uc3m | Universidad Carlos III de Madrid

Grado Universitario de Ingeniería Informática

2018-2019

Trabajo Fin de Grado

Aplicación de *Deep Learning* a la Enseñanza de la Escritura Japonesa

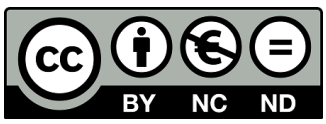
---

Gabriel García López

Tutores

Juan Manuel Alonso Weber

David Griol Barres



ABSTRACT

The teaching of language writing is a fundamental step that is needed in the learning process of a language. Despite this, big known applications specialized in the teaching of languages do not focus on this matter. There are a lot of small apps that focus uniquely on the teaching of writing through the usage of calligraphic templates that often correct or even delete the drawn outline. This final degree project proposes a new writing learning method: the usage of convolutional neural networks applied to character images recognition. The studied characters are the ones from the two syllabaries of the japanese language: *hiragana* and *katakana*. After a previous experimentation with other domains such as CIFAR10, Dogs vs. Cats and FERC, the ETL Character Database is used to conform two prediction models, one for each syllabary, obtaining a 99.23 % accuracy on the validation set for the *katakana* model and 99.61 % for the *hiragana* one. Afterwards, a web application for studying and writing japanese characters is developed integrating those models, reaching a new trusted didactic method for writing japanese without any templates and that accepts any calligraphic style.

Keywords: teaching, japanese, learning, calligraphy, writing, convolutional neural networks.

RESUMEN

La enseñanza de la escritura de idiomas es un paso fundamental en el proceso de aprendizaje de un idioma. Pese a esto, las aplicaciones más conocidas dedicadas a los idiomas no se focalizan en este ámbito. Hay muchas aplicaciones pequeñas que se especializan únicamente en la escritura de un idioma mediante el uso de plantillas caligráficas correctoras del trazado. En este trabajo de fin de grado se propone un nuevo método de enseñanza de la escritura: el uso de redes de neuronas convolucionales aplicadas al reconocimiento de imágenes de caracteres. Los caracteres estudiados son los respectivos de los silabarios japoneses *hiragana* y *katakana*. Tras una experimentación previa con otros dominios como CIFAR10, *Dogs vs. Cats* y FERC, con el conjunto de datos de caracteres ETL *Character Database* se crean dos modelos de predicción, uno por cada silabario, obteniendo un 99,23 % de acierto en el conjunto de validación en el modelo de *katakana* y un 99,61 % en el de *hiragana*. Posteriormente, se desarrolla una aplicación *web* dedicada al estudio y escritura de caracteres japoneses en donde se integran dichos modelos, consiguiendo así un nuevo método didáctico fiable para la escritura japonesa, libre de plantillas y que acepta todo tipo de caligrafías.

Palabras clave: enseñanza, japonés, aprendizaje, caligrafía, escritura, redes de neuronas convolucionales.

Dedicado a

Raúl y Mª Jesús, los dos factores más queridos del mundo que me dieron como producto,

a Marta, mi pequeña cultura y mi gran apoyo emocional,

a mis amigos de toda la vida, los que siempre llegan tarde pero siempre están ahí,

a mis tutores del proyecto, Juan Manuel y David,

a Zoraida Callejas, por su revisión desinteresada

y a mis amigos de la universidad, LaVendicionDevs.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

1. INTRODUCCIÓN

Para cualquier persona, el aprendizaje de un idioma es una tarea ardua y complicada que conlleva mucho sacrificio. En especial, para las personas occidentales, es muy complejo el aprendizaje de los idiomas orientales, y viceversa. Esto se debe a que los idiomas orientales, como el japonés o chino, y lenguas románicas y germánicas establecidas en toda Europa y América, tienen pocas similitudes entre sí.

Las lenguas románicas como son el español, el portugués o el italiano y las lenguas germánicas como el inglés o el alemán, tienen un listado organizado de las diferentes letras de las cuales se componen sus idiomas: **el abecedario**. Estas lenguas, a pesar de dividirse en varios subgrupos claramente marcados por algunas diferencias, parten del mismo abecedario y por lo tanto de las mismas letras. Siempre hay excepciones, como las lenguas germánicas septentrionales (danés o noruego) o la cedilla en el castellano antiguo, portugués o francés.

Esta misma composición de los diferentes abecedarios, con apenas variación, y, por lo tanto, del núcleo de los lenguajes hace que, para una persona occidental, aprender una lengua dentro de las lenguas románicas y germánicas sea más asequible que una lengua oriental.

El idioma japonés, que es el idioma oriental que aquí se trata, se fundamenta en unos símbolos que representan sílabas en vez de letras, es decir, el japonés es un idioma silábico, y por consecuencia, sus caracteres se ordenan en **silabarios**. Estos nada tienen que ver a los usados por las lenguas románicas o germánicas, tienen un trazado de líneas más complejo y son más numerosos que cualquier abecedario románico o germánico.

Por lo tanto, aquí radica la complicación de aprender este idioma o cualquier idioma oriental para una persona occidental (y viceversa): la diferenciación conceptual de la representación del lenguaje en sí mismo, como la adaptación a un nuevo sistema de expresión.

1.1. Motivación y objetivos

Debido a las tendencias actuales de tener todo al alcance de la mano, aprender un idioma con una aplicación o una página *web* se ha convertido en la gran apuesta.

Normalmente en estas aplicaciones y páginas *web* se usa el método de asociación, entre otros, para poder establecer relaciones automáticas entre conceptos. Es decir, se busca que el usuario aprenda el idioma a base de repetición de estructuras gramaticales, palabras y símbolos característicos sin tener un conocimiento práctico previo.

En el caso del japonés, esta regla de repetición de ejercicios no basta. Puede ser eficaz para familiarizarse con las estructuras gramaticales, formación de preguntas, afirmaciones, nuevas palabras o las terminaciones características del idioma, pero no para el aprendizaje de la escritura los caracteres. El hecho de que una persona sea capaz de diferenciar un carácter, o su pronunciación, de otro no basta para aprender el idioma: se necesita saber cómo se escriben los caracteres para entender realmente los cimientos del idioma. Aplicaciones como *LingoDeer*, reconocida mundialmente por la gran capacidad de aprendizaje que ofrecen a sus usuarios, no se centran en la importancia de la escritura y esto es un gran error a la hora de enseñar un idioma.

Esta falta de importancia que se le da a la escritura por parte de las aplicaciones y páginas *web* del momento es un problema grave, ya que conlleva a no tener una base sólida del idioma que se quiere aprender. En parte, este problema viene dado tanto por el ansia e impaciencia de la sociedad actual, en donde todo el mundo quiere todo al momento, como crítica Amparo Lasén en una entrevista para *El Independiente* [1], como por el hecho de cada persona escribe de una manera diferente. Es muy difícil crear una aplicación o *web* que se adapte y generalice la escritura de un idioma, y que al mismo tiempo cautive al usuario a seguir aprendiéndolo.

Es por esto por lo que muchas aplicaciones dedicadas a enseñar a escribir japonés optan por incluir una plantilla en la que el usuario es obligado a dibujar el carácter exactamente como establece esa plantilla. Con esto evitan tener que lidiar con el problema de la generalización. El único punto débil de este enfoque es que, si el usuario dibuja correctamente el carácter, pero no coincide con la plantilla, el carácter estará mal dibujado cuando en realidad no lo está como se ve en la Figura 1.1.

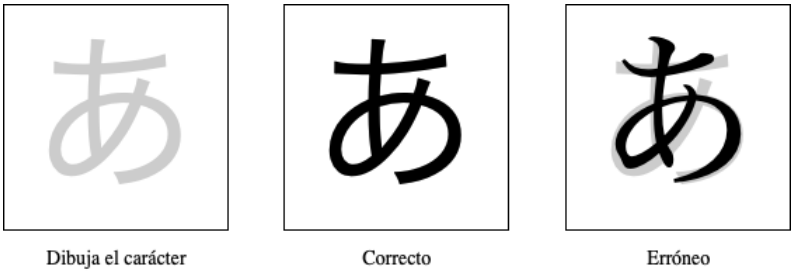


Figura 1.1. Funcionamiento de plantillas de aprendizaje de idiomas en aplicaciones

Esta aplicación *web* va a tratar de eliminar esa plantilla, y dejar al usuario que escriba los caracteres japoneses como él o ella lo haría en papel. Es decir, en la *web* se va a intentar emular la libre escritura como en las clásicas hojas de aprendizaje japonés, en la que no hay plantilla alguna, solo un recuadro en donde se debe escribir el carácter. Estas hojas caligráficas se utilizan en las escuelas niponas y en las de idiomas para aprender a escribir el japonés básico.

Otro problema que existe es que las aplicaciones no tienen en cuenta la caligrafía a la hora de enseñar a escribir con plantillas: siempre se enseñan los caracteres en un único tipo de caligrafía sin tener en cuenta que no hay dos personas que tengan la misma (Figura 1.1).

Para permitir y reflejar esta diversidad caligráfica a la hora de escribir y contando con la ayuda de la inteligencia artificial, la *web* intenta cubrir todos los estilos de escritura para hacer más flexible el aprendizaje del idioma, cosa que muy pocas *webs* o aplicaciones de enseñanza hacen hoy en día. Para entender mejor el propósito, si volvemos a la Figura 1.1, pero desde la perspectiva de la aplicación *web* aquí presentada, ambos caracteres escritos serán reconocidos como el correcto.

Por lo tanto, lo que se quiere conseguir con este trabajo es la creación de una *web* para aprender a escribir japonés mediante la libre escritura de los diferentes caracteres básicos del idioma, para aprender a diferenciar y asociar las distintas pronunciaciones de todos los caracteres de manera autodidacta. Además se intenta que también sirva de base o complemento para aprender el resto del idioma.

1.2. Estructura del documento

El documento se divide en varias secciones y esta tiene el único fin de informar de manera general de lo que se ha trabajado en este estudio.

- **Introducción**: trata la idea general que se va a llevar a cabo en el trabajo.
- **Estado del arte**: se realizará un estudio analítico de las herramientas y métodos de *deep learning*, así como de métodos docentes. Este estudio va a facilitar la elección de las herramientas y métodos principales con los que se sustenta la aplicación *web*. Se van a estudiar los métodos de enseñanza clásicos y modernos, además de los diferentes métodos de enfoque con respecto al reconocimiento de imágenes.

- **Experimentación**: aquí se explicarán las herramientas y metodología utilizadas para la experimentación junto con la exposición de las diferentes pruebas y evaluaciones previas al tratamiento del conjunto de imágenes de caracteres japonés. Además se comentarán las propias pruebas de dicho conjunto y por último, se harán unas conclusiones de lo aprendido.

- **Aplicación *web***: aquí se explicará el objetivo principal de la *web* y las herramientas y *framework* utilizados para la creación de esta. Además, se hará un recorrido completo por la *web* a modo de manual de usuario para visualizar el aspecto y funcionalidad que la aplicación ofrece. También se incluirán datos extraídos de un cuestionario que se brinda a todos los usuarios que prueben la aplicación para que en un futuro cercano esta mejore en función de lo que los usuarios requieran.

- **Maro regulador**: se comentarán los aspectos legales de la aplicación con respecto al tratamiento de datos de los usuarios y las licencias de las distintas herramientas utilizadas.

- **Entorno socioeconómico**: se especificará la planificación que se ha llevado a cabo, el presupuesto del proyecto y el impacto que tiene y tendrá este trabajo en el ámbito socioeconómico presente y futuro.

- **Conclusiones y trabajos futuros**: por último, aquí se darán las conclusiones generales extraídas del trabajo junto con los posibles trabajos futuros aplicables a lo realizado.

2. ESTADO DEL ARTE

En esta sección se hará un repaso a los métodos utilizados en las diferentes áreas en las que se basa este proyecto. Se hará hincapié en la historia de las redes de neuronas y en especial a las redes destinadas al reconocimiento de caracteres, explicando su funcionamiento básico. También se explorarán y explicarán otros métodos aplicados para una posterior comparación entre ellos. Seguidamente, se realizará un breve estudio diferenciando los distintos métodos de enseñanza de caligrafía que han existido y que existen para los idiomas.

2.1. Reconocimiento de caracteres

2.1.1. Breve historia y fundamentos de las redes de neuronas

Una red de neuronas es un modelo matemático que simula la estructura neuronal del cerebro: dada una información de entrada, es capaz de producir una salida que contiene la aproximación de una solución a un problema. Las redes de neuronas tienen un origen teórico establecido en el artículo de 1943 escrito por McCullogh y Pitts "A Logical Calculus Of The Ideas Immanent In Nervous Activity" [2] y un origen biológico, de ahí su nombre, que se basa en las teorías neuronales presentadas por Hebb en su libro *The Organization of Behaviour: A Neuropsychological Theory* de 1949 [3].

Por un lado, McCullogh y Pitts establecen la base teórica de lo que se conoce como **neurona artificial**: una estructura computacional con varias entradas por las que se pasa información y que con ella produce una salida. Esta definición se va a utilizar a lo largo de la historia de las redes de neuronas, y es por eso por lo que McCullogh y Pitts son considerados los padres de las redes de neuronas artificiales.

Y, por otro lado, Hebb estudia las capacidades psicofisiológicas humanas y determina que el aprendizaje del ser humano se localiza en las sinapsis neuronales y que la información es representada en nuestro cerebro mediante un conjunto de neuronas activas e inactivas.

Con estos fundamentos se construyeron los primeros modelos de redes como el **perceptrón simple** en 1958 [4] y el **adaline** en 1960 [5]. Ambos modelos de red, capaces de aprender de manera autónoma, se basan en una red muy simple con una sola ca-

pa oculta y una sola salida. Estas creaciones inician la primera época dorada de las redes de neuronas en la mitad del siglo XX. Pero en 1969, Minsky y Papert en su libro *Perceptrons: An Introduction to Computational Geometry* explican y demuestran que estas redes de neuronas no son capaces de aprender a resolver problemas no lineales [6], es decir, que estos algoritmos tienen una capacidad limitada. Esta poca capacidad resolutiva hace que las redes de neuronas caigan en el olvido en los siguientes 15 años.

Sin embargo, es a mitad de los años 80 cuando resurgen las redes de neuronas con un nuevo paradigma de aprendizaje que mejora su capacidad resolutiva: **el algoritmo de retropropagación**. El algoritmo de aprendizaje fue descubierto prácticamente al mismo tiempo por varios investigadores y pioneros de las redes de neuronas. Se les atribuye el descubrimiento a: McClelland, Rumelhart y Hinton [7], Le Cun [8] y Parker.

Este algoritmo permite que las redes puedan aprender a resolver cualquier tipo de problema, tanto lineal como no lineal (**perceptrón multicapa**). Además, estas dan una salida final más ajustada y acertada que las redes y algoritmos anteriores. Es gracias a este descubrimiento que las redes de neuronas proliferan hasta finales de los 90, cuando se crean las máquinas de soporte vectorial (SVM).

Aún así, el verdadero auge de las redes de neuronas se inicia entre 2006 y 2012 con la introducción del ***deep learning*** como método aún más elaborado para entrenar y crear redes de neuronas. Bien es cierto que las redes obtenidas a partir del *deep learning* están más enfocadas al reconocimiento visual de objetos y formas, al reconocimiento de texto y al reconocimiento de caracteres [9].

2.1.2. Reconocimiento de caracteres con redes de neuronas

Uno de los primeros intentos de reconocer patrones visuales mediante la técnica de las redes de neuronas se le atribuye al estudio realizado por el japonés Kunihiko Fukushima en 1982 titulado "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" [10]. Esta técnica se basa en los trabajos presentados en 1959 por Wiesel y Hubel sobre el estudio de los campos receptivos del córtex cerebral de un gato [11]. Wiesel y Hubel descubren que existen varios tipos de células biológicas encargadas de la visión en el córtex ocular. Estas células están conectadas entre sí a modo de cascada: transmiten la información recogida de unas a otras dejando cada vez más aislados ciertos

patrones visuales para que el cerebro pueda llegar a reconocer lo que realmente se está observando. Fukushima integra esta abstracción de las células y su modo de trasmisión de información en una extensión de una red de neuronas artificial: **el neocognitrón**.

Mediante la repetición de patrones visuales, sea por ejemplo una secuencia de imágenes representando a la letra "A" de diferentes maneras, el neocognitrón es capaz de aprender por sí solo los patrones de la imagen que hacen que en cualquier imagen se reconozca la letra "A". Para hacer esto posible, Fukushima hace uso del ***feature extraction***. El *feature extraction* busca características más fáciles de identificar, pero igual de relevantes, a partir de un patrón, para que el proceso de reconocimiento sea más eficiente.

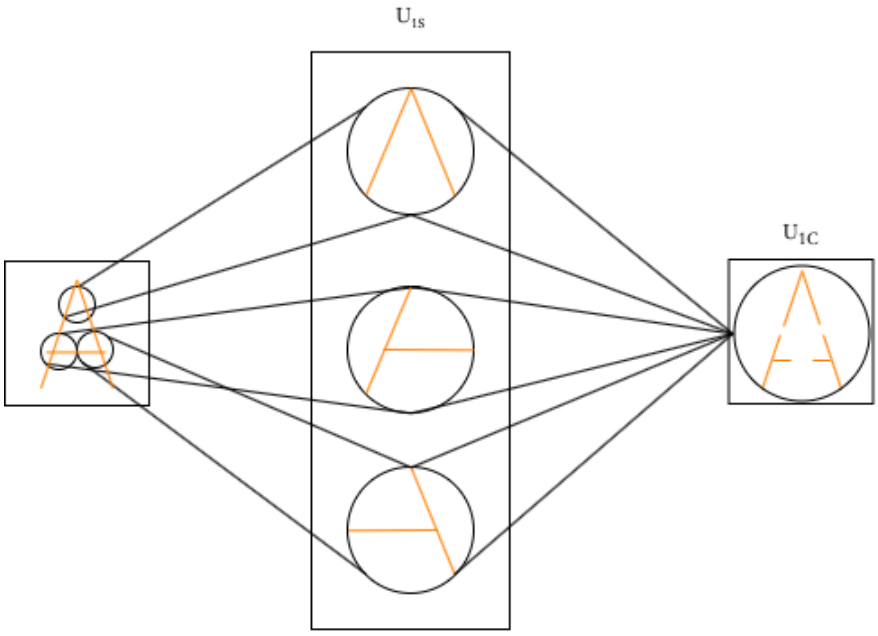


Figura 2.1. *Feature extraction* en el neocognitrón

En la Figura 2.1 se intenta reproducir el funcionamiento básico del neocognitrón de manera simple. La red emplea 3 neuronas en la capa U_{1S} que se especializan en los patrones específicos de la letra "A", dejando a disposición de la siguiente capa U_{1C} la letra "A" más descompuesta pero todavía reconocible. Esto hace posible que la red sea capaz de identificar la letra "A" con pequeñas variaciones en sí misma como se puede observar en la Figura 2.2.

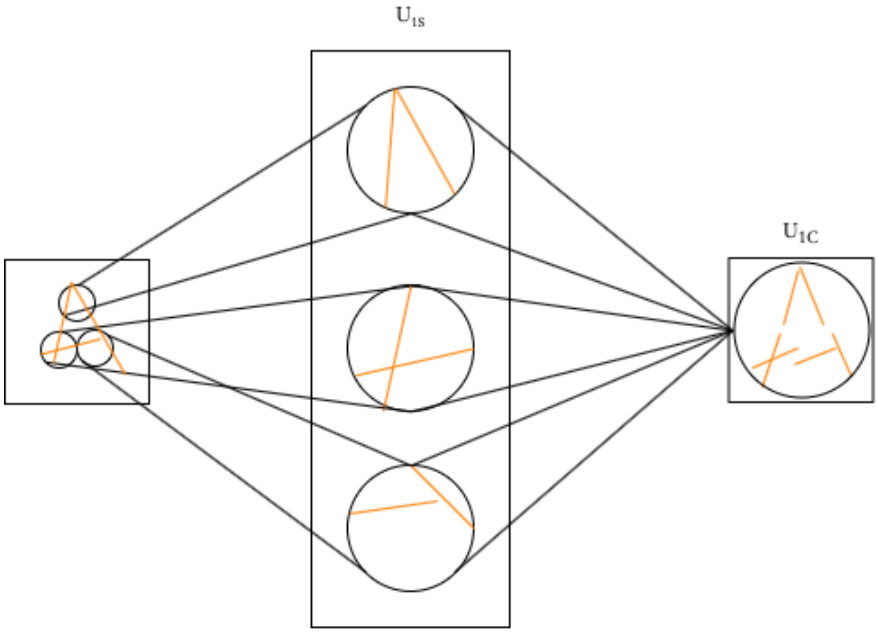


Figura 2.2. *Feature extraction* con variación de entrada en el neocognitrón

De esta manera, se consigue entrenar la red neuronal con varias instancias de un mismo patrón de información pero con transformaciones aplicadas. Es decir, el *feature extraction* hace posible que un algoritmo sea capaz de reconocer que los dos caracteres de la Figura 1.1 son, en efecto, el mismo. Esta capacidad analítica tan potente que permite a Fukushima generalizar las salidas de una red neuronal, sirve de base para la creación de las **redes de neuronas convolucionales**.

2.1.3. Redes de neuronas convolucionales

Las redes de neuronas convolucionales o CNN fueron creadas por el recién galardonado con el premio Alan Turing 2018, Yann Le Cun en 1989 en su artículo "Backpropagation Applied to Handwritten Zip Code Recognition" [12]. Estas redes aplican el algoritmo de retropropagación combinando el modelo propuesto por Fukushima al reconocimiento de formas u objetos en imágenes.

La red se divide por pares de capas convolucionales-*pooling*, para más tarde, conectarse a un perceptrón multicapa para poder clasificar las imágenes en base a los patrones extraídos. El conjunto de pares de capas pares se llama **base convolucional**.

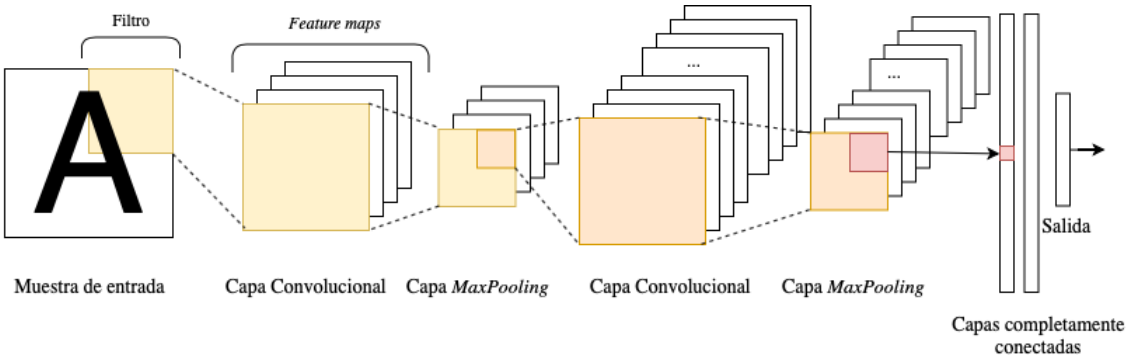


Figura 2.3. Estructura básica de una red de neuronas convolucional

En las capas convolucionales se realiza la **operación de convolución**. Esta operación consiste en la extracción de varios ***feature maps*** a partir de la imagen de entrada o **campo receptivo**. Estos mapas son pequeñas partes de la imagen en los que se va a extraer un patrón visual mediante el *feature extraction*. Cada mapa representa ciertos píxeles de la entrada y cada una de las posiciones del mapa contiene un número que comprende entre el 0 y el 255, representando el nivel de intensidad de cada píxel extraído.

Durante la operación de convolución, se desplaza una ventana de dimensión arbitraria (**filtro**) por toda la muestra de entrada para obtener nuevos patrones con los que trabajar en la red. Por ejemplo, si se aplica un filtro de 3x3 píxeles a una entrada de 4x4 se obtienen 4 mapas.

