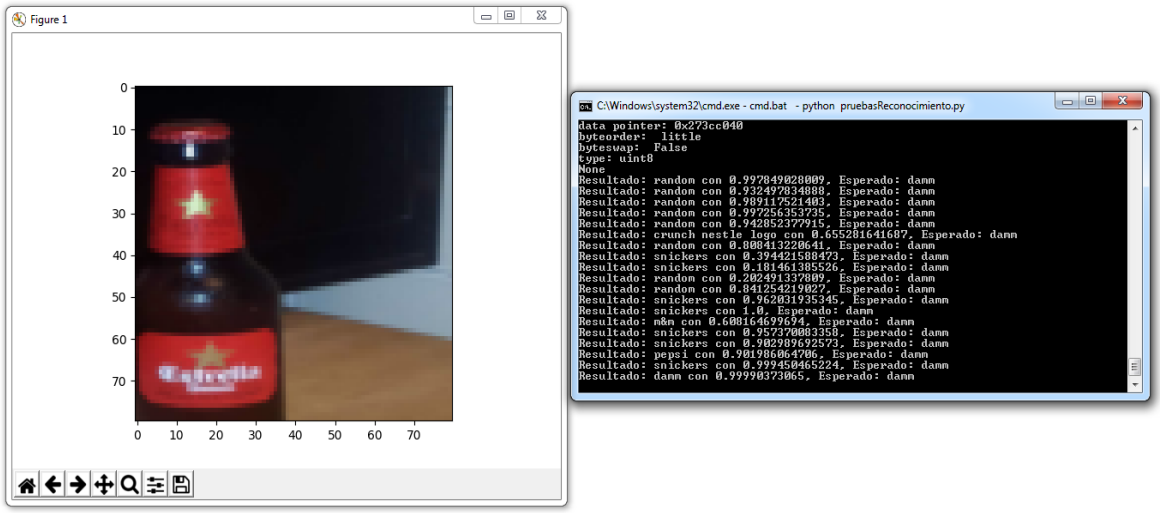


Figura 5.1. Ejemplo de reconocimiento del logo con imagen normalizada

Las predicciones anteriores corresponden a otras secciones de la imagen y en la mayoría acertó devolviendo la clase random o porcentajes inferiores al 70% que no daban un resultado claro de ser una clase específica, pero a veces sí que predecía por error que había encontrado otros logos. Por esta razón es importante ampliar el dataset, ya que entre mayor sea más casos diferentes contemplará y sabrá clasificar la imagen correctamente.

También, como se ha visto en la imagen anterior (Figura 5.1), si no aplicáramos la ventana deslizante no habría reconocido ese logo, porque se trata de una imagen de dimensiones 2322x4128, que al reescalarla a 80x80 (necesario ya que nuestra red ha sido entrenada con estas dimensiones) se vería borrosa como se muestra a continuación:

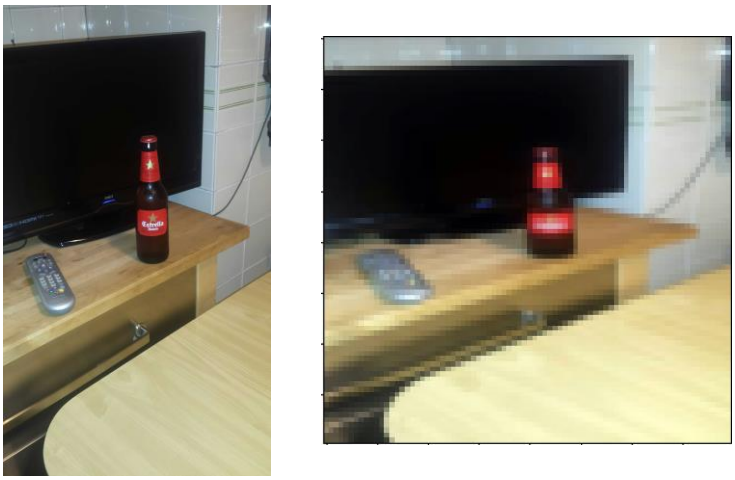


Figura 5.2. Comparación entre imagen real (izquierda) con imagen reescalada a 80x80 (derecha)

La ventana deslizante es fundamental para lograr que se reconozcan los logos. Se eligió que se desplazará de 40 en 40 posiciones ya que con un número más bajo se incrementaba mucho el coste de tiempo mientras que con un número mayor podría recortar el logo y solo verse la mitad, dificultando el reconocimiento (ver Figura 5.3).