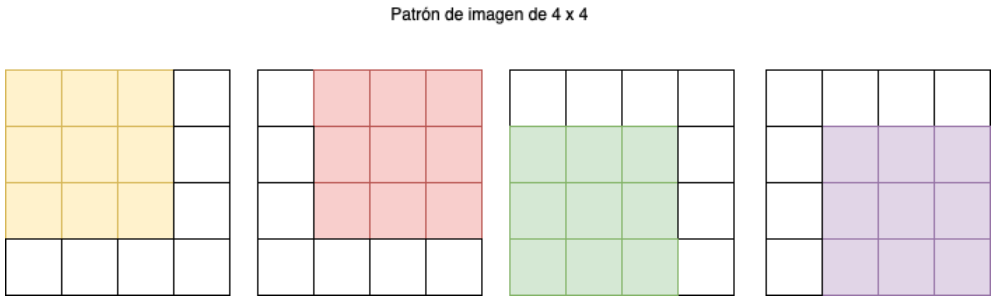


Figura 2.4. Extracción de mapas característicos de una imagen de 4x4

Siguiendo con el ejemplo expuesto en la Figura 2.4, con estos 4 mapas se procede a la siguiente operación: ***max pooling***. Esta simple operación sirve para hacer una criba de parámetros en los mapas para que la red no se sature. Consiste en la extracción del mayor valor píxel de cada *feature map* aplicando una ventana para reducir los parámetros de los mapas. Esta ventana de *pooling*, también de dimensiones arbitrarias, se va trasladando por el mapa al igual que los filtros. Se puede ver la operación

completa en un mapa de 4x4 con ventana de *pooling* de 2x2 en la siguiente figura.

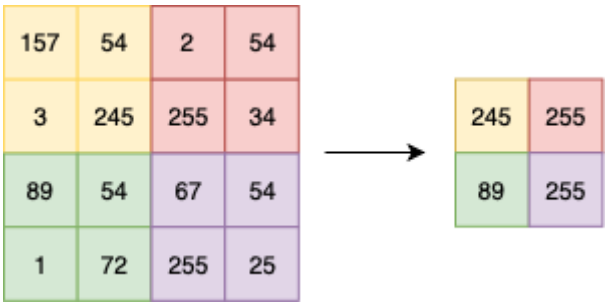


Figura 2.5. Operación de *max pooling* con ventana de 2x2

Después de la operación de *pooling*, el *feature map* proporciona la información con la que trabajará la siguiente capa convolucional, aplicando la misma operación anteriormente descrita. Gracias a estas técnicas, se extrae de cada patrón la información más relevante y fundamental de la imagen de entrada para generalizar los patrones y, así, conseguir que la red tenga una capacidad de reconocer cualquier patrón.

Finalmente, cuando se hayan aplicado las suficientes operaciones en la base convolucional, toda la información extraída se pasará a una nueva estructura como puede ser un perceptrón multicapa o una máquina de soporte vectorial. Este paso es vital ya que la base convolucional solo es capaz de aprender información espacial. Estas estructuras o modelos son necesarios para poder ordenar y clasificar los datos no estructurados extraídos de las imágenes. Una vez las muestras queden estructuradas y clasificadas, la red nos dará el porcentaje de acierto de la clasificación de todo el conjunto de datos.

Las CNN están vinculadas estrechamente con el tratamiento de imágenes y, debido a ello, se aplican en diversos campos como en el **reconocimiento de textos y caracteres** [13], seguridad [14] o medicina [15].

2.1.4. Otros métodos

Por supuesto, las redes de neuronas convolucionales no son el único método que se ha aplicado en los últimos tiempos al reconocimiento de caracteres escritos. El **perceptrón multicapa** (MLP), las **máquinas de soporte vectorial** (SVM) o el **algoritmo de *k-nearest neighbours*** (kNN) han sido otras opciones que se han considerado para el estudio de este campo. A continuación, se explican los diferentes algoritmos alternativos.

2.1.4.1 Perceptrón Multicapa

El perceptrón multicapa o MLP es una red de neuronas que utiliza el algoritmo de retropropagación para su aprendizaje. El MLP tiene una estructura basada en capas ocultas (generalmente más de una) con neuronas artificiales interconectadas entre sí como se aprecia en la Figura 2.6.

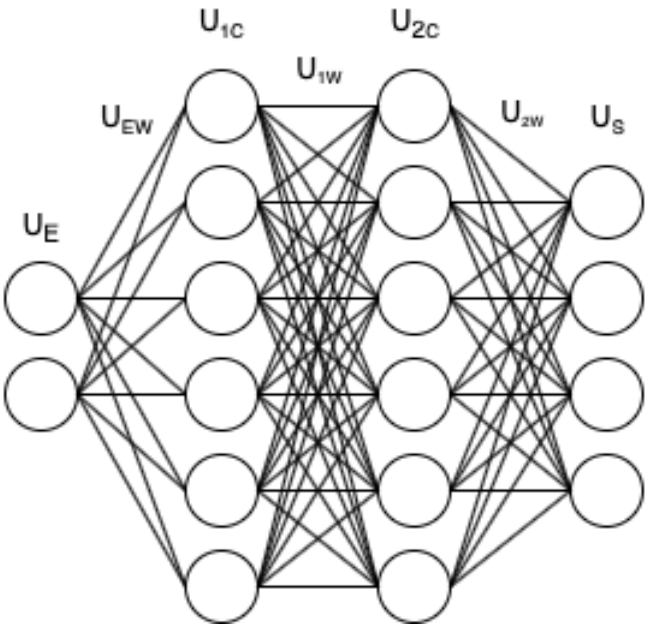


Figura 2.6. Estructura básica de un MLP

Donde U_E es la capa de entrada, por donde los datos entran en la red, U_{1C} y U_{2C} son las capas ocultas de la red, U_S es la capa de salida, en donde se contiene el resultado de la red, y los pesos generales U_W, en donde cada neurona de la red tiene varios asociados U_{NW_i}.

Cada capa tiene un número diferente de neuronas: el de U_E viene determinado por el número de atributos de cada patrón de entrada, el de U_S se determina mediante el número de clases si es un problema de clasificación, o en caso de un problema de regresión basta con una sola neurona de salida, y, por último, el número de neuronas de las capas ocultas es totalmente arbitrario, pero generalmente estas redes trabajan mejor con un número alto de neuronas en cada capa.

Para propagar un patrón por la red, las neuronas artificiales son las encargadas de mover la información de la capa de entrada a la capa de salida. La estructura básica de estas neuronas artificiales se observa en la Figura 2.7.

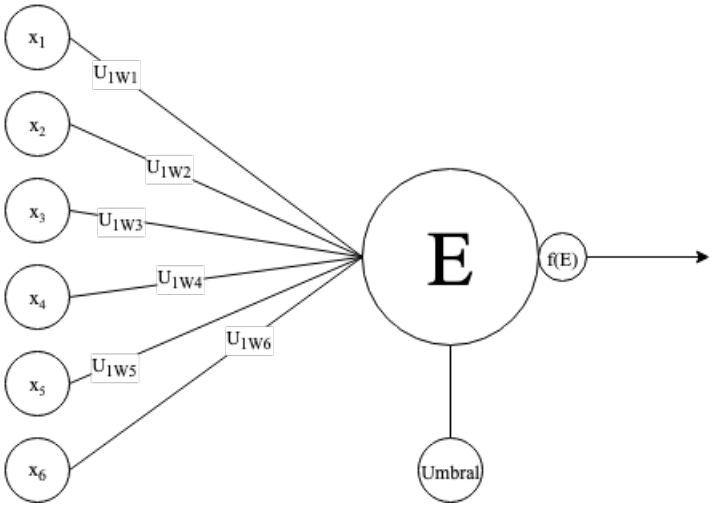


Figura 2.7. Estructura de una neurona artificial[1]

En cada neurona se calcula una ponderación E entre los pesos conectados a la neurona, U_{1W_N} por su entrada, X_N. Una vez calculada, se aplica la **función de activación** $f(E)$, condicionada por el **umbral**. Este es un valor constante independiente que permite definir el valor mínimo que $f(E)$ debe tomar para considerar si la neurona se activa o no para pasar la información a las siguientes capas.

Para entrenar, la red intenta minimizar la función de error de salida utilizando el **descenso de gradiente**. Este algoritmo va a buscar el mínimo de una función objetivo dada, en este caso, la función de error. De esta manera, se consigue un valor mínimo de la función de error de la red, definida como la salida deseada y la salida producida, clasificando patrones con menos margen de error y dando mejores resultados.

Para el aprendizaje de la red se hace uso del **algoritmo de retropropagación o la regla delta generalizada**, calculando unos valores δ por cada neurona en función de los valores de la neurona y pesos anteriores, y propagando hacia atrás dicho valor para actualizar los pesos y umbral de las neuronas. El proceso general de entrenamiento de un perceptrón multicapa se define en cuatro pasos [16]:

1. El primer paso es la inicialización aleatoria de los pesos, normalmente con valores pequeños.

2. Se presenta un patrón de entrada a propagar por la red, siempre hacia delante.

[1] Se ha tomado como referencia una neurona de la capa oculta 2 de la estructura establecida en la Figura 2.6.

3. Se calcula el error entre la salida de la red dada por el patrón introducido y por la salida real.

4. Se aplica el algoritmo de retropropagación actualizando y optimizando los pesos y umbrales de la red.

5. El proceso descrito a partir del paso 2 se repite para todos los patrones de entrada hasta que la red alcance unos pesos y umbrales óptimos y con una función de error mínima.

Estas redes de neuronas se consideran unos aproximadores universales debido a su gran capacidad de generalización y a su versatilidad a la hora de resolver problemas, tanto lineales como no lineales, siendo este algoritmo aplicable a muchos campos: **reconocimiento de caracteres** [17], medicina [18] o comercio electrónico [19].

2.1.4.2 Máquinas de Soporte Vectorial

La máquina de soporte vectorial o SVM es un algoritmo que se aplica a problemas de clasificación o regresión que fue inventado por Corinna Cortes y Vladimir N. Vapnik en 1993 [20]. Teniendo un plano de n-dimensiones con m puntos pertenecientes a diferentes clases, la SVM establece un **hiperplano** que mejor separa todos esos puntos clasificándolos por su clase.

Este hiperplano viene determinado por los vectores de soporte. Estos vectores son las coordenadas de los puntos y ayudan a decidir el hiperplano en situaciones comprometidas como la de la Figura 2.8.

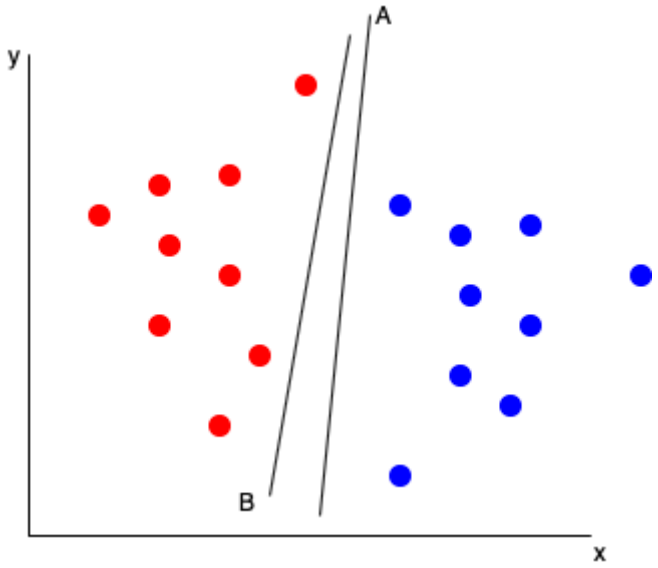


Figura 2.8. Frontera de decisión con dos hiperplanos en una SVM

La frontera de decisión entre los puntos rojos y los puntos azules está correctamente delimitada por A y por B. Ahora, la que más robustez ofrece al clasificador es el hiperplano A ya que los márgenes entre los vectores de soporte "límite" de ambas clases y la frontera son los máximos. Con esto se consigue que el error en la clasificación sea el mínimo posible. Esto ofrece a la SVM una mayor eficacia a la hora de delimitar las clases de un determinado problema [21].

Las máquinas de soporte vectorial supusieron una gran alternativa a las redes de neuronas artificiales como el perceptrón multicapa a finales de los 90 gracias a su gran capacidad y acierto en la definición de hiperplanos delimitadores. Hoy en día se siguen utilizando. Sus aplicaciones son varias: se aplican al **reconocimiento de caracteres**, objetos y formas [22], finanzas [23] o biología [24].

2.1.4.3 *K-Nearest Neighbours*

Al igual que las SVM, el algoritmo de kNN se aplica a problemas de clasificación o regresión. Dado un plano n-dimensional con m puntos ya clasificados en c clases, kNN va a clasificar un nuevo punto entre esas c clases en base a sus puntos vecinos más próximos.

El número de vecinos que determinan si un punto a clasificar pertenece a una clase u otra viene dado por el **factor** k principalmente. Este factor es un número totalmente arbitrario y determinante para la eficacia del algoritmo que se puede determinar de manera empírica.

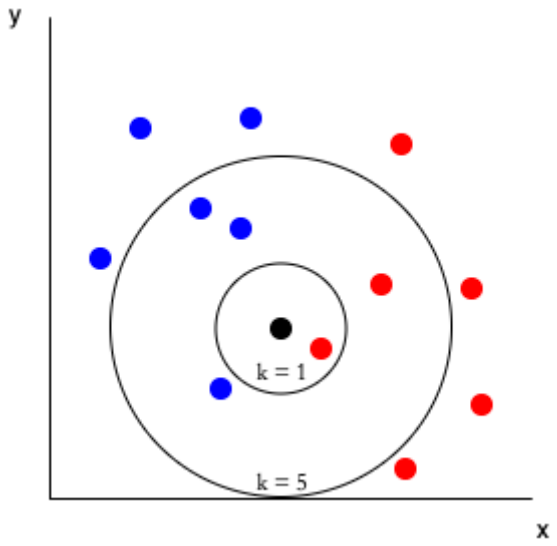


Figura 2.9. Variaciones del factor k en kNN

Como se observa en la figura, a medida que k aumenta, las clasificaciones del punto negro en la clase rojo o azul varían. Con $k = 1$, el vecino más próximo es el punto rojo que queda dentro de la circunferencia establecida, con centro en el punto negro, por lo tanto, se clasificaría como rojo, ya que es la clase dominante. En cambio, con $k = 5$, el radio de la circunferencia aumenta y vemos como sus vecinos más próximos son 3 azules y 2 rojos, clasificándose como azul.

Este algoritmo, simple y con grandes resultados, es recurrente en análisis de datos y aprendizaje automático, siendo utilizado en la docencia como algoritmo introductorio al mundo de la inteligencia artificial. Sus aplicaciones son varias: se aplican al **reconocimiento de caracteres escritos** [25], economía [26] o biología [27].

2.1.5. Comparativa de métodos

Los tres métodos explicados aquí son aplicados al reconocimiento óptico de caracteres escritos (OCR) entre otros tantos, como se ha visto. Pero algunos de los métodos descritos tienen unos inconvenientes que no se deben pasar por alto.

El mayor problema de las SVM es la memoria disponible durante la ejecución del algoritmo con grandes cantidades de datos. Las SVM funcionan bien con una cantidad de datos moderadamente pequeña ya que no saturan el espacio de puntos a construir para determinar el hiperplano. Cuando se quiere trabajar con muchos datos o imágenes, las SVM son ineficientes tanto en tiempo como en consumición de

recursos y memoria. Esto se traduce a que la matriz que sirve para la representación de los puntos en el espacio escala de manera desfavorable a medida que estos aumentan. Este problema se puede solventar mediante varias alternativas: optimizando la función *core* del algoritmo o mediante una criba de datos.

K-nearest neighbours pese a ser un algoritmo sencillo y que nos brinda resultados moderadamente buenos, tiene varios inconvenientes y estos tienen que ver con la alta arbitrariedad y escalabilidad del algoritmo.

Por un lado, la determinación del parámetro k es puramente empírica, no hay manera de saber si un valor dado de k es el óptimo o no, como se ha podido observar en la Figura 2.9 y esto causa incertidumbre en el algoritmo. El hecho de que el parámetro k disminuya, conlleva a una cantidad menor de muestras a analizar para decidir la clase de la instancia analizada. Si el conjunto de datos que se utiliza tiene mucho ruido (muestras a evitar por falta de información relevante), con una k pequeña el algoritmo tendrá en cuenta con mucha probabilidad una de esas muestras no válidas, dando lugar a clasificaciones erróneas. Por lo tanto, esto supone otro problema más: el algoritmo puede ser influido demasiado por el ruido en los datos.

Y, por otro lado, al igual que con las SVM, kNN resulta ineficiente cuando se trabaja con grandes cantidades de datos debido al gran cómputo que se debe realizar para calcular los puntos en el plano n-dimensional. Independientemente, tanto el problema del ruido como el de la ineficiencia tienen una solución directa: la realización de una criba de datos no relevantes o datos duplicados. Para el problema de la determinación de k no hay una solución genérica, pero se han estudiado varias opciones como utilizar validación cruzada [28] o utilizar $k = n^{1/2}$, donde n es el número de instancias o puntos en el plano. Pero una vez más, ninguno de los métodos es el óptimo, solo se podrá verificar su validez mediante ensayo y error.

Por su parte, las redes de neuronas convolucionales y el perceptrón multicapa tienen los mismos problemas principales: consumen muchos recursos y tiempo para realizar los cálculos correspondientes durante el entrenamiento y necesitan una gran cantidad de datos para poder alcanzar buenos resultados.

El coste computacional viene dado por el número de neuronas utilizadas en cada capa oculta, el número de capas ocultas o la cantidad de datos que se utilicen. Normalmente, para este tipo de algoritmos, se suelen utilizar numerosas capas ocultas

con muchas neuronas para poder producir resultados buenos, por lo que la carga computacional es inevitable.

La necesidad de obtener un conjunto de datos relativamente grande puede ser una tarea complicada, ya que no hay una regla genérica que estime cuántos datos son los suficientes para que un MLP o una CNN sea óptima, pero normalmente los conjuntos de datos que existen hoy en día están adecuados a esta necesidad. La tarea complicada es crear un conjunto de datos desde cero ya que requiere mucho tiempo.

La utilización de la unidad de procesamiento gráfico (GPU) como método de cálculo alternativo a la unidad de procesamiento central (CPU) supone una mejora muy notable en el tiempo de entrenamiento de las CNN y los MLP debido a la gran cantidad de operaciones en paralelo que se pueden realizar en las GPU. Este reemplazo de centro de operaciones es la mejor solución al problema de consumición de tiempo y recursos, ya que se tarda mucho menos tiempo en entrenar la red y la CPU queda libre para el resto de las actividades del ordenador.

Pero, a pesar de que el MLP y las CNN tienen los mismos inconvenientes, la principal desventaja del perceptrón multicapa frente a las redes de neuronas convolucionales es el hecho de que el perceptrón multicapa aprende patrones globales, mientras que las redes de neuronas convolucionales se centran en patrones locales (gracias al *feature extraction*). Esto conlleva dos ventajas con respecto al reconocimiento de imágenes:

- Los patrones que la CNN aprende son invariantes: la red puede aprender cierto patrón de una imagen y ser capaz de reconocerlo en cualquier posición y momento del entrenamiento.
- Las CNN son únicamente capaces de aprender patrones espaciales. Esto quiere decir que varias capas de la red pueden aprender diferentes patrones de una imagen y esos patrones se recompondrán para que otra capa de la red aprenda de ello, extrayendo más información de los patrones.

En definitiva, se observa que los cuatro métodos son susceptibles de ser ineficientes en tiempo y recursos ante una gran cantidad de datos. Pero, la gran diferencia es que tanto las SVM y kNN trabajan bien con conjuntos de datos pequeños consiguiendo resultados bastante competitivos con poca información. La desventaja de esto es que si se quiere trabajar con un conjunto de datos grande, la criba que se debe realizar para que los algoritmos sean eficientes supone una pérdida de información para los modelos. Mientras que las CNN y MLP necesitan trabajar con muchos datos para

conseguir unos resultados inigualables, ya que hacen uso de ellos para conseguir extraer mucha más información que los algoritmos anteriores.

Tras haber expuesto las distintas ventajas y desventajas de los diferentes algoritmos, se toman como referencia dos de los conjuntos referentes de datos del aprendizaje automático: el **MNIST** [29], que contiene 70.000 imágenes de caracteres numéricos del 0 al 9; y **CIFAR10** [30], conglomerado de otras 60.000 imágenes de objetos y animales corrientes como aviones, camiones, coches o caballos. Con estos dos conjuntos de datos, se van a comparar los distintos algoritmos estudiados para tener una idea más general del poder analítico de cada uno.

Algoritmo	Media de % de Error	Referencia
SVM	1,06 %	[31]
kNN	3,64 %	
MLP	2,22 %	
CNN	0,71 %	

Tabla 2.1. Comparación de tasa de error entre algoritmos con MNIST[2]

Algoritmo	% de Acierto	Referencia
SVM	55,22 %	[32]
kNN	41,78 %	[33]
MLP	78 %	[34]
CNN	95,59 %	[35]

Tabla 2.2. Comparación de tasa de acierto entre algoritmos con CIFAR10[3]

Se observa que las CNN es el algoritmo que obtiene el mejor resultado en MNIST con casi un 0,3 % de diferencia en la tasa de error y en CIFAR10 obtiene casi un 96 % de acierto, sin competencia alguna.

Se puede apreciar que hay una gran diferencia de resultados entre las clasificaciones de CIFAR10 y de MNIST. Esta diferencia puede darse por la dificultad de clasificación del CIFAR10, ya que este conjunto exige a los algoritmos fijarse en detalles muy complejos de las imágenes para decidir si, por ejemplo, una imagen es un ciervo o un caballo. Además, estas imágenes albergan mucho ruido debido a que los objetos a identificar no están aislados en la imagen como se observa en la Figura

[2] Se han tenido en cuenta para la media los algoritmos sin ningún tipo de procesamiento de imágenes previo.

[3] Se han seleccionado los mejores resultados encontrados para cada algoritmo.

2.10, añadiendo complejidad a la extracción de información. Esta dificultad no existe en el conjunto de MNIST ya que los números del 0 al 9 son claramente diferenciables.

(a) Imagen de CIFAR10

(b) Imagen de MNIST

Figura 2.10. Ejemplos de muestras de CIFAR10 y MNIST

Esta dificultad del ruido en imágenes (como la imagen 2.10a) y la extracción de información compleja en imágenes no afectan a las CNN mientras que los demás algoritmos se ven claramente perjudicados, como se observa en la Tabla 2.2. Por esto, gracias al poder de extracción de patrones específicos locales, a su gran capacidad de generalización en todo tipo de imágenes y a que son hoy en día el algoritmo más potente en relación con el estudio de imágenes, las redes de neuronas convolucionales es la mejor opción entre los algoritmos presentados para su aplicación al reconocimiento de caracteres japoneses escritos que se va a estudiar en este trabajo.

2.2. Caligrafía: métodos de enseñanza

Según la Real Academia Española, la caligrafía es el "conjunto de rasgos que caracterizan la escritura de una persona". El conjunto puede ser muy variado y extenso, ya que cada persona escribe de manera diferente. En esta sección se comentarán y compararán los distintos métodos que han existido y existen hoy en día para la enseñanza de la escritura.

2.2.1. Métodos clásicos

La caligrafía es un método clave en la enseñanza de un idioma. Este enseña a las personas a reconocer, escribir y diferenciar los caracteres de un idioma mediante la repetición de estos. La caligrafía se suele enseñar en una época temprana de la vida de una persona ya que es en ese momento cuando se es más receptivo al aprendizaje, según Alan Eyerly en su estudio para la universidad de California [36]. Cuando el niño o niña es capaz de diferenciar entre dibujo y escritura es cuando se le

enseña a escribir con cuadernos cuadriculados y especiales para aprender a escribir cada carácter atendiendo al trazado.

El proceso de aprendizaje se basa en la repetición de los caracteres hasta que los músculos y el cerebro se acostumbran a los movimientos del trazado. Los caracteres se enseñan por separado para que el alumno aprenda a diferenciarlos y a escribirlos individualmente para poder, después, ser capaz de concatenarlos para formar palabras.

El hecho de que en estos cuadernos se enseñe a escribir de una forma predeterminada durante su infancia, hace que el alumno generalice el concepto del símbolo representado y personalice su forma, siempre manteniendo el sentido de este. Por esta razón, un adulto o un adolescente, son capaces de reconocer las mismas letras en una palabra mientras que escriben las mismas con connotaciones diferentes.

Gracias a la enseñanza de los caracteres, un adulto ya sabe cuál es el proceso de aprendizaje de la escritura de su idioma natal, y eso le ayuda a extrapolar la misma metodología de aprendizaje a otros idiomas como el japonés, en este caso. El aprendizaje de este idioma se basa también en la repetición de los caracteres, con la peculiaridad de que se muestran los diferentes trazos a realizar por carácter (véanse las flechas azules de las Figuras 2.11 y 2.12). En el español, cada carácter se escribe de manera continua, en la mayoría de los casos, sin interrumpir el trazo del carácter, pero en el japonés, cada uno tiene un número de trazos determinado.

El japonés es un idioma de origen chino con una ordenación silábica. El idioma se divide en dos silabarios de 46 caracteres diferentes que no tienen valor conceptual (al igual que las letras en el castellano), sino fonético.

- *Hiragana*: este silabario es el utilizado para formar palabras en el japonés, aunque también sirven individualmente como partículas para denotar al sujeto u objeto de una frase.

Figura 2.11. Silabario *hiragana*

- *Katakana*: este silabario, sin embargo, sirve únicamente para trasladar expresiones o palabras anglosajonas o extranjeras al japonés, como las adaptaciones de *parking* o *ticket* al español.

Figura 2.12. Silabario *katakana*

Los caracteres ゐ,ヰ (WI) y ゑ,ヱ (WE) están obsoletos en ambos silabarios, ya que en Japón se utilizan en su lugar los caracteres い,イ (I) y え,エ (E) respectivamente por su parecido fonético. El aprendizaje se realiza con cuadernos especializados

cuadriculados con las siguientes plantillas para aprender los caracteres (Figura 2.13 [37]).

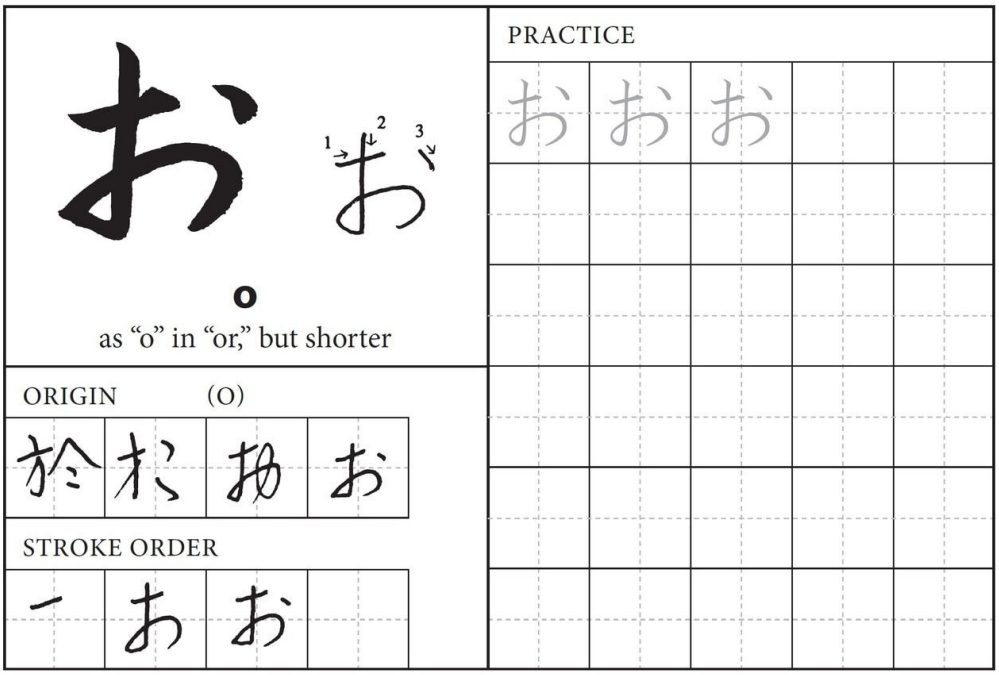


Figura 2.13. Plantilla de *Learning Japanese Hiragana And Katakana*

Pero como en cualquier idioma, no todas las personas escriben de la misma manera. En los cuadernos caligráficos se deja el suficiente espacio para dibujar el carácter, para que el usuario se acostumbre al trazo correcto en la manera que lo desee. La anchura, la altura y los arcos del trazado varían en función de cada persona y, aún así, somos capaces de generalizar las distintas representaciones y reconocer un carácter por diferente que sea del canónico (véanse Figuras 1.1 y 2.13)[4].

Como conclusión, para cualquier idioma, ya sean dos idiomas tan distintos como el castellano y el japonés, se utiliza el mismo método de enseñanza para la caligrafía: repetición y atención al trazado del carácter. Los cuadernos de caligrafía se utilizan hoy en día en colegios y escuelas de idiomas para que los niños aprendan su idioma natal o un nuevo idioma y tanto las personas adultas como los niños que aprenden estos idiomas son capaces de generalizar y hacer evolucionar su escritura en el idioma estudiado. Aunque haya una plantilla y una guía para dibujar la forma de los caracteres, el alumno intenta imitar su forma acorde a ello queriendo siempre modificar el carácter de manera inconsciente para hacerlo de manera cómoda y personal.

[4] En el ejercicio de la Figura 2.13 se pueden apreciar hasta 5 tipos diferentes de caligrafía.

2.2.2. Métodos actuales

Los métodos actuales en cuanto a la enseñanza de caligrafía japonesa se fundamentan en la misma base que los clásicos. Estos métodos evolucionan de la escritura en papel a la escritura en dispositivos móviles táctiles con aplicaciones que intentan imitar el proceso de aprendizaje de los cuadernos caligráficos clásicos, añadiendo incluso más ejercicios para mejorar la experiencia.

Estas aplicaciones tienen una gran ventaja frente a los métodos clásicos: **el audio** y el uso del **micrófono**. Por un lado, el uso del audio en este tipo de aplicaciones, y para este tipo de enseñanza, es una ventaja muy importante ya que supone un apoyo muy útil a la hora de aprender la pronunciación de los caracteres más complicados. Por otro lado, el micrófono del dispositivo se usa para poder contrastar la pronunciación correcta de los caracteres con la proporcionada por el usuario.

Además del audio y el uso del micrófono, también se utilizan **ejercicios asociativos** para aprender a asociar la pronunciación de los caracteres y su forma escrita. Aunque esta interacción es más llamativa en dispositivos táctiles y con la ayuda del audio, estos ejercicios de asociación se utilizan en algunos métodos clásicos.

Por último, aunque no todas las aplicaciones cuentan con ejercicios de escritura, las que sí lo hacen tienen un sistema de gestión de errores en el que reportan al usuario si ha escrito bien o no un carácter en función de una **plantilla**.

Para ver los diferentes métodos que las aplicaciones utilizan para el aprendizaje de la escritura de los caracteres japoneses se pondrán como ejemplo las aplicaciones ***LingoDeer*** [38], ***Write It! Japanese*** [39], ***Japanese Writing*** [40] y ***Write Japanese*** [41].

2.2.2.1 *LingoDeer*

LingoDeer es una aplicación de enseñanza que se especializa en los principales idiomas orientales como el japonés, el coreano o el chino. La aplicación está dedicada a la enseñanza de un idioma al completo, y, por ello, ofrece ayuda para el aprendizaje de los caracteres japoneses con audio, los ejercicios asociativos y dibujo.

Toda la aplicación está destinada a aprender principalmente vocabulario y frases

para, al final, poder sobrellevar una conversación en japonés. En este ámbito, *LingoDeer* ofrece un extenso y rico vocabulario al usuario y unos ejercicios muy buenos para aprenderlo. El escalamiento que hay de dificultad es correcto y por ello es una de las aplicaciones de enseñanza que más gusta a los usuarios.

A continuación se exponen algunos ejercicios de asociación de esta aplicación: relación de caracteres con sus homólogos fonéticos del otro silabario (Figura 2.14a), relación con su pronunciación en un audio reproducible (Figura 2.14b) y relación del carácter con su fonema escrito (Figura 2.14c).

(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3

Figura 2.14. Ejercicios de asociación de *LingoDeer*

Estos ejercicios son los únicos que están relacionados a la enseñanza de los caracteres básicos del japonés. Lo más cerca que está la aplicación de enseñar a escribir y dibujar correctamente estos caracteres es con una breve imagen en movimiento que detalla el orden del trazado, acompañada de un audio con su pronunciación.

Figura 2.15. Memorización de formas del carácter de *LingoDeer*

La aplicación no permite al usuario dibujar el carácter, simplemente muestra el recorrido correcto del trazo (línea negra) para que el usuario lo memorice. Al ser caracteres simples, esta aplicación no radica en la escritura de estos caracteres y, en cambio, enfatiza en la de los complejos. *LingoDeer* ofrece una sección para aprender a escribir estos caracteres: primero se escoge un carácter, después se enseña una visualización como la de la Figura 2.15 y por último se dibuja en el recuadro naranja.

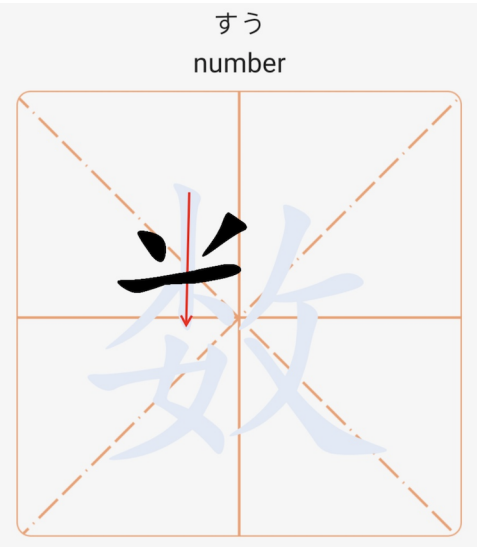
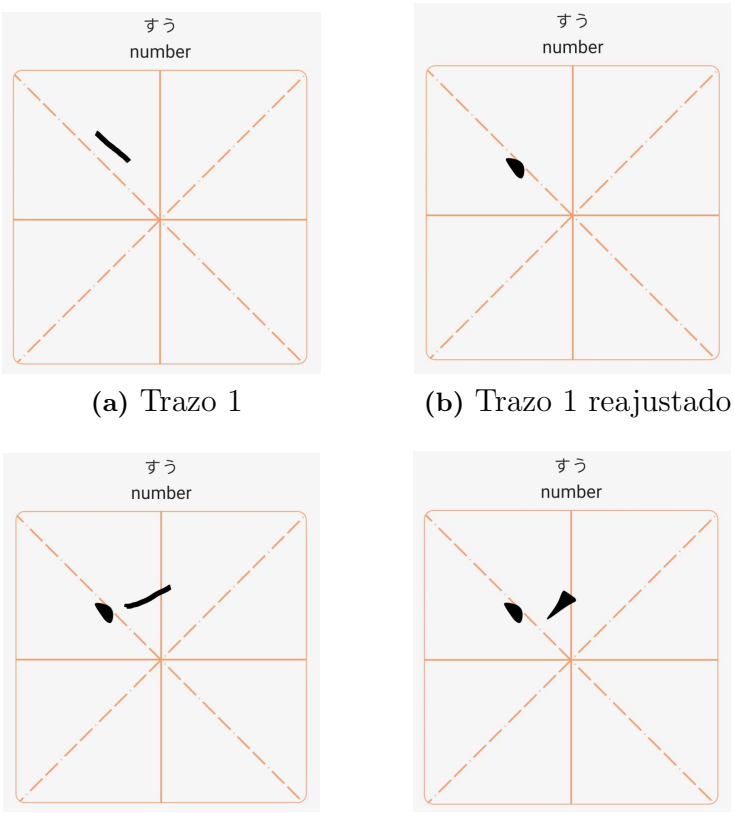


Figura 2.16. Escritura de carácter de *LingoDeer*

El usuario, acorde a la Figura 2.16, debe atender a los trazos que se realizan en la plantilla (carácter atenuado) y después reproducirlos en el orden correcto en el recuadro naranja sin esta. El problema está en que si los trazos no se realizan en su debida

posición con respecto a la plantilla, la aplicación rechaza el trazo realizado, teniendo que volver a comenzar. El remedio que implantan para que los trazos realizados correspondan *exactamente* a la plantilla, y así poder validar la correcta escritura del carácter, es *atraer* los trazos correctos a la plantilla, los posicionados cerca de esta, reajustándolos al propio estilo caligráfico de la plantilla (véase el proceso en la Figura 2.17). Esto perjudica gravemente al usuario y al aprendizaje a la hora de escribir con su propia caligrafía, ya que independientemente de cómo lo dibuje, la aplicación adaptará los trazos a su plantilla, dejando como inútil el trazo realizado por el usuario, cuando los trazos realizados en las Figuras 2.17a y 2.17c son correctos.



(a) Trazo 1 **(b)** Trazo 1 reajustado

(c) Trazo 2 **(d)** Trazo 2 reajustado

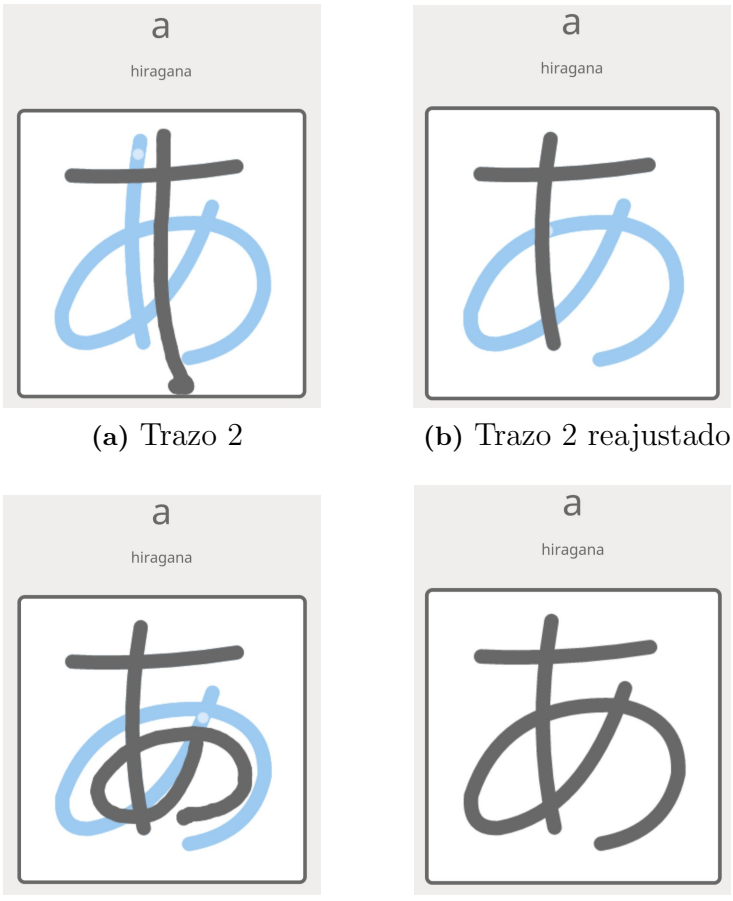
Figura 2.17. Proceso de ajustamiento de trazado de *LingoDeer*

Pese a permitir la práctica de la escritura de algunos caracteres japoneses, el aprendizaje se distorsiona debido a las restricciones del dibujo. Por supuesto, la aplicación cumple su cometido principal y puede haber personas que tras utilizar y haber aprendido con *LingoDeer* sepan hablar japonés, pero la escritura de este se ve muy debilitada debido a la falta de práctica de los caracteres base y a la falta de adecuación de un estilo caligráfico *libre*.

2.2.2.2 *Write It! Japanese*

Write It! Japanese es una aplicación destinada exclusivamente al aprendizaje de escritura de los caracteres japoneses. Ofrece al usuario todos los caracteres de ambos silabarios para practicar su escritura. Aparte de la práctica, la aplicación ofrece unos pequeños exámenes con **ejercicios asociativos** entre la pronunciación de un carácter (disponible en audio), y ejercicios de escritura en base a la pronunciación.

La escritura del carácter sigue los mismos procedimientos que los de *LingoDeer*: se enseña la plantilla con el orden del trazo que el usuario debe seguir y dibujar, y, posteriormente, dicha plantilla desaparece, dejando al usuario una *aparente* libre escritura en base a los trazos realizados previamente. Pese a ser una aplicación dedicada a la escritura, ocurre exactamente lo mismo que con *LingoDeer*: los trazos que no estén debidamente colocados, acorde a la plantilla, se eliminan o se reajustan a esta.



(a) Trazo 2 **(b)** Trazo 2 reajustado

(c) Trazo 3 **(d)** Trazo 3 reajustado

Figura 2.18. Proceso de ajustamiento de trazado de *Write It! Japanese*

Si se sigue el orden de trazado y el carácter queda como el de la Figura 2.18c, este se puede reconocer como el mismo que el de la plantilla, pero con diferente caligrafía. Pese a este claro reconocimiento, la aplicación no ofrece soporte a diferentes caligrafías más que a la de su plantilla. En los exámenes se establece el mismo recuadro que el de la figura pero sin que la plantilla esté visible, manteniendo la técnica de ajustamiento del trazo, y haciendo que el usuario pueda acertar la pregunta mediante el dibujo de trazos aleatorios (ya que alguno se podría ajustar a la plantilla).

2.2.2.3 *Japanese Writing*

Japanese Writing es otra aplicación exclusivamente destinada a la enseñanza de caligrafía japonesa. Tiene exactamente las mismas funciones que *Write It! Japanese* de práctica y exámenes pero con una principal mejora: en la práctica existe *un poco* de libertad caligráfica a la hora de dibujar el carácter. Primero enseñan el orden del trazado y después se puede comenzar a dibujar. Si un trazo está lo *suficientemente cerca* del trazo de la plantilla y está correctamente dibujado, este se quedará en esa posición, sin ningún reajuste. Una vez realizada la representación, esta se podrá comparar a la plantilla para que el usuario decida si está correctamente interpretado o no.

(a) Carácter あ representado con caligrafía libre

(b) Comparación con plantilla

Figura 2.19. Proceso de comparación de caracteres de *Japanese Writing*

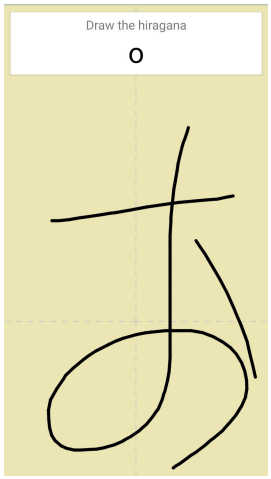
En la Figura 2.19a se observa que se ha trazado el carácter, siguiendo el orden de los trazos, y la aplicación ha determinado que cada trazo está lo *suficientemente* ajustado a la plantilla como para darlo por válido. En cambio, en los exámenes esta libertad se pierde, ya que no aceptan *aproximaciones* de los caracteres preguntados.

Esta aplicación realiza el primer paso a la aceptación de múltiples representaciones de un solo carácter, aunque es cierto que tiene sus limitaciones: si, por ejemplo, se quiere representar un carácter dado en unas proporciones bastante más grandes que las de la plantilla, la aplicación eliminará los trazos.

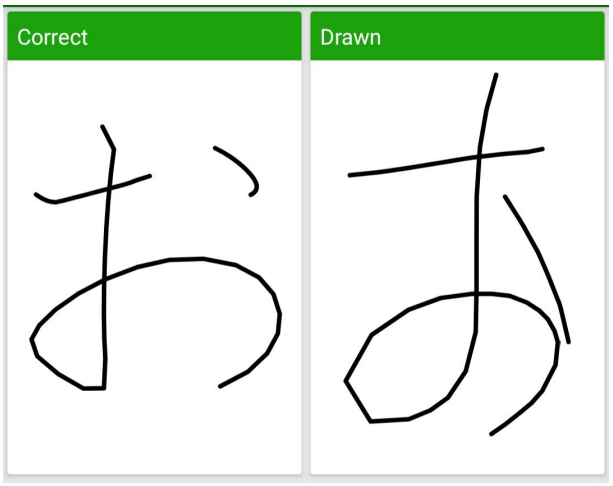
2.2.2.4 *Write Japanese*

Write Japanese es otra aplicación enteramente destinada al aprendizaje de la escritura y que realmente imita a la perfección a los métodos clásicos. Esta aplicación acepta cualquier tipo de dibujo y lo compara con la correspondiente representación del carácter preguntado. Es decir, esta aplicación acepta cualquier tipo de caligrafía y aun así será capaz de reconocer si se corresponde con el carácter dado. A continuación se ven dos ejemplos con dos caligrafías diferentes.

En el tipo de caligrafía mostrado en la Figura 2.20, más alargada que la original, se ve que el arco de la derecha es bastante más largo y el círculo es más ancho que la mostrada en la resolución y aún consigue evaluarlo correctamente.



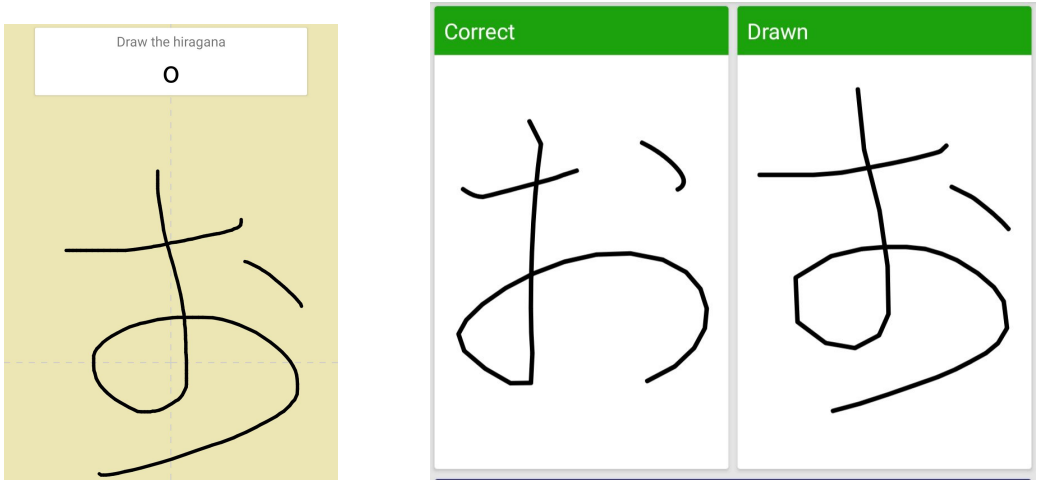
(a) Carácter お tipo 1



(b) Resultado 1

Figura 2.20. Reconocimiento de caracteres variando la caligrafía de *Write Japanese* 1

En cambio, en el tipo de caligrafía mostrado en la Figura 2.21, el círculo se adecua más al original pero el terminado del trazo central es mucho más alargado. Aun así, la aplicación es capaz de reconocer que el carácter dibujado es el mismo.



(a) Carácter お tipo 2 **(b)** Resultado 2

Figura 2.21. Reconocimiento de caracteres variando la caligrafía de *Write Japanese* 2

Esta aplicación consigue que cualquier usuario dibuje los caracteres que se le pregunten como lo haría en un cuaderno de caligrafía japonesa, atendiendo o no al trazado idóneo de los caracteres.

2.2.3. Comparativa de métodos

Por un lado, los métodos clásicos, como los cuadernos de caligrafía japonesa, son una herramienta muy eficaz tanto para aprender a escribir el idioma nativo de un niño o niña japonés como para aprender a escribir el idioma en la edad adulta. Estos ofrecen libertad al alumno a la hora de escribir, siempre y cuando siga las reglas de trazado. Además, dependiendo del cuaderno, cuando se estudie un carácter, el alumno podrá estudiar su origen, las distintas caligrafías más comunes o las diferentes aplicaciones que puede llegar a tener en el idioma.

Por otro lado, los métodos actuales quieren suplantar a los métodos clásicos mediante la inclusión de mejoras interactivas que son atractivas para los usuarios como el audio o el micrófono y técnicas autodidactas. Aquí se ha visto una progresión de mejora en aplicaciones para acercarse a la efectividad de los métodos clásicos pero, aún así, muchos de los métodos actuales no consiguen reproducir el nivel de

abstracción y generalización caligráfica existente en la escritura. El hecho de que la mayoría de estos métodos reescriban, o directamente eliminen, los trazos del usuario al no adecuarse a una plantilla hacen que este no aprenda en absoluto, ya que el dibujo representado no es el que el usuario ha trazado.