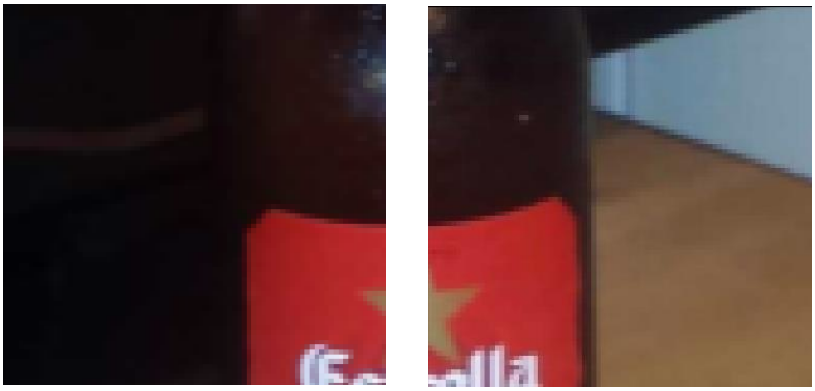


Figura 5.3. Resultado de dos iteraciones consecutivas con una ventana deslizante de 90

Ahora vamos a comparar las dos redes entrenadas.

Después de entrenarlos 500 iteraciones este ha sido el resultado de cada red:

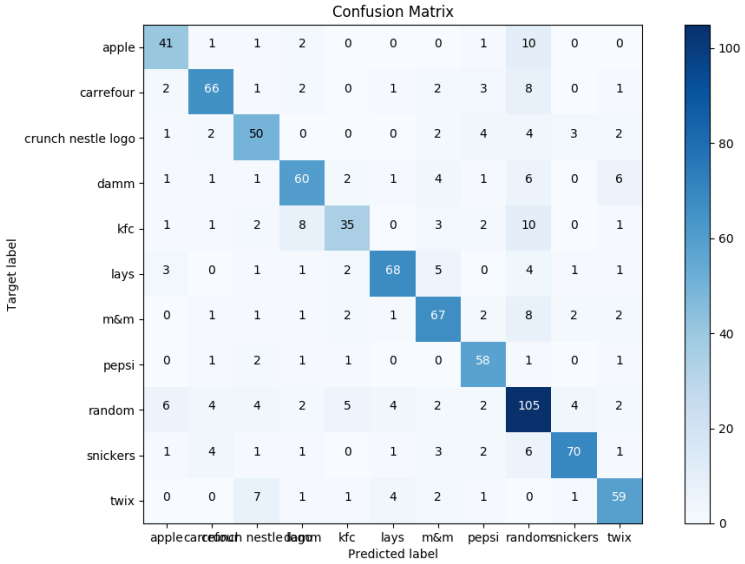


Figura 5.4. Matriz Confusión de red personalizada

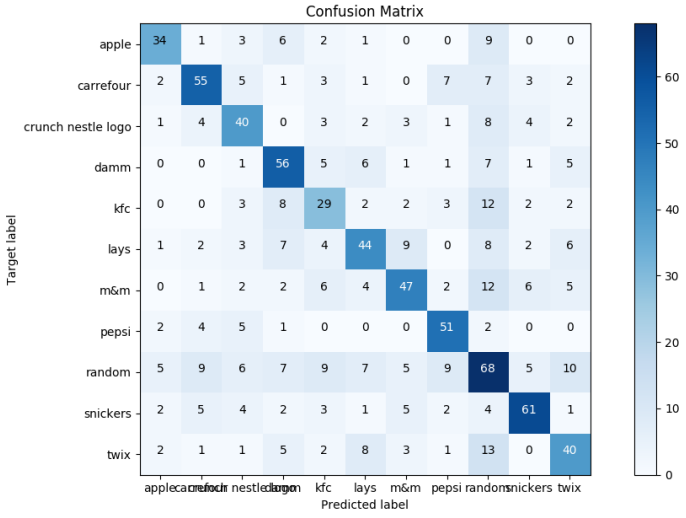


Figura 5.5. Matriz Confusión de AlexNet

En la Figura 5.4, vemos nuestra red personalizada. Esta gráfica muestra la tasa de acierto para cada clase. En global se ha tenido una tasa de acierto del 74,2%. Se puede ver que la clase que más problemas da es la random y por eso en el ejemplo anterior confundía otras clases con la random.