Con la aplicación *web* que se estudia en este trabajo se intenta solventar el principal problema de los métodos actuales: **no permitir una caligrafía variable**. Gracias a las redes de neuronas convolucionales, la aplicación *web* va a ser capaz de reconocer cualquier tipo de caligrafía para un carácter dado, de algún modo parecido a lo realizado por David Meeuwis en su aplicación *Write Japanese*. Este tipo de aplicaciones tienen un problema: no se atiende al orden del trazo. Pero si es cierto que en los cuadernos de caligrafía, y en general en situaciones reales, se puede escribir un carácter de cualquier idioma en la manera que se desee, siempre y cuando se parezca al canon establecido (véanse las Figuras 2.22 y 2.23).

Como se ve en la Figura 2.22, entre los dos caracteres escritos (2.22b y 2.22c) no se aprecia ninguna diferencia a pesar de haberse escrito en un orden de trazos diferente, y ambos se reconocen como el carácter original (2.22a).

(a) Orden de trazado idóneo **(b)** Orden: 1-2-3 **(c)** Orden: 3-2-1

Figura 2.22. Escritura de carácter japonés con diferente orden de trazado

Para entenderlo mejor, en el castellano hay un ejemplo claro de este fenómeno con la letra "p". La letra "p" se escribe en un solo trazo, siendo el orden 1-2-3. Pese a esto, hay personas que escriben la letra "p" empezando por el rabo y acabando en el cierre del círculo (2.23b), y hay otras que empiezan a escribir la letra por el cierre de la concavidad, acabando en el rabo (2.23c), y ambas se reconocen como la letra "p" original (2.23a). Pese a no realizarse el trazo 1 y 2, la letra "p" sigue reconociéndose y aceptándose como una escritura válida.

(a) Orden de trazado idóneo **(b)** Orden: 3 **(c)** Orden: 3 inverso

Figura 2.23. Escritura de carácter románico con diferente orden de trazado

Por esto, pese a que en este tipo de aplicaciones no sea necesario un trazado correcto del carácter para ser aceptado, al usuario, en la aplicación propuesta, se le ofrecerá una guía visual opcional del trazado reglamentario para que este desarrolle una buena praxis a la hora de dibujar el carácter. Esta ayuda es opcional ya que aunque el orden del trazo sea relevante en etapas tempranas de aprendizaje, no repercute en el reconocimiento del carácter y, además, ayuda a una adaptación personal del trazado y estilo más rápida que con métodos convencionales.

3. EXPERIMENTACIÓN

En esta sección se exponen las herramientas utilizadas para la experimentación y la metodología que se ha seguido para ella, además de los experimentos más relevantes de cada dominio estudiado (*Dogs vs. Cats*, *Emotion Detection From Facial Expressions* − FERC y CIFAR10) acompañados de explicaciones y conclusiones de lo aprendido con cada uno de ellos.

3.1. Herramientas

3.1.1. *Framework*

El modelo de predicción se basa en una red de neuronas convolucional, y como tal, se necesita una herramienta de trabajo potente, sencilla y que de soporte a la GPU para poder realizar los cálculos en el menor tiempo posible. Las herramientas propuestas son: **Keras** [42] como extensión de Tensorflow [43] y **PyTorch** [44]. Ambas son unas herramientas ampliamente usadas en el mundo científico y académico, programables en el lenguaje de programación **Python**. Como consecuencia directa, son las principales herramientas dedicadas al *deep learning* más buscadas en Google en los últimos 6 años.

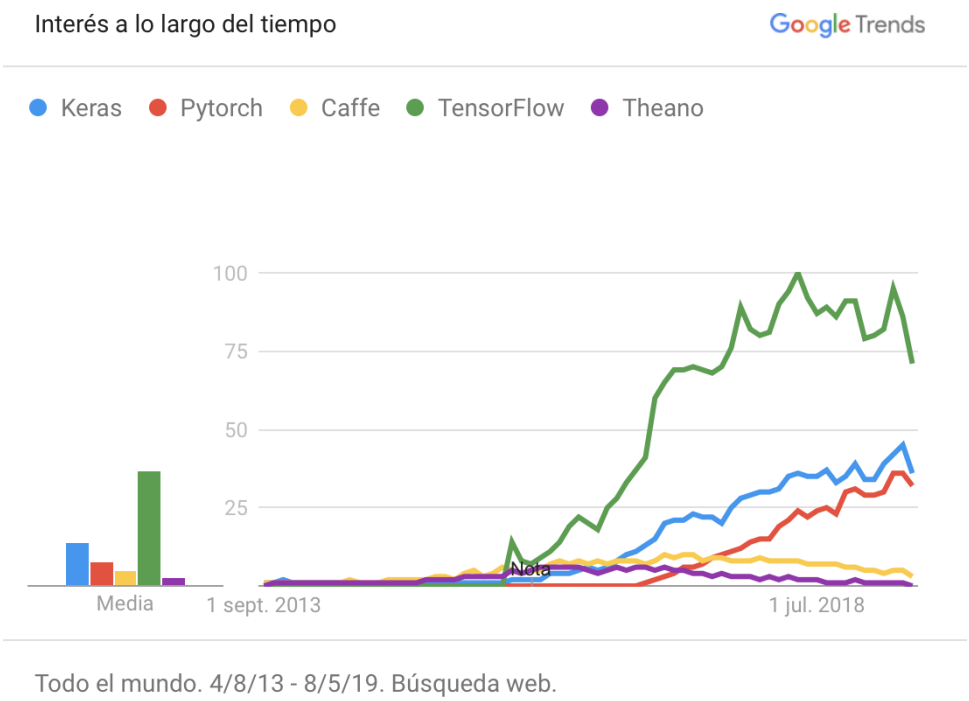


Figura 3.1. Tendencia de interés entre *frameworks* para *deep learning*

Por un lado, Keras ofrece una interfaz (API) de alto nivel bastante sencilla y muy accesible tanto para el científico como para una persona que está empezando en el mundo del *deep learning*. Gracias a esto, esta herramienta se sustenta con miles de tutoriales, documentación y libros dedicados. Al ser una librería de alto nivel, su nivel de abstracción es mayor con respecto a las operaciones que el algoritmo realiza *por detrás* y permite al programador modificar únicamente los parámetros esenciales del algoritmo elegido.

Por otro lado, PyTorch es una herramienta de bajo nivel que requiere que el programador sepa las bases matemáticas intrínsecas del algoritmo, ya que permite la modificación de cualquier parámetro. Este hecho hace que PyTorch esté ganando popularidad entre los investigadores últimamente como herramienta alternativa a Keras y Tensorflow. Aparte, ambas herramientas ofrecen soporte de GPU para agilizar los cálculos: PyTorch requiere la elección explícita de cuando se quiere utilizar la unidad de procesamiento gráfico o la central mientras que Keras maneja esta administración de procesos implícitamente.

Pese a la gran ventaja que supone que PyTorch permita más libertad para optimizar un algoritmo que Keras, la principal ventaja de Keras es la flexibilidad en la exportación que ofrece de los modelos creados, gracias a que los guarda como archivos H5. Con Keras se pueden exportar y optimizar fácilmente modelos a Javascript con Tensorflow.js para utilizarlos en páginas *web*. PyTorch, en cambio, no ofrece tanta versatilidad (a la hora de exportar los modelos creados a otros entornos). PyTorch almacena los modelos en archivos *pickle*, disponibles solo para Python.

Por lo tanto, por la facilidad a la hora de exportar los modelos creados, por su usabilidad accesible a todos y reusabilidad, y por la gran cantidad de documentación que existe, se ha elegido Keras con Tensorflow como principal herramienta para la construcción de las redes de neuronas convolucionales.

3.1.2. Versiones

El ordenador dedicado a los experimentos es un ordenador con Ubuntu 16.04 como sistema operativo, con un procesador Intel(R) Core (TM) 2 Dúo y con una gráfica Nvidia GeForce GTX 1050 Ti. En este ordenador se han instalado las diferentes herramientas que permiten el uso de Keras con Tensorflow. Las versiones de las herramientas que se han utilizado a lo largo del trabajo se muestran en la Tabla 3.1.

Herramienta	Versión
Python	3.5.2
Keras	2.0.4
Tensorflow-gpu	1.2.0

Tabla 3.1. Versiones de las herramientas para *deep learning*

Las versiones de Keras y Tensorflow-gpu no son las más actualizadas. Esto se debe a la incompatibilidad con el procesador central del ordenador: al ser una CPU antigua, esta no está preparada para soportar las versiones más actualizadas de las herramientas. Por ello, se requirió la instalación de una versión anterior de Keras y Tensorflow-gpu.

3.2. Metodología

Para el estudio de las diferentes redes de neuronas convolucionales con cada conjunto de datos que aquí se presenta, se ha creado una nomenclatura para nombrar los diferentes archivos e imágenes que se extraen de cada experimento. La nomenclatura elegida para los experimentos es la siguiente:

<# de prueba general >-<Dominio>-<Estructura>-<# de prueba especifica>

Donde el número de prueba general es un número de 4 dígitos que establece el **orden general** entre todos los experimentos, independientemente del conjunto de datos que se esté estudiando, el dominio es una **abreviatura para el conjunto de datos** estudiado en el experimento, la estructura es la **estructura de la base convolucional** utilizada para el experimento, y, por último, el número de prueba específico es un número de 4 dígitos que establece el **orden de los experimentos de un conjunto de datos específico.**

Esta estructura se sigue para todos los diferentes archivos generados por cada experimento y, para diferenciarlos entre sí, se añaden las siguientes terminaciones a la nomenclatura (véase Tabla 3.2).

Descripción	Terminación
Programa de Python	*.py*
Modelo con los pesos óptimos	*.h5*
Registro de ejecución del programa	*.txt*
Imagen de la matriz de confusión para el conjunto de Prueba	*-test-conf.png*
Imagen de la matriz de confusión para el conjunto de Validación	*-validation-conf.png*
Imagen de la gráfica de acierto para Validación y Entrenamiento	*-acc.png*
Archivo de texto con las matrices de confusión de Prueba y Validación junto con los aciertos totales de Entrenamiento, Validación y Test	*-conf.txt*
Archivo de texto de valor real vs. valor predicho	*-pred-vs-real.txt*

Tabla 3.2. Terminaciones de los archivos creados por cada experimento

Por lo tanto, por cada experimento realizado, estos ocho archivos se generan automáticamente conteniendo la información más relevante. Con esta nomenclatura se intenta crear un entorno estable y ordenado para los diferentes experimentos que se realizarán en el trabajo.

Para llevar un registro de cada experimento con comentarios sobre los resultados y demás conclusiones extraídas, se utiliza la herramienta **Trello**. Esta herramienta permite a un usuario ordenar de manera personalizada la información que desee con la ayuda de un tablero. En el tablero se pueden crear diferentes listas refiriéndose a diferentes temas a ordenar, y dentro de cada una de estas listas, se crean tarjetas. Estas tarjetas son acciones o subtareas específicas dentro del tema principal de la lista en donde, en este caso, se va a escribir todo lo relacionado con los experimentos y con el desarrollo de la aplicación.

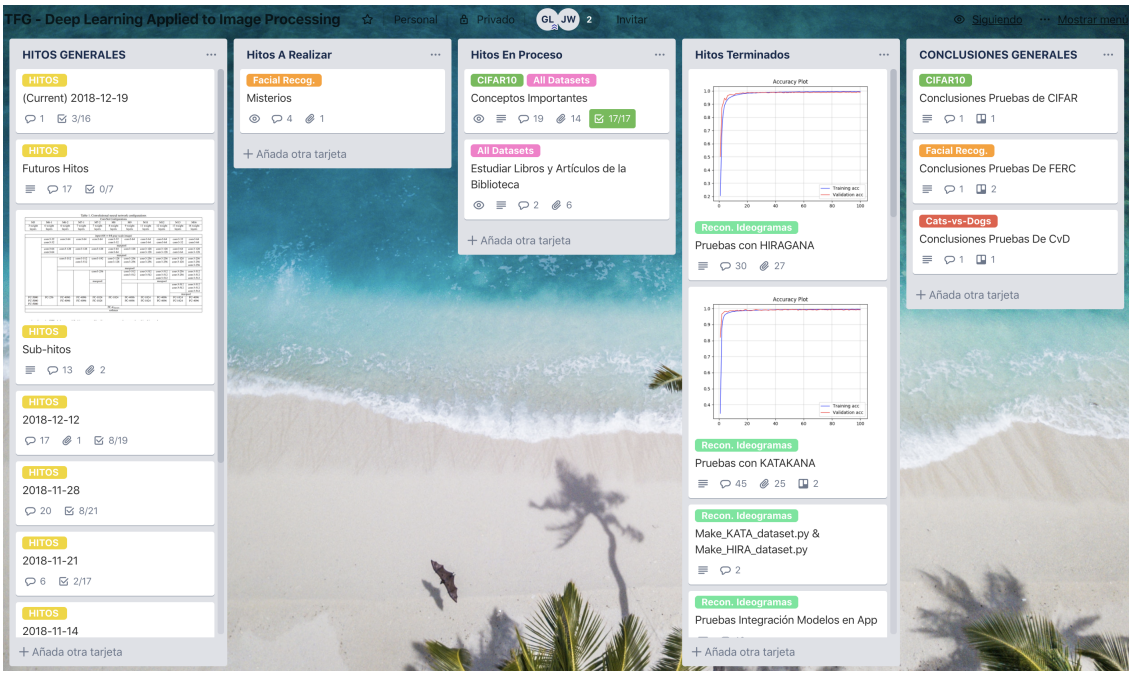


Figura 3.2. Exposición del tablero de Trello usado para la experimentación

Por cada conjunto de datos se crea una tarjeta para apuntar los datos relevantes de cada experimento. Una vez terminada la ejecución de una prueba, se crea una entrada en forma de comentario en la tarjeta conteniendo: la información de la red, las gráficas y matrices de confusión, las tasas de acierto de cada conjunto de entrenamiento y una breve conclusión del experimento argumentando los errores o los buenos resultados. De esta manera, todas las conclusiones extraídas y todas las mejoras implantadas se pueden consultar de manera cómoda y rápida, para observar cómo han ido evolucionado los experimentos.

3.3. Pruebas previas

En la experimentación previa se ha trabajado con tres conjuntos de datos diferentes: *Dogs vs. Cats*, *Emotion Detection From Facial Expressions* (FERC) y CIFAR10. Son conjuntos que no tienen nada en común y que han servido para familiarizarse con el *deep learning* y sus librerías, y para adecuar los parámetros de tal manera que al trabajar con el conjunto real de datos fuese lo más eficiente posible.

Antes de comenzar con la experimentación, en los dominios estudiados se realiza una partición del conjunto de datos en entrenamiento, validación y prueba para realizar el entrenamiento de la red debidamente y para que el algoritmo nos devuelva un valor real y aceptable de acierto:

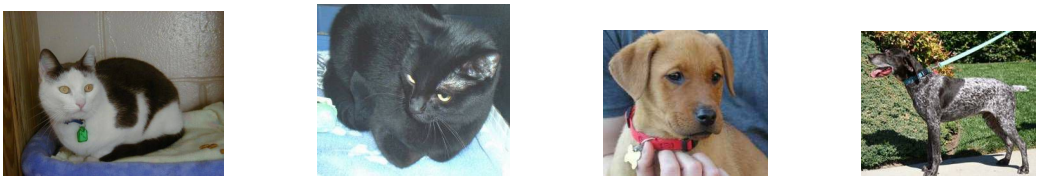
- El **conjunto de entrenamiento** sirve para que la red entrene. Es muy importante que este conjunto no se utilice para probar la eficiencia de la red, ya que como este conjunto es el que se utiliza para aprender, las muestras ya las conocería y, por tanto, daría un acierto muy alto en falso.
- El **conjunto de validación** sirve para que, a la vez que entrena la red, se vayan evaluando muestras distintos para determinar si el entrenamiento se está haciendo debidamente.
- El **conjunto de prueba** una vez que el modelo ya está completo y el aprendizaje ha concluido, será utilizado para validar el modelo y probar su eficiencia, eficacia y generalización con muestras completamente diferentes a las usadas durante el entrenamiento.

Con estos tres conjuntos derivados se asegura que las redes creadas para los distintos dominios aprendan patrones nuevos durante todo su entrenamiento, mientras que también hace que los resultados no sean falseados por patrones ya aprendidos, demostrando una correcta generalización.

Como ayuda e información adicional, se puede consultar el **Anexo I**, que contiene una explicación en profundidad del código base utilizado para la experimentación, y el **Anexo II**, que contiene el listado completo de todos los experimentos realizados, junto con el porcentaje de acierto de cada prueba.

3.3.1. *Dogs vs. Cats*

El conjunto de *Dogs vs. Cats* (CvD) consta de **25.000 imágenes** a color RGB de perros y gatos no ordenadas con dimensiones no uniformes. Es un conjunto colgado en la plataforma de competición de *machine learning* Kaggle [45]. Las imágenes no vienen clasificadas en carpetas por clase, pero cada archivo tiene un nombre con la forma: *"<cat/dog><número de imagen>.png"*. Por lo que con *<cat/dog>* se puede discernir directamente la clase de cada imagen sin ningún programa previo de ordenación.



(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.3. Muestras de *Dogs vs. Cats*

A lo largo de los primeros experimentos con este conjunto se han ido añadiendo diferentes medidas de apoyo a la evaluación del modelo, como la matriz de confusión. Estos experimentos no se comentarán ya que contienen errores y medidas falsas sobre los modelos probados. Es a partir de la prueba 0005-CvD-VGG16-0002 que las métricas utilizadas están correctamente programadas. De este experimento en adelante, las imágenes se redimensionan a 75×75 para que tengan unas dimensiones uniformes a lo largo del entrenamiento y la base convolucional utilizada es una VGG16 [46] adaptada a las dimensiones de las imágenes. Para la evaluación de los modelos, se utiliza la tasa de acierto del conjunto de validación.

En el primer experimento válido, 0005-CvD-VGG16-0002, no se utiliza el conjunto de datos entero. Se realiza una criba aleatoria quedando el conjunto de entrenamiento con 7.200 imágenes, el de validación con 4.800 imágenes y el de prueba con 3.000 imágenes. En cada conjunto hay un reparto igualitario entre clases para que cada conjunto esté perfectamente equilibrado. Este reparto se lleva acabo en todos los demás experimentos.

El hecho de que no haya imágenes alteradas y el uso de pocas imágenes de entrenamiento resulta en un acierto bastante reducido: **69,56 %** (Figura 3.4). Por eso en el siguiente experimento, se añade un vasto *data augmentation* para que la red generalice de manera idónea, además de ampliarse el conjunto de entrenamiento en 10.000 imágenes, reduciendo el de validación y prueba en 2.000 imágenes y 3.000 imágenes respectivamente.

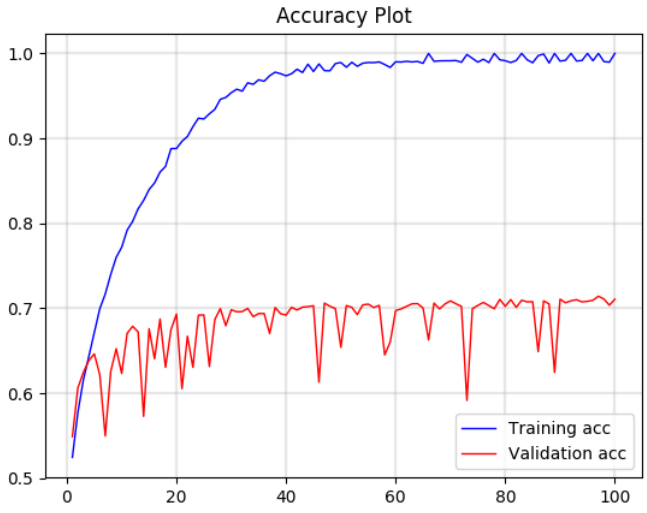


Figura 3.4. Experimento 0005-CvD-VGG16-0002

El experimento 0006-CvD-VGG16-0003 y el 0007-CvD-VGG16-0004 (Figura 3.5) son el mismo, excepto que el 0007 tiene un mayor tiempo de entrenamiento. Esto se debe a que en el experimento 0006 se observó que ambas curvas de acierto no convergían en 100 ciclos, por lo que se le añadió al entrenamiento siguiente 100 ciclos más. Con este añadido el modelo alcanza un **83 %**, pero las curvas siguen sin llegar a converger del todo.

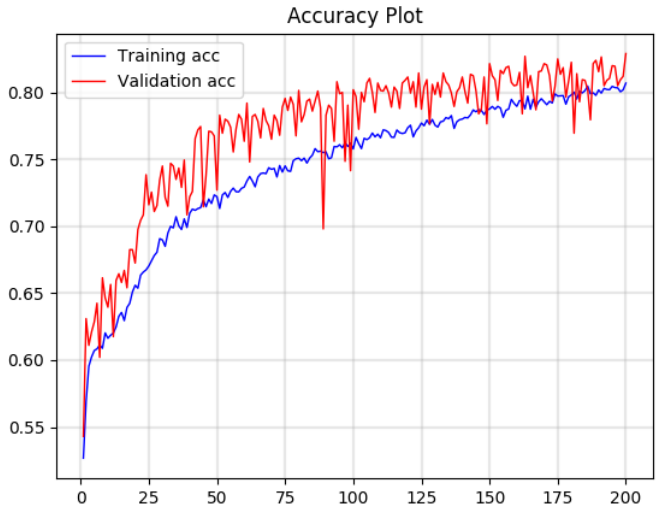


Figura 3.5. Experimento 0007-CvD-VGG16-0004

Aunque el modelo mejora con respecto al 0005, se observan varias peculiaridades en la Figura 3.5: la curva de acierto de validación obtiene mejor resultado que la de entrenamiento a lo largo de la prueba y el rizado de la misma es más brusco. Por una parte, el acierto del conjunto de entrenamiento es el que tiende a converger antes que el de validación debido a que es el conjunto usado por el modelo para aprender patrones y dichos patrones se acaban reconociendo antes. Y por otra parte, el rizado se puede deber a que la red encuentra muchos mínimos locales durante el entrenamiento o al mal ajuste de parámetros de la red. Estas anomalías se deben en parte a la falta de recursos para el aprendizaje de patrones y al vasto *data augmentation*. Para intentar remediar estos problemas, en el experimento 0008-CvD-VGG16-0005 se amplía aún más el conjunto de entrenamiento y validación, y se reduce el de prueba: 16.000, 6.400 y 2.600 imágenes respectivamente, dejando el *data augmentation* sin modificar y se aumentan los ciclos de entrenamiento a 300 (véase Figura 3.6).

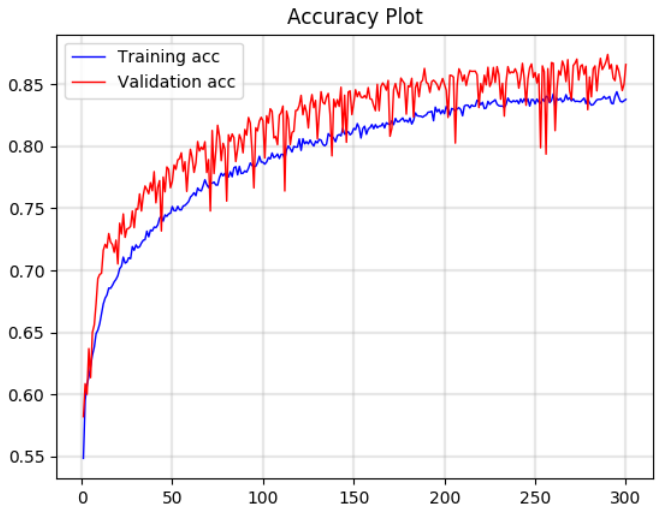


Figura 3.6. Experimento 0008-CvD-VGG16-0005

Se observa que en el experimento 0008, el rizado se mitiga un poco, pero se mantiene a lo largo de la prueba, mientras que la curva de validación permanece con una tasa de acierto mayor. Pero ambas curvas si que convergen a un valor de acierto con el aumento de ciclos. Durante las próximas pruebas se sigue intentando resolver los problemas mencionados en el experimento 0007 sin éxito hasta el experimento 0012-CvD-VGG16-0009. En este experimento clave se elimina todo el vasto *data augmentation* añadido en el 0007 dejando únicamente un volteo horizontal y una mínima traslación de anchura y altura en las imágenes. Además de esto, se reajustan los lotes de imágenes y el número de imágenes por lote, `Imgs`, para cada conjunto en base a la relación $I_L \leq \texttt{Imgs} \times P_L$, siendo L el conjunto de entrenamiento, validación y test, I_L las imágenes del conjunto L y P_L el número total de lotes de imágenes a extraer de L antes de finalizar un ciclo de entrenamiento[5]. De esta manera se obtiene la siguiente gráfica.

[5] En los experimentos previos se establecen números al azar porque no se tenía conocimiento de la relación $I_L \leq \texttt{Imgs} \times P_L$ explicada en el Anexo I, Sección A.2.

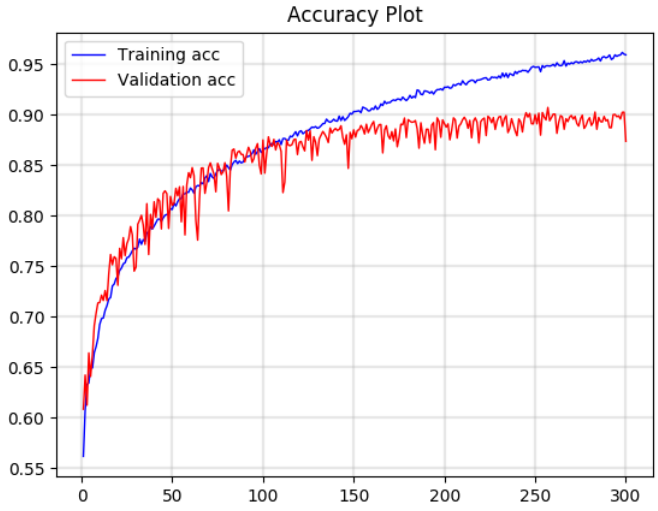


Figura 3.7. Experimento 0012-CvD-VGG16-0009

Gracias a la reducción del *data augmentation* y al ajuste de los lotes, la red tiene un comportamiento realmente bueno, dando un acierto del **87,17 %** haciendo desaparecer el rizado. Al hacerse dos cambios tan significativos en el modelo, es difícil atribuir el causante de la mejora. Para ello se realiza un nuevo experimento dejando el reducido *data augmentation* de la prueba 0012 y manteniendo los lotes con los parámetros establecidos en las pruebas anteriores a la 0012. Se concluye de la prueba que al mantener el resultado obtenido en la Figura 3.7, la mejora se le atribuye exclusivamente a la reducción del *data augmentation*.

En pruebas posteriores se juega con el *data augmentation* para probar si se puede mejorar el modelo con alguna transformación más. Por eso, se añade una rotación a las imágenes y en la prueba 0014-CvD-VGG16-0011 el modelo alcanza un acierto de **88,39 %** (Figura 3.8), mejorando levemente el modelo. En la prueba 0016-CvD-VGG16-0013, se aumenta el rango de valores para la traslación de anchura y altura de las imágenes, y se aumentan los ciclos de entrenamiento a 400. Este aumento en ciclos se debe a que el *data augmentation* comienza a ser abundante y el modelo empeora, como en las Figuras 3.5 y 3.6, sin que el acierto llegue a converger a ningún valor por falta de tiempo. La prueba 0016, con los cambios mencionados, alcanza un **91,17 %** de acierto (Figura 3.9), el mejor resultado obtenido en esta experimentación[6].

[6] La mejor tasa de acierto para este dominio en particular, extraído de la propia competición de Kaggle, es de 98,91 %. Resultado obtenido por el equipo Pierre Sermanet.

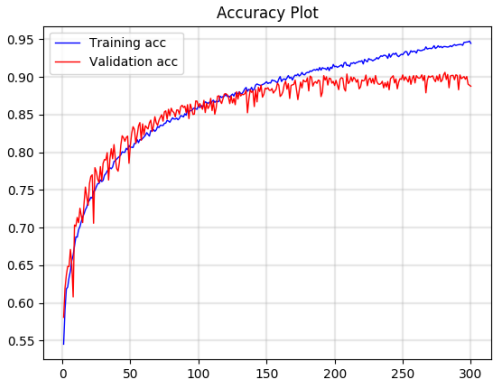


Figura 3.8. Experimento 0014-CvD-VGG16-0011

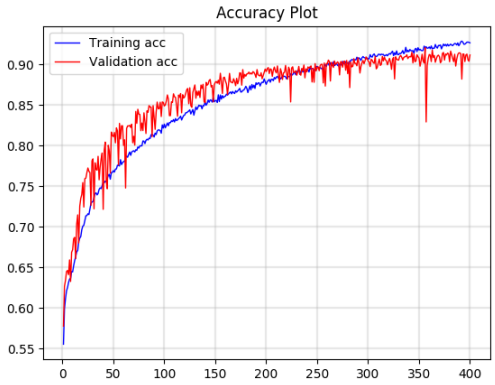


Figura 3.9. Experimento 0016-CvD-VGG16-0013

Como conclusión, un *data augmentation* mal ajustado supone la creación de ejemplos para entrenar irreales y aporta al modelo casos poco probables, dando lugar a una mala generalización. Al crear malos ejemplos para que entrene el modelo, este no alcanza a diferenciar entre unas clases y otras, dando un acierto en entrenamiento inferior al de validación.

Gracias al *data augmentation* moderado el modelo es capaz de reconocer las diferenciaciones entre clases y el acierto pasa de ser de un 70 % (experimento 0005-CvD-VGG16-0002) a un 90 % (experimento 0016-CvD-VGG16-0013). Con esta mejora, esta técnica de generación de imágenes se reafirma como la más eficaz para hacer que el modelo generalice mucho mejor con imágenes completamente nuevas pero significantes en el dominio.

Con respecto a los datos utilizados, se ha visto que la utilización de más muestras en el conjunto de entrenamiento y validación mejora el acierto y comportamiento del modelo (experimento 0007-CvD-VGG16-0004 y 0008-CvD-VGG16-0005). Con más imágenes de las que aprender, el modelo extrae mucha más información y, así, aprende más patrones para poder alcanzar un mejor acierto en predicción.

3.3.2. FERC

El conjunto de datos de *Emotion Detection From Facial Expressions* (FERC) obtenido de Kaggle [47] es un conjunto de datos que consta de **13.718 imágenes**. Este conjunto sirve para distinguir expresiones faciales de personas mundialmente famosas. En consecuencia, el conjunto tiene ocho clases diferentes: *anger, contempt, disgust, fear, happiness, neutral, sadness* y *surprise*.

(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.10. Muestras de FERC

Las dimensiones de todas las muestras son de 350×350. Al tener unas dimensiones tan grandes, se pueden generar cantidades ingentes de datos, por lo que la red podría tardar mucho tiempo en cada prueba. Por esto, a las imágenes se les aplica una redimensión a 64×64. Con estas dimensiones, la red no se satura y mantiene la relevancia de la información de las imágenes.

Las muestras no vienen ordenadas de ningún modo y el nombre de cada muestra no es lo suficientemente significativo como para establecer una diferenciación condicionada en ello. Los nombres de cada imagen varían en función de cada celebridad: *"<nombre><apellido><número de imagen asociada al personaje>.png"*. Como en el nombre de las muestras no hay suficientes datos como para extraer la clase, se crea un programa auxiliar que analiza todas las imágenes y las ordena por su clase con la ayuda de un fichero *csv*. Este fichero contiene un listado de todas las muestras del conjunto ordenadas de la siguiente forma: *nombre imagen−clase*[7].

Al comparar el fichero *csv* y las imágenes se observan irregularidades: hay imágenes que no aparecen en el fichero y que si lo hacen en las muestras, y viceversa. Para remediar este problema, se eliminan a mano las imágenes del conjunto y las tuplas irregulares del fichero. Una vez terminado el proceso de criba, con el fichero y conjunto resultante, se buscan los pares entre imágenes y tuplas por medio del nombre de la imagen, para así extraer su clase y clasificarla, dejando al conjunto con un total de **12.756 imágenes**. Son con estas muestras con las que se trabajará en los experimentos.

Para este conjunto se han realizado numerosos experimentos sin ninguna diferencia significativa. Todos las pruebas se mantienen en el **96 %−97 %** de acierto desde el inicio de estas. Se han estudiado varias estructuras de la base convolucional diferentes a la VGG16, obteniendo aparentemente un *muy buen* resultado. El conjunto

[7] El fichero *csv* viene conjuntamente con los datos en la descarga del conjunto.

está creado para reconocer expresiones faciales, una tarea realmente compleja, por lo que en un principio se esperaban unos resultados entorno al 60 %. De hecho, en Kaggle, en la clasificación de la competición, **la media de resultados es de un 52,68 %** de acierto, por lo que los resultados podrían estar falsificados de alguna manera.

Para averiguar porqué los resultados son tan buenos comparados con los de la competición, se añade la matriz de confusión a los experimentos. Al ejecutar la prueba 0018-FERC-VGG16-0017[8] con 7.652 imágenes en el conjunto de entrenamiento, 3.807 en el de validación y 1297 en el de test, se consigue un acierto del **89,30 %** (Figura 3.11).

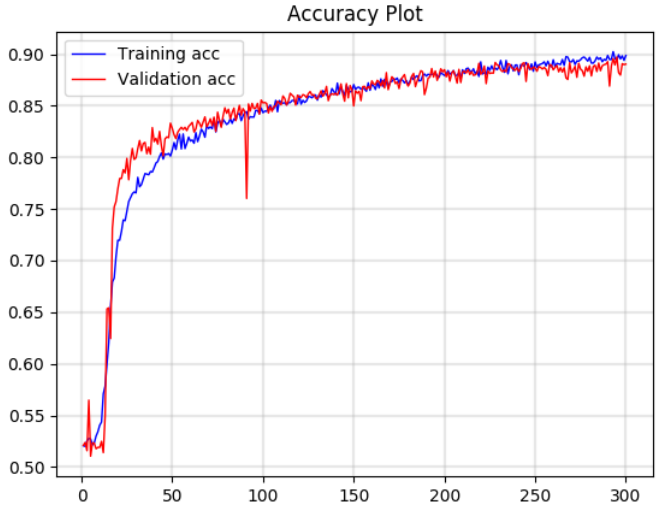


Figura 3.11. Experimento 0018-FERC-VGG16-0017

El buen resultado obtenido en las anteriores pruebas se mantiene en este experimento pero al observar la matriz de confusión observamos una anomalía.

[8] Los anteriores experimentos se realizaron antes de establecer la nomenclatura general de las pruebas y, en consecuencia, este experimento es el *primero* con esta nomenclatura, pese a haber pruebas previas. Las anteriores pruebas también se ejecutaron antes de incluir la matriz de confusión a *Dogs vs. Cats*.

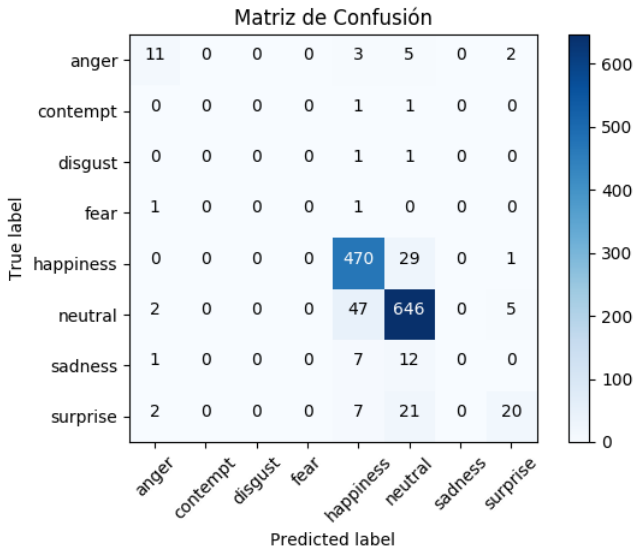


Figura 3.12. Matriz de confusión del experimento 0018-FERC-VGG16-0017

Se observa que prácticamente todas las muestras del conjunto de prueba (el utilizado, en este caso, para la creación de la matriz) pertenecen a las clases *happiness* y *neutral* únicamente. Esto significa que el conjunto de prueba tiene una distribución de datos totalmente desbalanceada. Esta distribución se extrapola a los demás conjuntos, lo que significa que no solo el conjunto de prueba está mal balanceado, sino que todo el conjunto de **FERC está desbalanceado** (véase Figura 3.13).

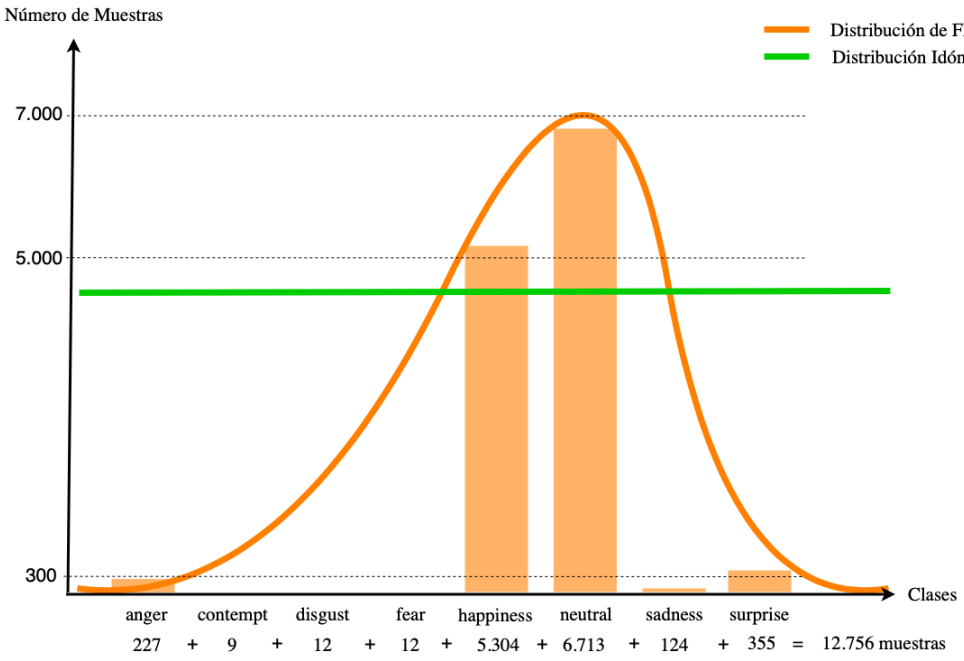


Figura 3.13. Distribución de datos en FERC

Volviendo a la Figura 3.12 se observa que las clases *happiness* y *neutral* obtienen

un 94 % y un 92,28 % de acierto respectivamente, mientras que las demás obtienen prácticamente un 0 % de acierto, dando en el modelo un acierto total de **88,43 %** en el conjunto de prueba. El simple hecho de que las clases mayoritarias hayan clasificado muy bien, hace que el modelo dé un acierto muy elevado. Al tener tantas muestras de diferencia (más de 5.000) con respecto a las demás clases, *happiness* y *neutral camuflan* el resto de malos resultados de las demás clases, reflejando un acierto del modelo irreal y falso. Todo esto lleva a la conclusión de que todos los experimentos que se han realizado en el conjunto son poco aprovechables. Para poder conseguir un modelo realmente relevante se tendrían que aplicar técnicas que equilibren el conjunto de datos para conseguir una distribución idónea y equitativa para todas las clases.

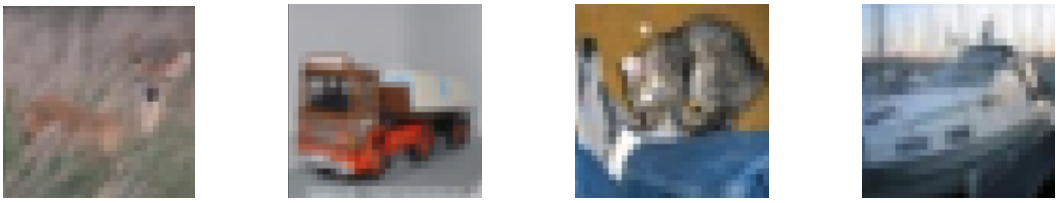
Las técnicas de generación o eliminación de muestras para el correcto balanceo de las clases son: ***oversampling* y *undersampling***. Por un lado, el *oversampling* es una técnica que se aplica a las clases desfavorecidas para tratar de igualarlas en número de muestras a las clases dominantes. Esta técnica consiste en duplicar los datos o en aplicar algoritmos para generar nuevas muestras en la clase menos representada haciendo que se vea poblada hasta alcanzar a las demás clases, balanceando así los datos. Y, por otro lado, el *undersampling* es una técnica que consiste en la eliminación aleatoria de muestras de las clases dominantes en un dominio dado. Esta eliminación se lleva a cabo hasta que las muestras de todas las clases estén repartidas por igual, balanceando el conjunto. El *undersampling* conviene aplicarlo para conjuntos en los que las clases minoritarias tengan suficientes muestras como para extraer información relevante, pero como en este caso, las clases minoritarias tienen muy pocas muestras, conviene utilizar *oversampling*. Ya que el objetivo no es trabajar con este conjunto, y la aplicación de técnicas de *oversampling* es un área bastante extensa, los experimentos con FERC se paran al momento de descubrir este desbalance de clases.

Como conclusión, el desbalance de clases es un problema muy importante que se debe de reconocer rápidamente para evitar experimentaciones poco significativas y que se puede remediar con técnicas como el *oversampling* y el *undersampling*. Gracias a que este conjunto está desbalanceado, se ha visto la eficacia de utilizar una métrica de evaluación adicional, como la matriz de confusión, ya que solo con la tasa de acierto, el desbalance no se puede apreciar. Y, por último, con la cantidad de pruebas y cambios de la estructura del modelo, que todas ellas produzcan una tasa de acierto tan parecida se debe también al desbalance de clases. Con una estructura A y B, las clases dominantes pueden obtener la misma tasa de acierto, mientras que con la A, las minoritarias pueden tener un acierto muy alto, y con la B, uno

muy bajo. Pero como las clases dominantes obtienen el mismo resultado con ambas estructuras, el resultado del modelo no varía. Es por esto que se debe prestar especial atención a cómo está distribuido el conjunto de datos con el que se trabaja antes de empezar a experimentar con él.

3.3.3. CIFAR10

El conjunto de datos CIFAR10 consta de **60.000 imágenes** de 32×32: 50.000 imágenes destinadas al conjunto de entrenamiento y 10.000 al de prueba [30]. Como tienen una dimensión uniforme, no se aplica una redimensión al preprocesar los datos. Los datos están distribuidos equitativamente en las siguientes diez clases: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión.



(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.14. Muestras de CIFAR10

Este conjunto, al ser un referente en el *machine learning*, viene incluido en Keras ya ordenado y clasificado correctamente, por lo que no se necesita un programa auxiliar para ordenar las imágenes.

El conjunto de validación se extrae del de entrenamiento quedando: 40.000 imágenes en entrenamiento, 10.000 en validación y otras tantas en prueba. Ya que se tiene una tasa de acierto de referencia, 95 % de la Tabla 2.2, en estos experimentos se trata de llegar a esa cifra en la medida de lo posible. En la toma de contacto con el conjunto, la prueba 0020-CIFAR-VGG16-0003, se obtiene un **78,78 % de acierto** sin ningún tipo de problema (Figura 3.15), pero al examinar la matriz de confusión se observa que hay una gran discrepancia entre la clasificación entre perros y gatos: hay 123 gatos que se clasifican como perros y 168 perros que se clasifican como gatos. Estas dos clasificaciones hacen que el modelo empeore haciendo que las demás clases alcancen prácticamente un 85 % de acierto, sin equivocaciones relevantes.

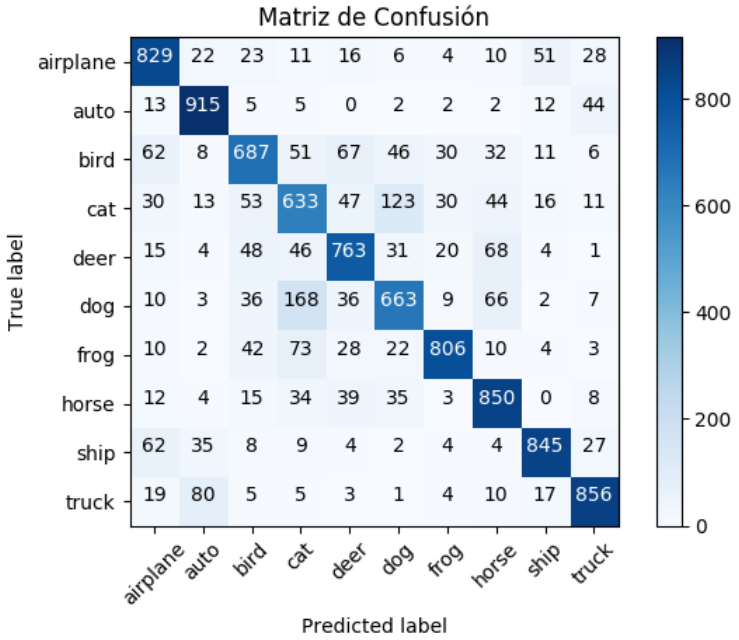


Figura 3.15. Matriz de confusión del experimento 0020-CIFAR-VGG16-0003

Para la prueba 0021-CIFAR-VGG16-0004 se reduce el *data augmentation* quitando las rotaciones a las imágenes, mejorando el modelo en un 1 %, pero manteniendo la mala clasificación de perros y gatos. Al observar que con el conjunto no consigue mejorar la clasificación, se reordenan las imágenes por cada conjunto dejando 50.000 imágenes en entrenamiento y 5.000 tanto en validación como en prueba. Añadiendo 10.000 imágenes al conjunto de entrenamiento y ejecutando la prueba 0022-CIFAR-VGG16-0005, el modelo mejora de nuevo llegando al **81 % de acierto** (Figura 3.16).

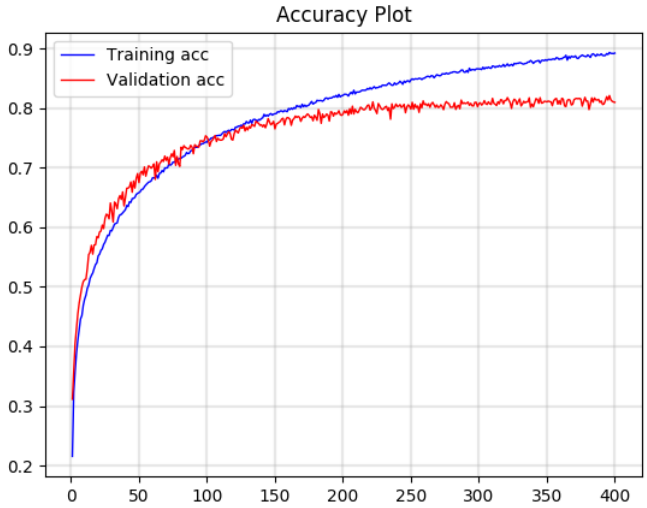


Figura 3.16. Experimento 0022-CIFAR-VGG16-0005

Ya que ni la ampliación del conjunto de entrenamiento ni la mitigación del *data augmentation* han supuesto una mejora realmente relevante, se estudia en el siguiente experimento una nueva base convolucional: $C(16)C(32)M \times C(64)M \times C(128)M \times Drop(0{,}2)$, donde C es una capa convolucional, M capa de *max pooling* y *Drop* una capa de ***dropout***. Esta capa de *dropout* se añade para eliminar un posible **sobreaprendizaje** del modelo. El sobreaprendizaje es un fenómeno que ocurre cuando un modelo se adecua en exceso a las muestras del conjunto de entrenamiento, aprendiendo únicamente los patrones de dichas muestras, y en consecuencia, produciendo una mala generalización. Esto se puede observar en un modelo mediante las gráficas de acierto: si el acierto de entrenamiento aumenta a medida que el de validación baja, existe sobreaprendizaje en la red. Para mitigarlo, se utiliza el *dropout*. El *dropout* es una técnica que consiste en la desactivación totalmente aleatoria de neuronas activas o inactivas. Con esta desactivación de neuronas se elimina información relacionada con los patrones de entrenamiento consiguiendo, en cierta medida, *retrasar* el sobreaprendizaje de la red, pero pagando el coste de perder información *posiblemente* relevante. Con todo esto en mente, el resultado obtenido en la prueba 0023-CIFAR-CCMCMCMDD-0006 es de **75,80 % de acierto** (Figura 3.17). El resultado es bastante peor que los de anteriores pruebas y se le atribuye a la utilización del *dropout* consecuente de esa pérdida de información. Por esta misma razón, se opta por no utilizar esta técnica en los experimentos.