Por otro lado, en la Figura 5.5 tenemos la AlexNet. Ambas gráficas se hicieron con 500 iteraciones y se puede ver que esta red es más imprecisa. Tan solo obtuvo una tasa de acierto del 61,2%, pero sería interesante repetir este experimento con datasets más completos para ver si estos resultados varían.

Ahora veamos una comparación de las tasas de acierto por iteración:

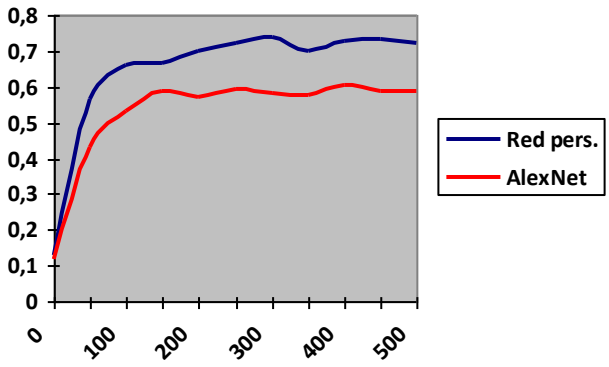


Figura 5.6. Comparativa de tasas de acierto

En la Figura 5.6, al lado izquierdo tenemos la tasa de acierto y abajo el número de iteraciones. A partir 200 iteraciones, la tasa de aciertos ya empieza a mantenerse y se puede ver que la tasa de aciertos de nuestra red personalizada es superior a la obtenida en la AlexNet, por eso se ha tomado la decisión se usar esta red.

Ahora vamos a hacer una comparativa del loss obtenido entre training y test de cada red. El loss es una suma de los errores cometidos y entre que menor sea significa que el modelo es mejor (siempre que no haya overfitting en el training). Estas son las gráficas:

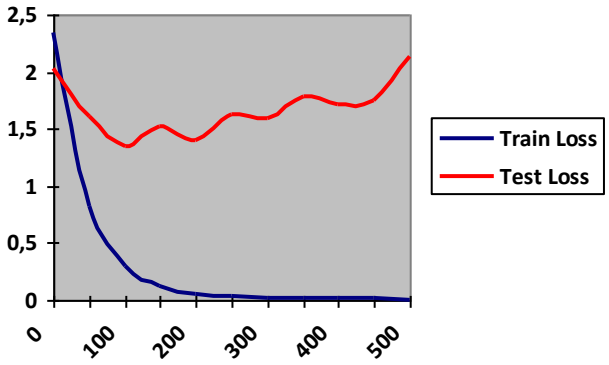


Figura 5.7. Comparativa de training y test loss en la red personalizada

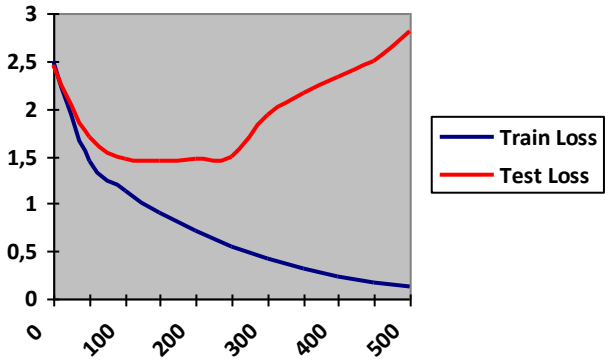


Figura 5.8. Comparativa de training y test loss en la AlexNet

En ambas imágenes, el loss de test acaba creciendo, sobre todo en la AlexNet, es un aspecto que se tendría que mejorar ya que afecta a nuestros resultados. Por otro lado, el training loss muestra un mejor aspecto, decrementándose en cada iteración.

También se probó normalizando las imágenes del training, pero como se muestra en la Figura 5.9, el resultado no afectó positivamente.

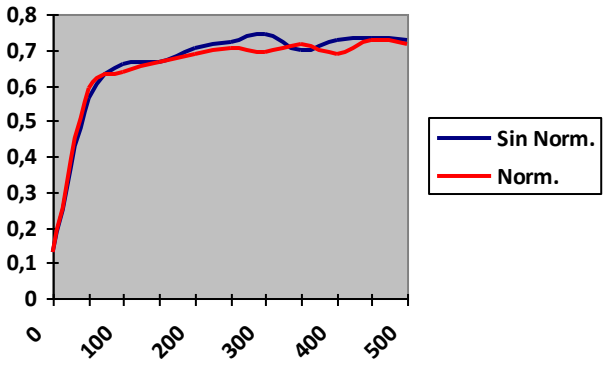


Figura 5.9. Comparativa de tasas de acierto

6. Conclusiones y trabajos futuros

En esta sección se va a resumir los puntos importantes del proyecto y se propondrán algunas mejoras y ampliaciones.

6.1. Conclusiones

A continuación, un resumen de lo que se ha visto:

- Se ha desarrollado una aplicación que buscará eventos donde ganar recompensas o lugares con descuentos.
    - Estará disponible en dispositivos Android y ios.
    - Para participar en el evento tendrás que tomar la fotografía de un logo preseleccionado.
- Para el reconocimiento de imagen se han usado redes neuronales convolucionales (CNN).
    - Se ha comparado entre una red personalizada propuesta por el tutor y una AlexNet.
- La red convolucional recibe como entrada imágenes.
    - En total hay ~6000 imágenes distribuidas en 10 clases de logos y un conjunto de imágenes aleatorias.
    - La información de estas imágenes está agrupada en un Dataset.

- Se ha dividido el Dataset en dos conjuntos de imágenes: Training y Test.
    - El conjunto Training está formado por el 85% de las imágenes y el resto por el conjunto Test.
    - El conjunto Training sirve para entrenar la red y mejorar su precisión.
    - El conjunto Test sirve para evaluar la eficacia de la predicción.

- A pesar del esfuerzo no se han conseguido grandes resultados.
    - Para mejorar los resultados son necesarias muchas más imágenes y clases.
    - También sería necesario invertir en Hardware para poder mejorar el rendimiento mediante la GPU.

6.2. Trabajos futuros

En relación con la parte tratada en este documento, lo más importante para el proyecto sería invertir en obtener más fotografías y mejorar el Hardware del servidor para entrenar modelos más rápido.

En algunos eventos como conciertos, hay compañías de teléfono que desactivan la cobertura. Otro punto importante sería dar la opción de descargar el modelo para poder realizar las predicciones desde el dispositivo del cliente en local sin necesidad de Internet.

También hay pendiente la resolución de algunos bugs antes de subir la aplicación a producción.

Respecto a nuevas funcionalidades, hay varias pensadas como nuevos eventos con un rol más educativo como buscar ciertas obras en un museo, mejoras de cara al cliente empresarial, que pueda modificar sus eventos con libertad, un sistema de puntos por participación canjeable por premios y un sinfín de ideas.

Lo positivo de este proyecto es que tiene un objetivo muy amplio, no necesariamente hay que quedarse con la organización de eventos, sino que puede ser ampliado a cualquier tipo de negocio. El límite de la aplicación lo pone la imaginación.

7. Bibliografía

1- Base de aplicación Ionic + Django: http://captaindanko.blogspot.com.es/2015/11/ionic-app-with-restful-backend-part-1.html

2- Página oficial Lasagne: http://lasagne.readthedocs.io/en/latest/index.html

3- Ejemplo con Lasagne: https://medien.informatik.tu-chemnitz.de/skahl/2016/10/20/image-classification-with-lasagne-part-1/

4- Información de Deep Learning: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/

5- Información de Deep Learning: https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/

6- Información de CNN: http://cs231n.github.io/convolutional-networks/

7- Información de CNN: https://www.coursera.org/learn/deteccion-objetos/lecture/hBpeK/l6-5-convolutional-neural-networks-cnns

8- Ejemplo de CNN: http://deeplearning.net/tutorial/lenet.html

9- Web de Scraping: https://www.google.es/

10- Web de Scraping: https://www.shutterstock.com/es/

11- Web de Scraping: https://www.bigstockphoto.com/es/

12- Web de Scraping: https://www.dreamstime.com/

13- Página oficial Keras: http://marcbs.github.io/multimodal_keras_wrapper

14- Repositorio Keras:
https://github.com/MarcBS/multimodal_keras_wrapper

15- Página oficial Bulk Image Downloader: http://bulkimagedownloader.com/

16- Algoritmo de AlexNet:
https://es.mathworks.com/help/vision/examples/image-category-classification-using-deep-learning.html