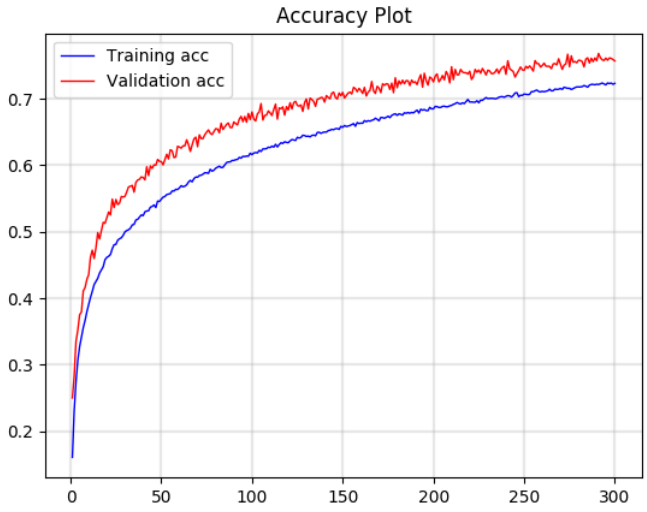


Figura 3.17. Experimento 0023-CIFAR-CCMCMCMDD-0006

En el experimento 0025-CIFAR-CCMCMCMDD-0008 se elimina el *dropout*, dejando la misma estructura, y se añaden los parámetros de *strides=1* y *padding="same"*

a las capas convolucionales[9]. Esto se hace para poder extraer más información de las muestras y para que la red tenga más parámetros con los que trabajar. Con este añadido, el modelo consigue un **81,40 %** mejorando levemente el resultado del experimento 0022.

Como última medida para intentar sobrepasar la marca del 82 % de acierto, se realiza un cambio del optimizador de gradiente[10] en la prueba 0027-CIFAR-CCMCMCMDD-0010 de `RMSProp` a `Adadelta`, manteniendo todas las mejoras realizadas hasta ahora en el modelo. El modelo alcanza un **88,28 % de acierto** (Figura 3.18). Aparte de obtener el mejor resultado hasta ahora, el cambio de optimizador también hace que el modelo converja de manera más rápida con respecto a los anteriores experimentos, reduciendo el tiempo de experimentación a casi la mitad.

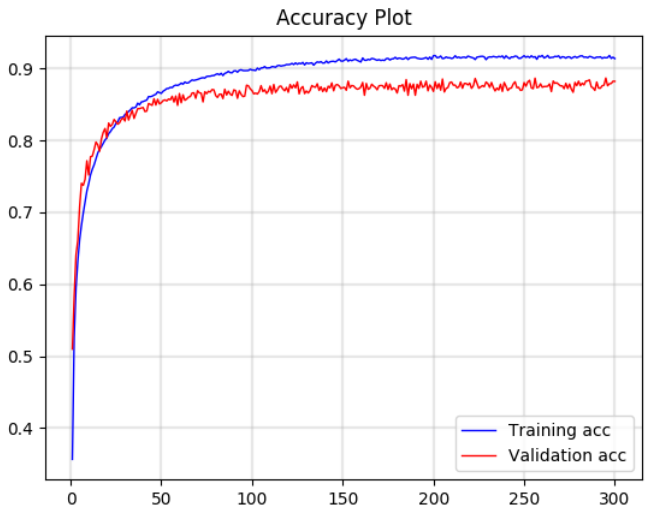


Figura 3.18. Experimento 0027-CIFAR-CCMCMCMDD-0010

Previamente, el optimizador utilizado era `RMSProp`[11] con un *learning rate* de 0,00002 que producía una convergencia muy lenta[12]. En base a los resultados obtenidos en la prueba 0027, se opta por utilizar `RMSProp` con un *learning rate* = 0,0001 para probar que la tasa de aprendizaje es proporcional a la convergencia de la red. Se obtiene un **85,24 % de acierto** (Figura 3.19). Pese a este resultado, se observa un comportamiento anómalo: la curva del conjunto de entrenamiento una vez ha

[9] Explicación de ambos parámetros en profundidad en el Anexo I, Sección A.3.
[10] Explicación del optimizador de gradiente en el Anexo I, Sección A.4.
[11] Al ser recomendado por François Chollet en su libro *Deep Learning with Python*, en todos los experimentos anteriores, incluidos los de otros dominios, se utiliza el optimizador `RMSProp`.
[12] Explicación del *learning rate* en el Anexo I, Sección A.4.

convergido a su valor máximo empieza a decrecer.

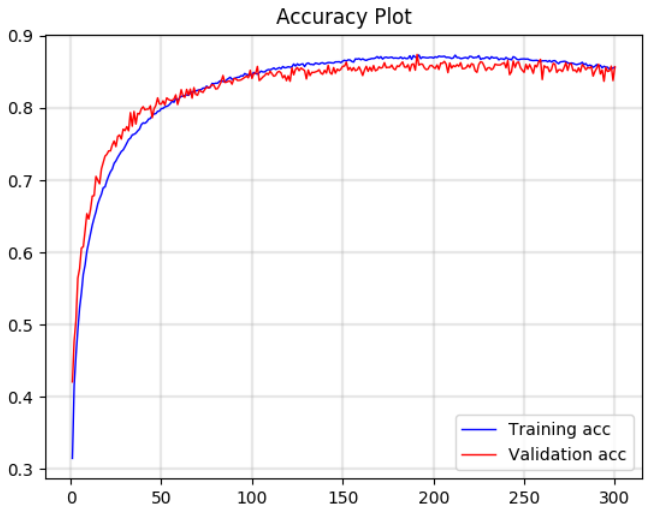
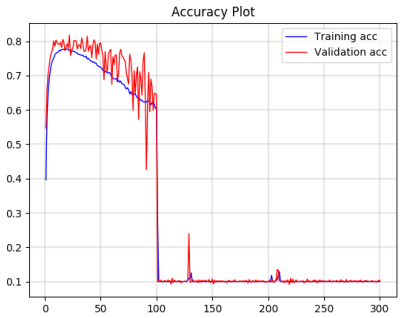


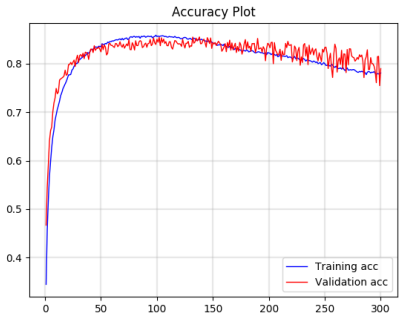
Figura 3.19. Experimento 0028-CIFAR-CCMCMCMDD-0011

Se observa que tanto el experimento 0027 como el 0028 convergen alrededor del ciclo 100 del entrenamiento, llegando prácticamente al mismo resultado. Por lo que queda claro que, entre otros factores, el principal problema de los experimentos anteriores era el valor tan pequeño de la tasa de aprendizaje.

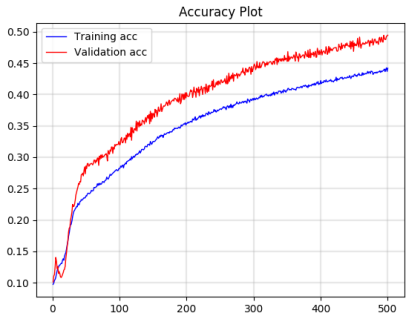
Finalmente, no se llega a la meta del 95 %, obteniendo como mejor resultado un 88,28 % de acierto en la prueba 0027. Se prueban distintas variaciones de tasas de aprendizaje con los diferentes optimizadores, pero sin conseguir grandes resultados. Se pueden observar en la Figura 3.20: la *caída* de la red por completo (Figura 3.20a), la convergencia decreciente (Figura 3.20b − presumiblemente el desarrollo natural del modelo 0028-CIFAR-CCMCMCMDD-0011) y la convergencia creciente anómala (Figura 3.20c).



(a) Exp. 0026



(b) Exp. 0029



(c) Exp. 0030

Figura 3.20. Experimentos con comportamiento anómalo

Como conclusión, gracias al aumento de información a procesar por la red, tanto por el aumento de muestras de entrenamiento como con el añadido de *padding* y *strides*, se consigue una mejora que, aunque pequeña, ayuda a la capacidad de generalización de la red. El *padding* y el *stride* resultan fundamentales para la máxima extracción de información en todas las capas convolucionales de la red, con el único inconveniente de que, al tener más parámetros, el modelo tarda más en entrenar. Y, por último, se ha visto la importancia de la tasa de aprendizaje en los modelos dependiendo del optimizador utilizado. Como no hay una regla genérica que permita la elección inmediata de un optimizador con una tasa de aprendizaje, hace que la búsqueda de un optimizador y valor de tasa óptimos sea muy compleja. Por lo que el haber encontrado que con la combinación de `Adadelta` con `lr=1,0` el modelo se comporta de una manera correcta (véase Figura 3.18) es buena noticia.

3.4. Modelo de reconocimiento de caracteres japoneses

Una vez se han realizado todos los experimentos con conjuntos de datos de prueba y se ha adquirido el suficiente conocimiento sobre como se comporta una red de neuronas convolucional en la práctica, se procede a trabajar con el conjunto

definitvo de datos: ETL *Character Database*. Este conjunto se usa para crear dos subconjuntos de caracteres japoneses escritos, según el silabario al que corresponden, los cuales se utilizan para el estudio y experimentación en este trabajo. En consecuencia, **se crean dos conjuntos de datos diferentes: uno para *katakana* y otro para *hiragana***.

En esta sección se explica como se ha obtenido este conjunto de datos y su posterior ordenación por clase. Y, finalmente, se comentan las pruebas previas que sirven de base para la experimentación final de ambos conjuntos.

3.4.1. Conjunto de datos ETL *Character Database*

El conjunto de datos ETL *Character Database* está compuesto por 9 subconjuntos de datos de imágenes escaneadas de caracteres escritos e impresos formalmente suministrados por el Laboratorio de Electrotecnia (ETL − *Electrotechnical Laboratory*) y está siendo administrado actualmente por el reorganizado sucesor, el Instituto de Ciencias Industriales Avanzadas y Tecnología (*National Institute of Advanced Industrial Science and Technology* − AIST) [48].

Este conjunto consiste, en su totalidad, de 1,2 millones de imágenes de numerales, símbolos y caracteres latinos y japoneses compilados en 9 subconjuntos (ETL-1 *Character Database* hasta ETL-9 *Character Database*). Como este trabajo se enfoca únicamente en los caracteres básicos de *hiragana* y *katakana*, no se utilizan todos los subconjuntos que se proporcionan inicialmente, **solo se utilizan los subconjuntos 1, 4, 7, 8 y 9** por albergar en su interior imágenes de dichos caracteres.

Estos subconjuntos consisten en un compendio de archivos binarios donde están almacenadas las imágenes y cada uno está configurado de diferente manera. En la especificación de cada subconjunto se muestra con todo detalle lo que almacena cada archivo, de qué manera, las dimensiones de las imágenes y un código auxiliar escrito en Python para poder acceder a estas imágenes. A continuación, se explica la ordenación de cada uno de estos subconjuntos divididos entre los que almacenan caracteres de *katakana* y de *hiragana*.

3.4.1.1 Ordenación del conjunto de *katakana*

El subconjunto ETL-1 *Character Database* está compuesto por 13 archivos binarios y solo los archivos ETL1C-07, ETL1C-08, ETL1C-09, ETL1C-10, ETL1C-11,

ETL1C-12 y ETL1C-13 albergan en su interior imágenes escaneadas de caracteres japoneses pertenecientes a *katakana*, por lo que solo se tratan estos archivos del subconjunto. Estos archivos contienen un total de 71.961 imágenes de caracteres de 64×63. En este conjunto, existen caracteres obsoletos como ヰ (WI), ヱ (WE), WU, YI y YE. Al estar en desuso, las muestras de dichos caracteres se obvian.

Tras la ordenación y contando los caracteres que han sido obviados, el conjunto total del ETL-1 *Character Database* se reduce a un total de 64.763 imágenes de caracteres escritos de *katakana*.

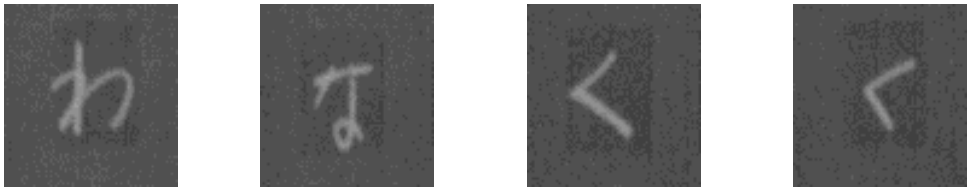
(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.21. Muestras de ETL-1 *Character Database*

En el conjunto de datos ETL *Character Database* no se encuentran más imágenes de caracteres del silabario *katakana*, por lo que el compendio de **64.763 imágenes** del ETL-1 *Character Database* va a conformar el conjunto de datos de *katakana* que se va a utilizar en los experimentos.

3.4.1.2 Ordenación del conjunto de *hiragana*

El subconjunto ETL-4 *Character Database* contiene únicamente imágenes de caracteres japoneses escritos pertenecientes al silabario de *hiragana*. El subconjunto contiene un archivo binario que almacena un total de 6.120 imágenes de 72×76 en el que también existen caracteres obsoletos, por lo que se obvian todas esas imágenes para la ordenación. Con ello, **el subconjunto se queda en 5.520 imágenes**.



(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.22. Muestras de ETL-4 *Character Database*

Si comparamos las casi 65.000 imágenes en el conjunto de *katakana* con las 5.500 extraídas para *hiragana*, se decide que no son suficientes imágenes para conformar el conjunto de *hiragana*. Por lo que se añade al ETL-1 *Character Database* el subconjunto ETL-7 *Character Database*. Este subconjunto contiene 4 archivos binarios que almacenan un total de 16.100 imágenes escaneadas de 64×63 de caracteres japoneses de *hiragana*.

En el ETL-7 *Character Database* no vienen incluidas imágenes de caracteres obsoletos, por lo que todas ellas son válidas. Por tanto, el conjunto de *hiragana* asciende a un total de **21.620 imágenes**.

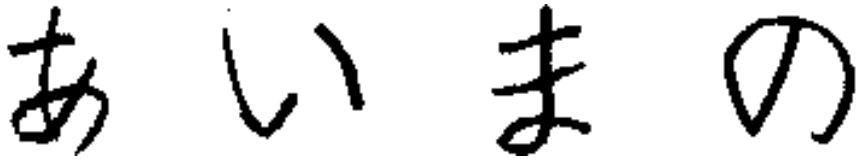
(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.23. Muestras de ETL-7 *Character Database*

A pesar de la gran cantidad de muestras, no son suficientes ya que en el ETL *Character Database* existen todavía otros dos subconjuntos que almacenan caracteres de *hiragana*: ETL-8B2 *Character Database* y el ETL-9B *Character Database*.

El subconjunto ETL-8B2 *Character Database* contiene 3 archivos binarios pero solo un archivo contiene caracteres de *hiragana*, aunque los tres archivos contienen primordialmente *kanji* (caracteres más complejos). El archivo que contiene los caracteres básicos contiene 11.040 imágenes de 64×63.

En este conjunto se almacenan las variaciones de los caracteres japoneses: el carácter た (TA) tiene una variación だ con la que le cambia la pronunciación a DA gracias a ese acento. Existen 23 variaciones diferentes, es decir, 23 clases más a tener en cuenta, pero como se quieren ordenar solo los caracteres básicos, estas variaciones se criban y se obtienen un total de **7.360 imágenes**.



(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.24. Muestras de ETL-8B2 *Character Database*

Por último, el subconjunto ETL-9B *Character Database* contiene cinco archivos binarios almacenando imágenes de 64×63 de caracteres de *hiragana* y *kanji*. En este subconjunto pasa lo mismo que en el anterior: contienen las variaciones de los caracteres, además de los obsoletos, por lo que al cribar todas esas imágenes el subconjunto se reduce a un total de **9.200 imágenes**.

(a) Ej. 1 **(b)** Ej. 2 **(c)** Ej. 3 **(d)** Ej. 4

Figura 3.25. Muestras de ETL-9B *Character Database*

Juntando todas las muestras de los 4 subconjuntos utilizados, **el conjunto de *hiragana* contiene un total de 38.180 imágenes**.

3.4.2. Preprocesamiento de imágenes para ETL *Character Database*

Se ha observado que tanto en el conjunto de *katakana* y de *hiragana* las dimensiones de las imágenes, pese a ser prácticamente uniformes, no son unas dimensiones idóneas tal que $W_I = H_I$. Ya que la entrada de la red neuronal requiere de una imagen con dimensiones $W_I = H_I$, al redimensionar las imágenes de estos conjuntos para cumplir el requisito, puede que los píxeles que faltan repercutan en la imagen redimensionada, haciendo que se pierda información vital. Para tratar de solventar este problema y conseguir unas imágenes de dimensiones $W'_I \times H'_I$ tal que $W'_I = H'_I$, **se añade un pequeño *padding* de píxeles a las imágenes**.

Este *padding* copia en intensidad de píxel a sus vecinos más próximos para poder ajustarse lo más posible al histograma de grises de la imagen original. Si se crease el *padding* enteramente negro, se estaría distorsionando el histograma de la imagen

por un extremo, con lo cual en el momento que se quiera ajustar el contraste, no se podría hacer por el extremo de los negros (ya que hay un negro límite). El añadido de este *padding* mantiene el ratio de la imagen original sin llegar a distorsionarla.

Se observa en la Figura 3.26 que apenas hay diferencia entre la imagen original y la redimensionada, por lo que el *padding* no afecta en absoluto a la información contenida en la muestras de ambos conjuntos y se consigue redimensionar las muestras para que tengan unas dimensiones $W'_I \times H'_I$ tal que $W'_I = H'_I$.

Figura 3.26. Ajuste de dimensiones de imagen con *padding*

3.4.3. Evaluación de modelos

Creados los conjuntos, se quiere dejar claro que **ambos tienen 46 clases definidas** acorde a los caracteres de los silabarios. El conjunto de *katakana* tiene **64.763 imágenes** con una media de alrededor de 1.407 imágenes por clase, y el conjunto de *hiragana* tiene un total de **38.180 imágenes** con 830 por clase. Estos datos informan de que ambos conjuntos están correctamente balanceados y listos para la experimentación.

A diferencia de las pruebas anteriores, para que el programa de experimentación no resulte tedioso, para estos dos conjuntos se crea un programa auxiliar que realiza una división aleatoria de las muestras en entrenamiento, validación y prueba como se ha venido haciendo en los demás conjuntos. En los experimentos previos no se

ha realizado esa aleatorización de los datos ya que las muestras de los diferentes conjuntos no están ordenadas de ninguna forma aparente. Pero, en el caso de los conjuntos ETL *Character Database*, las muestras vienen ordenadas en base a los diferentes escritores de los caracteres, mostrando un patrón visible de tendencia en ellos. Por esto, en los experimentos con los conjuntos principales de *hiragana* y *katakana* que se exponen a continuación, se hace una **aleatorización de los datos** previa a la división en entrenamiento, validación y prueba. Este añadido de aleatoriedad en los conjuntos sirve para evitar posibles patrones de datos preestablecidos y, en consecuencia, un posible *overfitting* del modelo.

3.4.3.1 *Katakana*

En base a lo aprendido en las pruebas previas, para la primera prueba relevante del conjunto de *katakana*, 0034-KATA-VGG16-0005, además de añadirse el *padding*, se realiza una partición del conjunto en: 50.963 imágenes para entrenamiento, 6.900 para validación y otras tantas para el de prueba. Como se observó con el conjunto *Dogs vs. Cats*, el aumentar el número de muestras de entrenamiento mejoraba el modelo, por lo que el conjunto de entrenamiento se crea en un principio con un número suficientemente grande de muestras. Además de esto, se entrena con una estructura VGG16, se añade el mismo *data augmentation* que dio el mejor resultado en *Dogs vs. Cats* y se entrena con unas imágenes de 64×64 (para reducir el tiempo de entrenamiento) y con el optimizador `adadelta`.

Con todas estas mejoras introducidas, se obtiene un **99,18 % de acierto** (Figura 3.27). Se observa una *muy* rápida convergencia del modelo, al igual que cuando se utilizó este mismo optimizador en FERC y, además, no se observa *overfitting*.

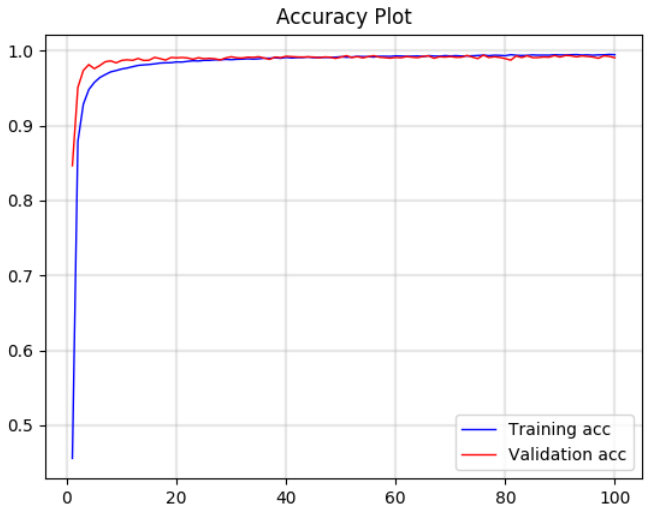


Figura 3.27. Experimento 0034-KATA-VGG16-0005

Con estos tan buenos resultados en la primera toma de contacto, se piensa que ocurre lo mismo que con FERC, pero al estar los datos bien balanceados, el modelo no obtiene falsos resultados.

En los dos experimentos siguientes, se prueba a aumentar las muestras de validación a 13.800 imágenes y a reducir las de entrenamiento a 44.063 (0035-KATA-VGG16-0006) y a modificar la base convolucional mediante una extensión sobre la VGG16 (0036-KATA-VGG16-0007), manteniendo la extensión sobre el conjunto de validación. Se obtiene un **98,99 % y un 99,15 % de acierto respectivamente**. Se aprecia que en las 3 pruebas realizadas, la variación entre el porcentaje de acierto no es muy significativa, llegando al 0,2 %, y consecuentemente, son poco concluyentes. Se realiza la prueba 0037-KATA-VGG16-0008, exactamente igual que la anterior, excepto que se redimensionan todas las imágenes del conjunto a 32×32 para observar si se produce alguna fluctuación importante en el acierto del modelo con la reducción de tamaño e información de las muestras.

Se obtiene un acierto del **98,94 %** (Figura 3.28), obteniendo una diferencia importante que no se aprecia en ninguna gráfica o métrica: el tiempo invertido en el entrenamiento es mucho menor que con imágenes de 64×64 y el resultado se mantiene. Para los tres experimentos anteriores se requieren 2 horas y media de entrenamiento, mientras que para esta prueba se requieren solo 40 minutos. Esto significa que con la cuarta parte de parámetros a entrenar, la red es capaz de alcanzar el mismo resultado que con el cuádruple de datos.

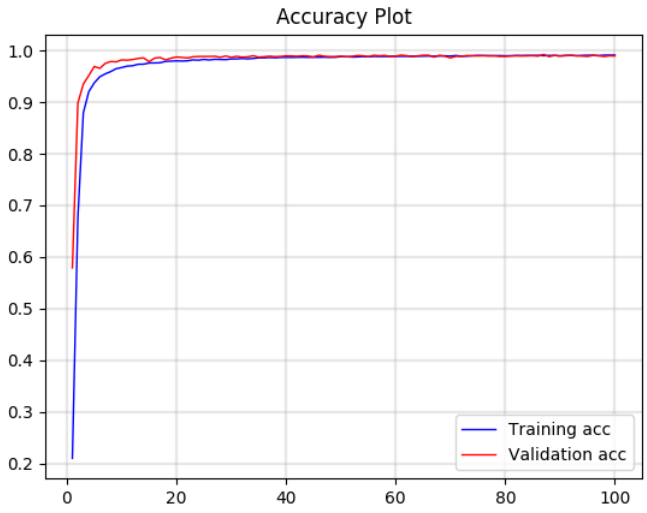
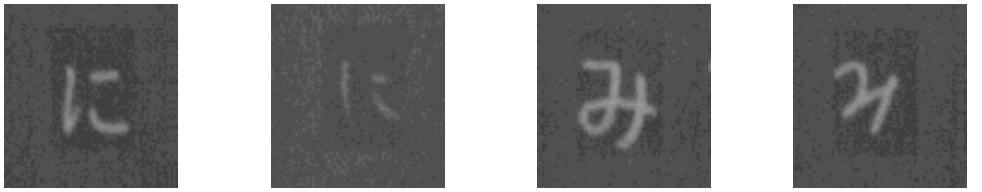


Figura 3.28. Experimento 0037-KATA-VGG16-0008

Por lo tanto, estos experimentos sugieren que independientemente de los cambios que se realicen en el modelo, la red va a seguir extrayendo la información suficiente como para alcanzar un resultado muy competitivo: entre el 98 % y el 99 % de acierto. El hecho de que los resultados difieran tan poco unos con otros hace que técnicas como la validación cruzada sean poco útiles, ya que no se va a poder extraer ninguna conclusión significativa y el resultado final va a ser el mismo que el obtenido en una prueba.

3.4.3.2 *Hiragana*

El conjunto de *hiragana* consta de imágenes de 4 subconjuntos diferentes con mucha variedad. En un principio, las muestras del ETL-4 *Character Database* se consideran *outliers* por su mala definición de imagen comparadas con las de los demás subconjuntos y porque las imágenes de los caracteres, aunque estén bien catalogadas, son difícilmente reconocibles. Los ***outliers*** son muestras de datos que difieren de la norma, haciendo que el modelo generalice mal o tarde más en hacerlo. Aunque estas muestras puedan empeorar el resultado del modelo, también pueden servir como fuente de extracción de datos específicos de anormalidades. A continuación se muestran algunos *outliers* del subconjunto.



(a) Carácter に **(b)** *Outlier* に **(c)** Carácter み **(d)** *Outlier* み

Figura 3.29. *Outliers* de ETL-4 *Character Database*

En las dos primeras pruebas del conjunto, los *outliers* no se consideran para el entrenamiento de la red, dejando al modelo con un total de 32.660 imágenes: 25.300 para entrenamiento, 3.680 para validación y 3.680 para prueba. Tomando los mismos parámetros expuestos en el experimento 0036-KATA-VGG16-0007, el resultado obtenido en la segunda prueba 0039-HIRA-VGG16-0002 es de un **99,53 % de acierto** (Figura 3.30).

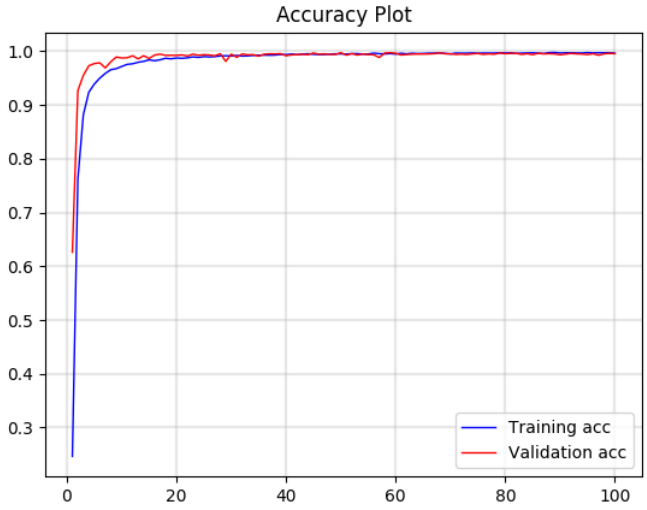


Figura 3.30. Experimento 0039-HIRA-VGG16-0002

No se aprecia un gran cambio con respecto a las pruebas de *katakana*: rápida convergencia del modelo y sin causar *overfitting*. También se alcanza una tasa de acierto muy alta para ser la primera prueba relevante, pero una vez más, al estar balanceadas las muestras, el resultado es válido. Para poder contrastar realmente la relevancia de los *outliers*, se añaden las muestras del ETL-4 *Character Database* al conjunto de *hiragana* para la próxima prueba. El experimento 0040-HIRA-VGG16-0003 cuenta con un total de 38.180 muestras: 30.360 imágenes en entrenamiento, 3.910 en validación y 3.910 en prueba.

Se obtiene un total de **99,18 % de acierto** (Figura 3.31), teniendo el modelo el mismo comportamiento que todos los experimentos vistos hasta ahora.

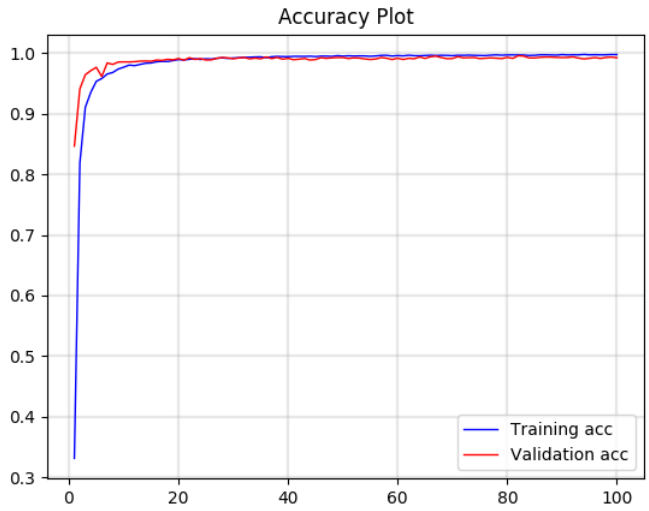


Figura 3.31. Experimento 0040-HIRA-VGG16-0003

Pese al resultado, se nota una diferencia de 0,4 % entre el modelo sin *outliers* y con *outliers*, casi el doble de diferencia que en los modelos de *katakana*. Para ver si la tendencia de +0,4 % se mantiene, se repiten ambos experimentos siete veces más.

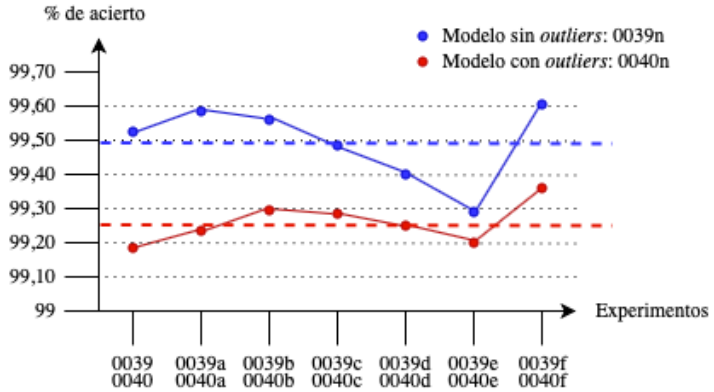


Figura 3.32. Tendencia entre modelos de *hiragana*

Como se observa en la figura, pese al descenso del experimento 0039e, la tendencia se mantiene entre los dos modelos, predominando el conjunto sin *outliers*. Por lo tanto, para la realización de la experimentación final, se escoge el conjunto de *hiragana* sin *outliers* ya que obtiene un mejor resultado de predicción.

3.4.3.3 Modelos Finales

Como última experimentación, se realizan diez experimentos más para el conjunto de *katakana*, con 50.963 imágenes en entrenamiento, 6.900 en validación y 6.900 en prueba, y para *hiragana*, con 25.300 imágenes en entrenamiento, 3.680 en validación y 3.680 en prueba. En estas pruebas se varían únicamente las bases convolucionales de los modelos y las dimensiones de las muestras para intentar mejorar el acierto.

Esta experimentación comprende las pruebas 0039 a 0059. Para poder observar las estructuras utilizadas en cada una de ellas, se crea la Tabla 3.3, añadiendo también el nombre del experimento junto con la tasa de acierto obtenida.

Los resultados para ambos conjuntos, como se esperaba, no varían significativamente. Aún así, se observa que los modelos que tienen menos cantidad de capas convolucionales, es decir, menos capacidad de filtrar información de las muestras, obtienen un peor resultado que las que tienen una base convolucional mayor. En su conjunto, los resultados obtenidos tanto para *hiragana* y para *katakana* son muy prometedores y estos han sido validados con el artículo de Charlie Tsai "Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks" [49].

Estructuras de los modelos finales										
64×64	32×32	32×32	32×32	32×32	64×64	32×32	32×32	64×64		
Conv2D(32)	Conv2D(32)	Conv2D(32)	Conv2D(16)	Conv2D(32)	Conv2D(32)	Conv2D(16)	Conv2D(32)	Conv2D(32)		
Conv2D(32)	Conv2D(32)	MaxPooling2D	Conv2D(32)	Conv2D(32)	Conv2D(32)	Conv2D(16)	Conv2D(32)	Conv2D(32)		
MaxPooling2D	MaxPooling2D	Conv2D(64)	Conv2D(32)	MaxPooling2D	MaxPooling2D	MaxPooling2D	Conv2D(32)	Conv2D(32)		
Conv2D(64)	Conv2D(64)	MaxPooling2D	MaxPooling2D	Conv2D(64)	Conv2D(64)	Conv2D(32)	MaxPooling2D	MaxPooling2D		
Conv2D(64)	Conv2D(64)	Conv2D(128)	Conv2D(64)	MaxPooling2D	Conv2D(64)	Conv2D(32)	Conv2D(64)	Conv2D(64)		
MaxPooling2D	MaxPooling2D	MaxPooling2D	Conv2D(128)	MaxPooling2D	MaxPooling2D	MaxPooling2D	Conv2D(64)	Conv2D(64)		
Conv2D(128)	Conv2D(128)	Conv2D(128)	MaxPooling2D	Conv2D(128)	Conv2D(128)	MaxPooling2D	Conv2D(64)	Conv2D(64)		
Conv2D(128)	Conv2D(128)	MaxPooling2D		MaxPooling2D	MaxPooling2D	MaxPooling2D	MaxPooling2D	MaxPooling2D		
Conv2D(128)	Conv2D(128)					Conv2D(128)	Conv2D(128)	Conv2D(128)		
MaxPooling2D	MaxPooling2D					MaxPooling2D	Conv2D(128)	Conv2D(128)		
							MaxPooling2D	MaxPooling2D		
Flatten										
Dense(1024)										
Dense(1024)										
Dense(46)										
Softmax										
0042-KATA	0043-KATA	0052-KATA	0053-KATA	0054-KATA	0055-KATA	0056-KATA	0057-KATA	0058-KATA	0059-KATA	
99,21 %	99,18 %	98,98 %	98,65 %	99,11 %	99,10 %	99,17 %	98,99 %	98,80 %	99,23 %	
0039-HIRA	0041-HIRA	0044-HIRA	0045-HIRA	0046-HIRA	0047-HIRA	0048-HIRA	0049-HIRA	0050-HIRA	0051-HIRA	
99,53 %	99,42 %	99,10 %	99,18 %	99,40 %	99,51 %	99,29 %	99,37 %	99,34 %	99,40 %	

Tabla 3.3. Experimentos finales de *katakana* y *hiragana*

3.5. Conclusiones de la experimentación

Se puede concluir esta gran experimentación con los grandes resultados obtenidos en los conjuntos de *hiragana* y *katakana* con pocos experimentos gracias a la experimentación previa realizada sobre los conjuntos *Dogs vs. Cats*, FERC y CIFAR10. Con todas estas pruebas se han demostrado varios puntos:

- El uso del ***data augmentation*** en las redes de neuronas convolucionales aumentan su capacidad de generalización de manera categórica, reafirmándose como uno de los pasos fundamentales a seguir para alcanzar resultados competitivos (siempre que su uso sea razonable).
- La importancia de la **división aleatoria del conjunto en entrenamiento, validación y prueba**. Esta separación de las muestras es uno de los pasos fundamentales en este tipo de experimentación, ya que permite al modelo entrenar y evaluarse de manera correcta y sin trampas ni resultados falseados.
- **Aumentar el número de imágenes en el conjunto de entrenamiento** supone una gran mejora. Este aumento ayuda a la capacidad de generalización de la red de neuronas gracias a que se entrena con un gran número de muestras distintas.
- La importancia de un **buen preprocesamiento** de datos: si los datos no son correctos o no están bien balanceados entre clases, el modelo se puede comportar de manera errática, la supervisión de la dimensiones de las imágenes, la identificación de *outliers*...
- La importancia de la implementación de **métricas auxiliares de evaluación del modelo**, como la matriz de confusión, que permitan al investigador extraer información adicional sobre el modelo e identificar comportamientos anómalos de la red.
- La importancia de la **elección del optimizador de gradiente y la tasa de aprendizaje** de la red. Puede que un optimizador ayude de manera más óptima al gradiente a encontrar el mínimo de la función de error, pero todo depende de cómo se trate la tasa de aprendizaje.
- Y, por último, la elección de una **buena metodología** para realizar los experimentos es fundamental para tener todos los datos y conclusiones extraídas organizadas correctamente, quedando demostrado en estas secciones.

Con los tan buenos resultados obtenidos en los conjuntos de *hiragana* y *katakana* y con la poca variación de estos, la experimentación se considera terminada. Por obtener los mejores resultados dentro de cada conjunto, **se escogen el modelo 0059-KATA-CCCMCCCMCCM-0018 (99,23 %) y el 0039f-HIRA-VGG16-0002 (99,61 %) para la implementación de la aplicación *web***.

4. APLICACIÓN *WEB*

En esta sección se explicará el objetivo principal de la *web*, así como las herramientas utilizadas para su desarrollo. Además, se hará un recorrido ilustrativo por la *web* para explicar su aspecto y para que sirva de guía al usuario. Por último, se comentará la evaluación realizada de la aplicación con usuarios reales.

4.1. Herramientas

4.1.1. *Framework* para *front-end*

Para la creación de la interfaz de la aplicación *web* se han estudiado dos opciones: **Angular** y **React**, ambas basadas en Javascript y apoyadas mayoritariamente por Node.js. Estas opciones son dos de las más buscadas en Google en los últimos 6 años debido a su capacidad y a su gran éxito en entornos empresariales.

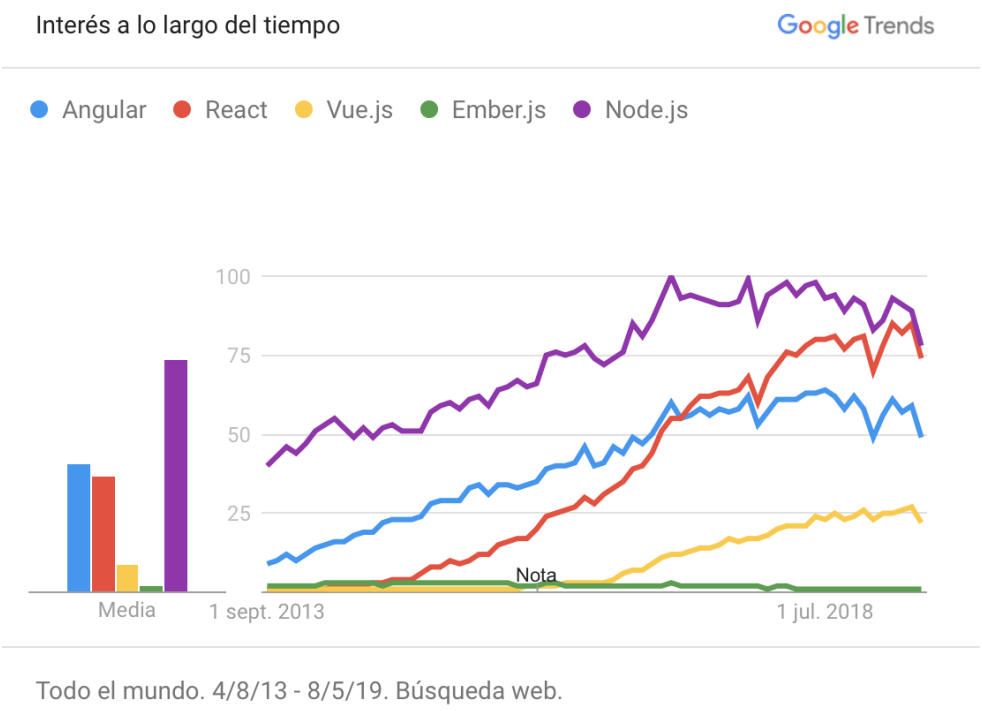


Figura 4.1. Tendencia de interés entre *frameworks* de Javascript

Angular es un *framework* completo basado en Typescript, una versión mejorada de Javascript, que ofrece la capacidad de crear aplicaciones móviles o páginas *web* complejas modernas [50]. Este *framework* tiene un magnífico soporte, tanto por parte de su comunidad como por parte de los propios desarrolladores. La jerarquía de archivos para crear páginas se basa en componentes y cada cual tiene un archivo `html`,

`css` y `ts`. Esta diferenciación de archivos hace que programar el estilo (`html` y `css`) y la funcionalidad de la página (`ts`) sea muy cómodo ya que se diferencian completamente. Además, el uso de paquetes auxiliares para ayudar en el desarrollo de la aplicación es un elemento clave de Angular. Hay miles de paquetes y herramientas de apoyo gratuitos disponibles y para distintos fines como por ejemplo estilización, mejora de gráficos, soporte para otras herramientas...

Uno de esas herramientas es **Ionic**. Ionic es un *framework* que está destinado, principalmente, a la experiencia de usuario o UX de páginas *web* y aplicaciones móviles de manera limpia, sencilla y acorde a los protocolos estándar de UX [51]. A pesar de ello, ofrece alternativas a Angular para algunas funcionalidades como el enrutamiento o redirecciones por `url`. Ionic está actualmente desarrollado para Angular, aunque se está expandiendo.

Como herramienta alternativa se considera React [52]. Esta herramienta no es un *framework* en su totalidad como Angular, si no que es una librería complementaria destinada únicamente a la interfaz o vista de la *web* o aplicación. Angular, al ser un *framework* completo ofrece toda la funcionalidad que pueda requerir una aplicación compleja, pero React solo ofrece soporte para la interfaz. Por esto, al utilizar React, se deben elegir las librerías que más se ajusten a la necesidades y funcionalidades de la aplicación, haciendo que se dependa mucho de terceros, ofreciendo, en contra, una alta flexibilidad a la hora de elegir qué herramienta utilizar.

React utiliza una extensión de Javascript: `JSX` y no Typescript como en Angular. Esta extensión del lenguaje permite combinar `html` plano en los archivos de Javascript, creando archivos conteniendo tanto el estilo como la funcionalidad de la vista.

En conclusión, el hecho de que Angular ofrezca un "todo en uno" en cuanto a la funcionalidad y estilo de una página *web* es más *seguro* y *rentable* a largo plazo que depender de librerías externas como en React. Esto, por supuesto, repercute en la flexibilidad del *framework*. Además, como preferencia personal, la separación entre funcionalidad y estilo de la vista de Angular me parece más apropiada para aplicaciones, ya que está todo más ordenado y es más claro que con `JSX`. Por último, añadir que Angular e Ionic ya se habían usado para la creación de una aplicación en un trabajo previo, por lo que se conocía bien como trabajar con estas herramientas y de lo que eran capaces de ofrecer. Por esto, Angular e Ionic son las herramientas elegidas para el desarrollo de la aplicación.

4.1.2. Herramientas para *back-end*

Como herramientas para el *back-end* o lógica de la aplicación *web*, se han tenido en cuenta 2 herramientas fundamentales: **Firebase** y **Tensorflow.js**.

Para poder evaluar imágenes con caracteres japoneses en la *web*, hace falta integrar los dos modelos creados para el reconocimiento de caracteres de *hiragana* y *katakana*. Para ello se utiliza **tensorflow-converter**, que transforma los modelos de Python a archivos JSON, tratables en Typescript. Este conversor está incluido dentro de Tensorflow.js [53], una adaptación del Tensorflow original a Javascript. Ya que se ha trabajado con Keras, y Angular funciona mediante Typescript, el uso de Tensorflow.js para hacer predicciones con los modelos creados resulta esencial.

Adicionalmente, para poder administrar usuarios, sus datos y para alojar la aplicación en servidores de manera gratuita se utiliza Firebase [54]. Firebase es una plataforma en la nube de servicios *web* que pertenece a Google. Esta plataforma, junto con Angular, ofrecen unas capacidades al desarrollador potentes, totalmente gratuitas, seguras y con una buena comunidad y documentación.

4.1.3. Versiones

En un momento del desarrollo de la *web* se utiliza Angular 5 e Ionic 3 como versiones de *framework* pero, debido a un problema con la navegación de la aplicación, el desarrollo es interrumpido. Tras estudiar el suceso, se concluye que es necesaria una actualización completa de las herramientas: a Angular 7 e Ionic 4. Con esta actualización se resuelve el problema de la navegación en la aplicación, y el desarrollo de la misma queda desbloqueado de nuevo.

Herramienta	Versión
Angular	7.2.2
Ionic	4.1.0
Tensorflow.js	1.0.2
Firebase CLI	6.3.0

Tabla 4.1. Versiones de las herramientas para la aplicación *web*

4.2. Desarrollo de ¡Aprende! Japonés

4.2.1. Objetivo principal y descripción general

El objetivo principal de la aplicación *web* es crear un nuevo paradigma de enseñanza de escritura japonesa mediante el uso de la inteligencia artificial y, más concretamente, del uso de redes de neuronas convolucionales para reconocer caracteres japoneses escritos con cualquier estilo caligráfico en imágenes. En consecuencia, la *web* se llama **¡Aprende! Japonés**.

La *web* no se centra en la enseñanza de todos los caracteres japoneses, sino en los básicos, los contenidos en los silabarios *hiragana* y *katakana*. Por lo tanto, juntando los modelos de reconocimiento de caracteres de ambos silabarios y las capacidades de Angular, la premisa principal de la aplicación son los dibujos de los caracteres japoneses. Por ello, y para que tenga un enfoque más didáctico, la *web* consta de dos secciones diferenciadas: una dedicada a la práctica de los caracteres y otra dedicada a pequeños tests para que el usuario demuestre lo aprendido en la práctica.

- En la **práctica**, el usuario puede dibujar con su estilo caligráfico los 92 caracteres japoneses en la manera que desee, seleccionando uno u otro silabario, además del carácter que quiera dibujar. También se le ofrece información adicional sobre el carácter como el número de trazos, pronunciación y un pequeño vídeo mostrando cómo se debería de realizar el trazado idóneo del carácter en cuestión.
- En los **tests**, el usuario es preguntado por el carácter dibujado en base a su pronunciación. Con esto, el usuario consigue una asociación directa entre pronunciación y carácter, mostrando un aprendizaje correcto.

Todos los dibujos realizados en la aplicación se "auto-clasifican" y almacenan creando un *corpus* o un conjunto de datos propio de caracteres básicos japoneses que se ofrecerá gratuitamente a la comunidad científica para su uso en investigación.

4.2.2. Manual de usuario

Una vez definida la aplicación de manera general y después de haber definido el objetivo principal de la misma, se procede a explicar el manual de uso de la aplicación *web*. **Toda la aplicación *web* es responsiva**: se adapta al tamaño de la pantalla según el aparato que la muestre. En esta sección se decide exponer la

visualización de la *web* por medio de un iPhone 8.

Al entrar en la *web*, se muestra la **página de registro** (Figura 4.2). En esta página, el usuario podrá realizar tres operaciones: registrarse, si no tiene cuenta ya; iniciar sesión y practicar directamente los caracteres japoneses sin necesidad de registro. Debido a que son muchos los usuarios propensos a dudar del uso de datos en las páginas *web*, se habilita la sección de prácticas sin registro previo y, a su vez, se crea un botón auxiliar para informar qué tratamiento se le dará a los datos introducidos (correo electrónico y nombre).

Además de esto, la aplicación está disponible tanto en español como inglés y, por ello, hay una opción para cambiar el idioma en esta página. Esto solo cambia los textos germánicos y románicos, los textos suplementarios en japonés de la aplicación se mantienen.

Figura 4.2. Página de registro de ¡Aprende! Japonés

Una vez un usuario se registra o inicia sesión, se visualiza la **página principal** de la aplicación (Figura 4.3). En esta página se muestran toda la información relativa

a todos los tests realizados en unos gráficos y se puede acceder tanto a la sección de prácticas como a la de test. Si es la primera vez que el usuario entra en la aplicación, un tutorial de uso se abrirá automáticamente para guiarle por la página, pero independientemente de si es un usuario nuevo o no, el tutorial siempre queda disponible en las páginas habilitadas.

Por una parte, en los gráficos se exhiben: el número de tests realizados con ambos silabarios (para llevar un estudio equilibrado) y el número total de respuestas correctas y erróneas separados por silabarios. De esta manera, el usuario puede observar en qué silabario necesita más refuerzo y dedicación. Y, por otra parte, y como detalle, antes de entrar en la sección de los tests, el usuario deberá elegir qué silabario escoger para la realización del test.

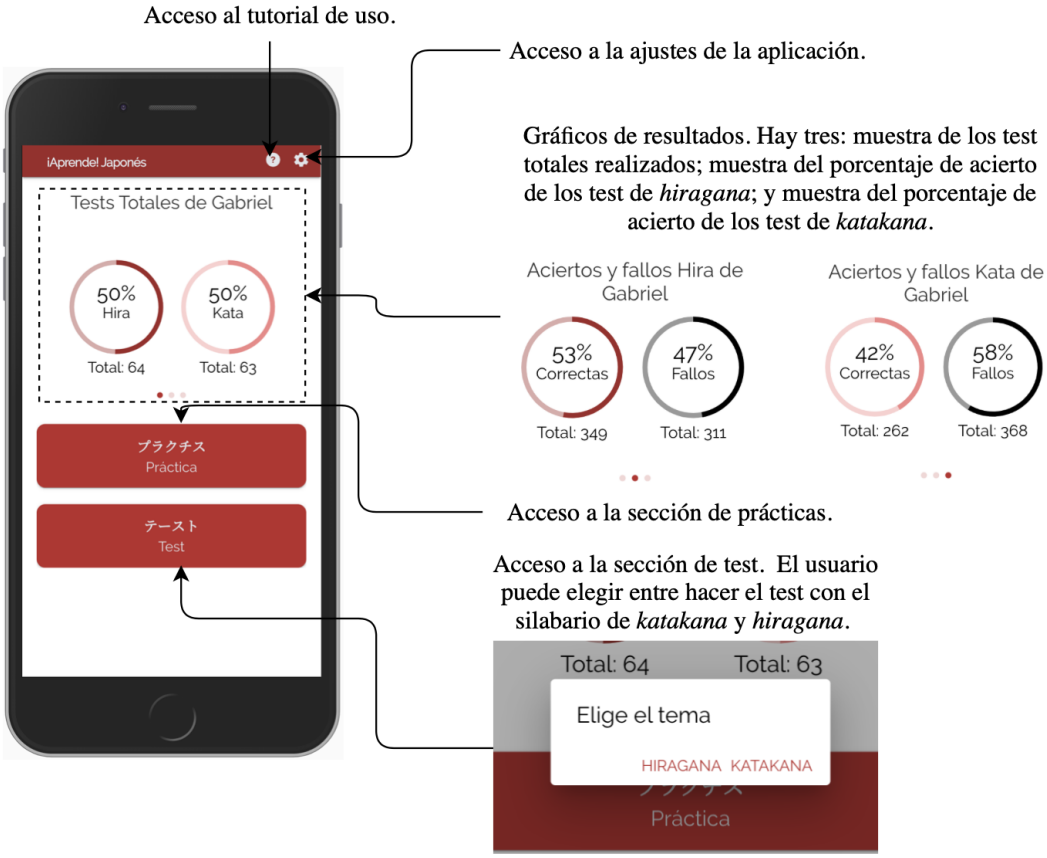


Figura 4.3. Página principal de ¡Aprende! Japonés

Únicamente desde la página principal se puede acceder a los ajustes de la aplicación, entrando en la **página de ajustes** (Figura 4.4). En esta página, el usuario podrá administrar su cuenta (cerrando sesión o eliminando su cuenta, y consecuentemente sus datos) y modificar los datos utilizados en el registro de la aplicación: el nombre

y la contraseña. Además de esto, es en esta página donde se le permite cambiar de nuevo el idioma de la aplicación. Para cambiar la contraseña o eliminar la cuenta, al tratarse de acciones comprometedoras, se requiere de una verificación previa del usuario para llevarse a cabo.

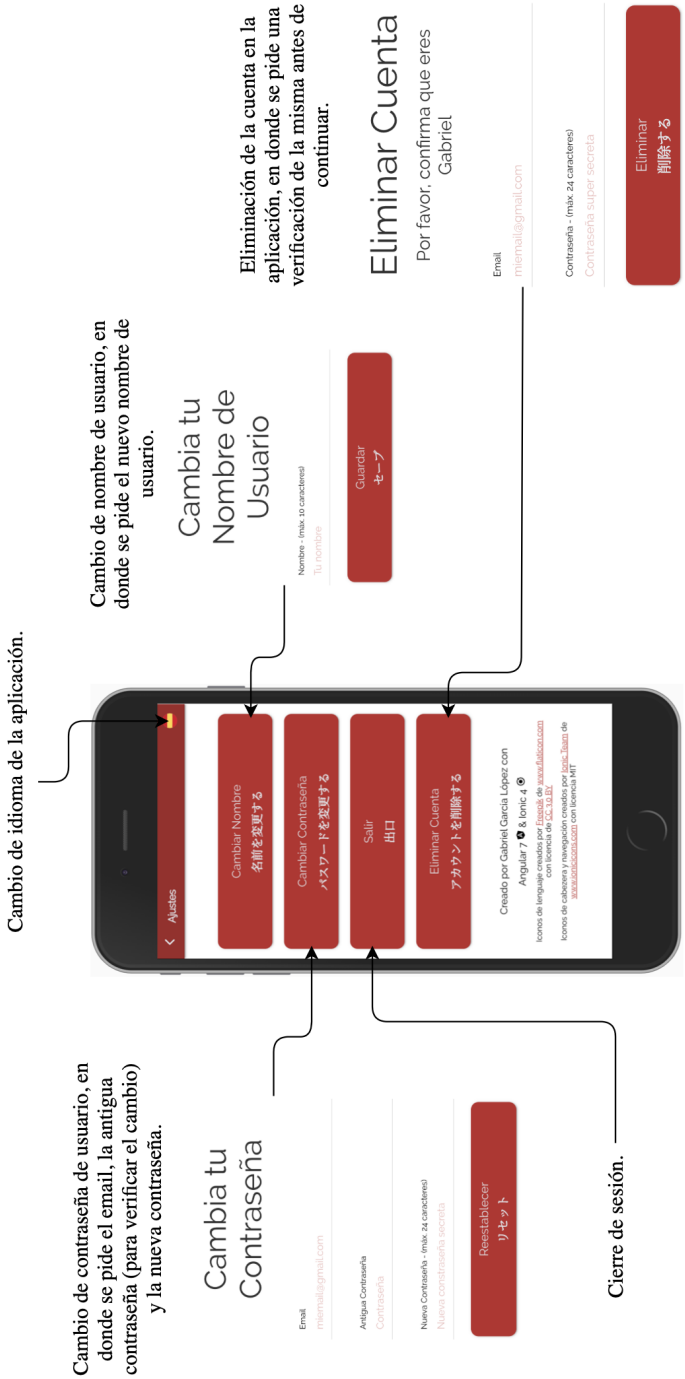


Figura 4.4. Página de ajustes de ¡Aprende! Japonés

Volviendo a la página principal, se accede a la **página de prácticas** (Figura 4.5). En esta página, los modelos de predicción elegidos en la Sección 3.4.3 toman parte.

Aquí, el usuario podrá elegir el carácter del silabario que desee para aprender a dibujarlo (usando su propio estilo caligráfico) y, utilizando los modelos de *hiragana* y *katakana*, reconocer el carácter en el dibujo y mostrar si este es válido o no.

En esta página, debido a la gran cantidad de opciones disponibles, se habilita un tutorial de uso para el usuario. El carácter elegido se muestra en grande para que el usuario lo vea bien y para que sea capaz de reproducirlo correctamente en el lienzo. Este se muestra junto con información útil para comprenderlo y para que el usuario se familiarice con su pronunciación y trazado idóneo.

Una vez el usuario dibuja el carácter, este pasa a evaluarse en los modelos de *hiragana* o *katakana* (según el silabario escogido, en el caso de la Figura 4.5: *hiragana*). La predicción se realiza con la imagen de lo dibujado en el lienzo, obteniendo una probabilidad de que en el dibujo se represente el carácter seleccionado. Acorde a esa predicción, si coincide con el carácter seleccionado, el carácter dibujado será aceptado, mientras que si no coincide, será rechazado. En ambos casos, se le informa al usuario de cuál ha sido el carácter que el modelo a reconocido en el dibujo realizado como información adicional.

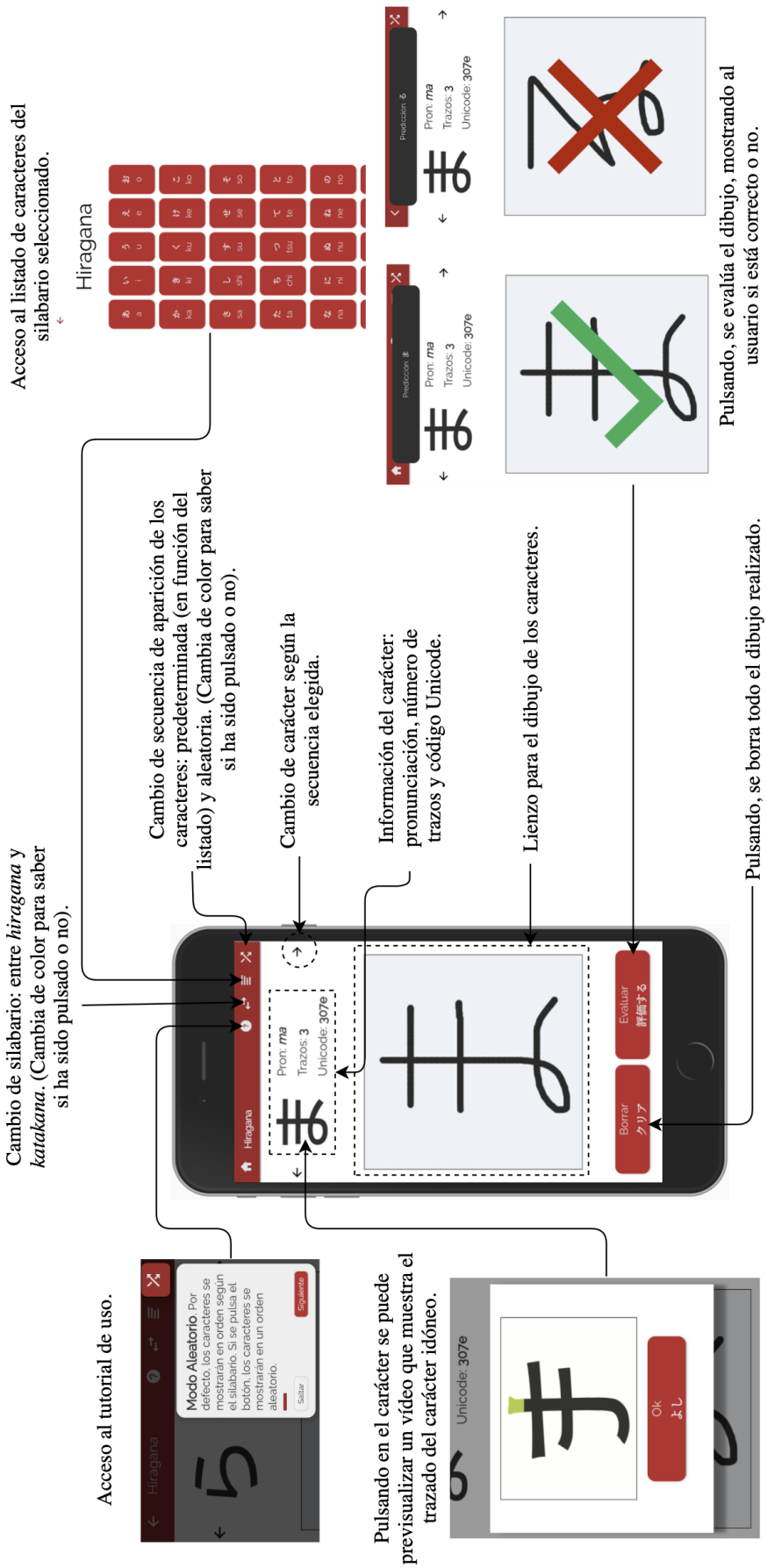


Figura 4.5. Página de prácticas de ¡Aprende! Japonés

Para demostrar que los modelos funcionan correctamente y que la aplicación *web* permite el reconocimiento de cualquier estilo caligráfico, se presentan las Figuras 4.6 y 4.7, mostrando el carácter み de *hiragana* y テ de *katakana* con diferente caligrafía. Además de estos ejemplos, se han probado todos los caracteres de ambos silabarios con varios estilos caligráficos y los resultados concuerdan con los obtenidos en la experimentación.

(a) Caligrafía 1 **(b)** Caligrafía 2 **(c)** Caligrafía 3

Figura 4.6. Comprobación de diferentes estilos caligráficos para *hiragana*

(a) Caligrafía 1 **(b)** Caligrafía 2 **(c)** Caligrafía 3

Figura 4.7. Comprobación de diferentes estilos caligráficos para *katakana*

Tras haber visitado la página de prácticas, se vuelve a la página principal y, esta vez, se accede a la **página de test** (Figura 4.8). Como se ha explicado antes, esta consiste de 10 cuestiones en las que al usuario se le pregunta por caracteres aleatorios en base a su pronunciación. El usuario no puede volver a preguntas anteriores por lo que debe dibujar los caracteres de manera secuencial hasta finalizar el test. En

cualquier caso, al usuario se le permite salir de este, perdiendo así, todo el progreso.

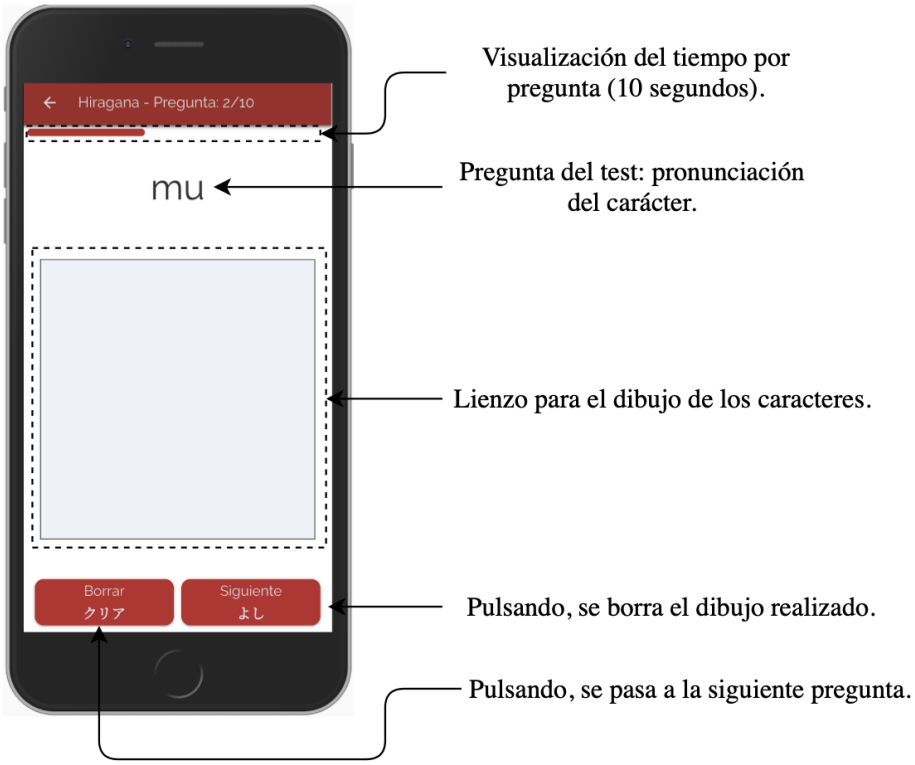


Figura 4.8. Página de test de ¡Aprende! Japonés

En ningún momento del test se le da al usuario ninguna pista de si las respuestas son correctas o no. Pese a ello, cada vez que se pasa a la siguiente pregunta, se realiza una predicción sobre el dibujo que haya en el lienzo. Este *feedback* se da en la **página de resultados** (Figura 4.9), que representa las predicciones realizadas por el modelo durante el test. En esta página se muestran todos los caracteres dibujados junto con la pregunta (la pronunciación) y su respuesta correcta, para que el usuario revise los posibles errores cometidos. Una vez revisados los errores, el usuario puede tomar dos decisiones: hacer otro test sin guardar el resultado del revisado o guardar el resultado y volver a la página principal.

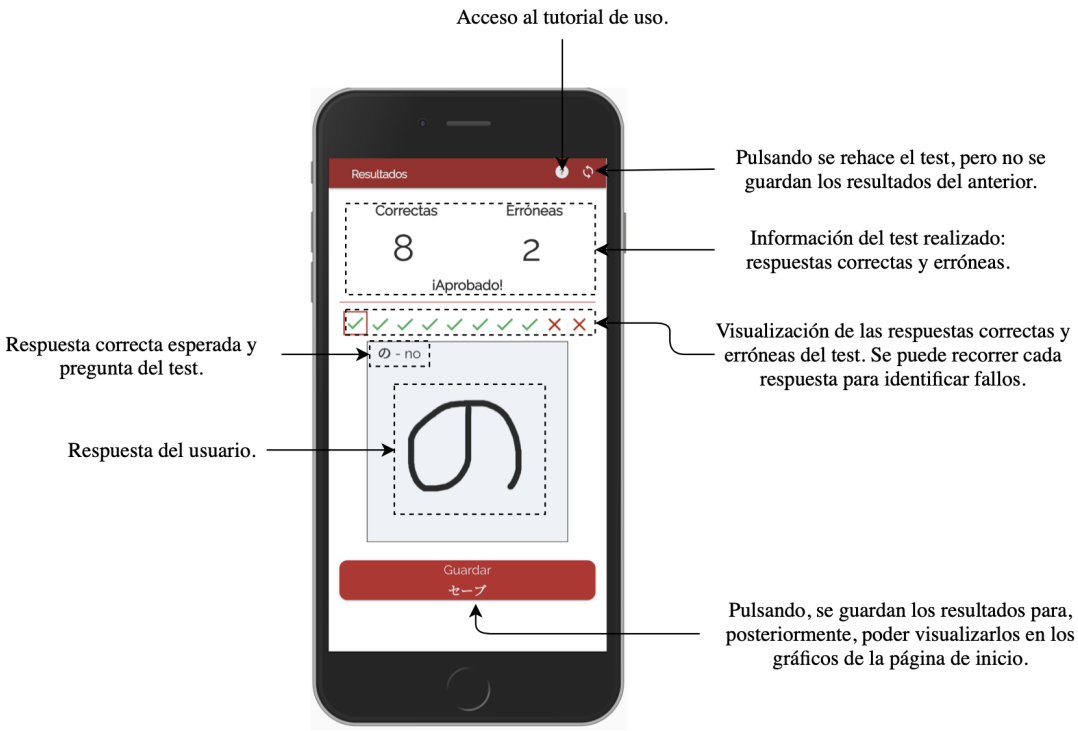


Figura 4.9. Página de resultados de ¡Aprende! Japonés

Con esta última página, la aplicación *web* queda totalmente explicada y desglosada, consiguiendo el objetivo principal del trabajo: crear una aplicación capaz de enseñar a escribir japonés básico mediante técnicas de inteligencia artificial capaces de reconocer caracteres con cualquier estilo caligráfico.

Como información adicional, en el **Anexo III** se detallan de manera técnica los requisitos de ¡Aprende! Japonés, mostrados implícitamente durante toda la Sección 4.2.2, así como los casos de prueba para verificar los propios requisitos.

4.3. Evaluación con usuarios reales

Por último, en esta sección se realiza un cuestionario anónimo para evaluar la aplicación en función de su experiencia con ella. Para ello se ha utilizado la herramienta de Cuestionarios Google, en donde se han realizado una serie de preguntas para determinar el interés por la aplicación además de una evaluación de la funcionalidad de esta.

La encuesta se ha realizado a 44 personas. Estas personas han sido escogidas al azar sin tener en cuenta si saben japonés o no y de un rango de edades lo más amplio posible: de 20 a 60 años. La primera sección de la encuesta se centra en un **breve**

estudio de mercado. Un 54,6 % de los encuestados ha utilizado previamente aplicaciones de aprendizaje de idiomas. Entre estas personas, la mayoría ha utilizado *Duolingo* y *Babbel* para el inglés y el francés entre otros.

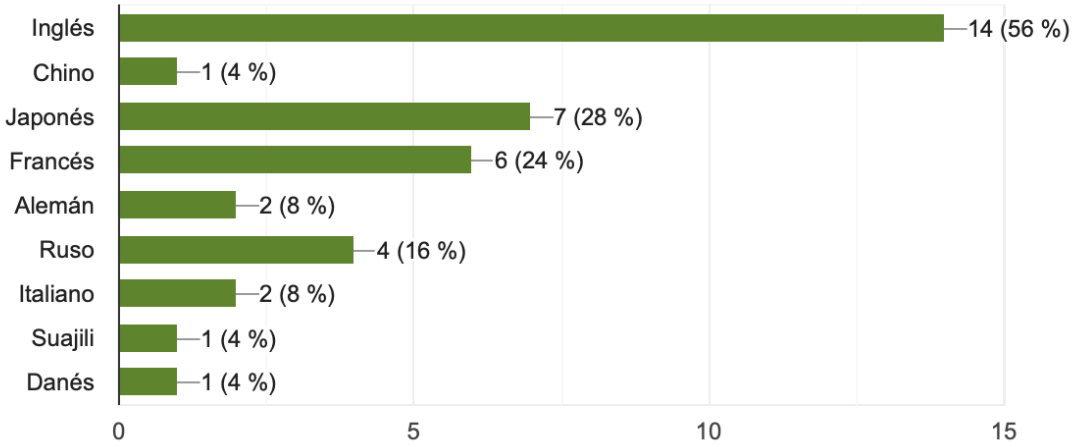


Figura 4.10. Idiomas estudiados por los encuestados

Las aplicaciones mencionadas son las más populares hoy en día y no hay sorpresa alguna en la elección de los encuestados. Al igual pasa con los idiomas. Lanzar una aplicación dedicada a la escritura únicamente y para un idioma oriental a un mercado tan monopolizado puede ser arriesgado, pero cuando a los encuestados se les pregunta por si en dichas aplicaciones había ejercicios de escritura se observa un dato curioso: un 32 % mantiene que no y un 44 % dice que sí, el resto no lo sabe. Esto puede desconcertar en un principio, ya que *Duolingo* y *Babbel* siguen el mismo patrón de aprendizaje que *LingoDeer*. Por lo que, los encuestados podrían haber mal interpretado la pregunta en cuestión, identificando como ejercicios de escritura, no los ejercicios en los que el usuario dibuja el carácter, sino los ejercicios de asociación y similares.

La segunda sección de la encuesta consiste en averiguar si existe una **necesidad** de una aplicación *web* como la que aquí se presenta. Un 84,6 % de los encuestados se muestran propensos a estudiar y aprender un nuevo idioma, los que más destacan son el japonés y el chino. Pero a la hora de elegir la forma en la que se quiere aprender a escribir un nuevo idioma oriental, la mayoría elige un profesor nativo (61,4 %) frente a una *web* (15,9 %) o una academia (22,7 %), véase Figura 4.11.

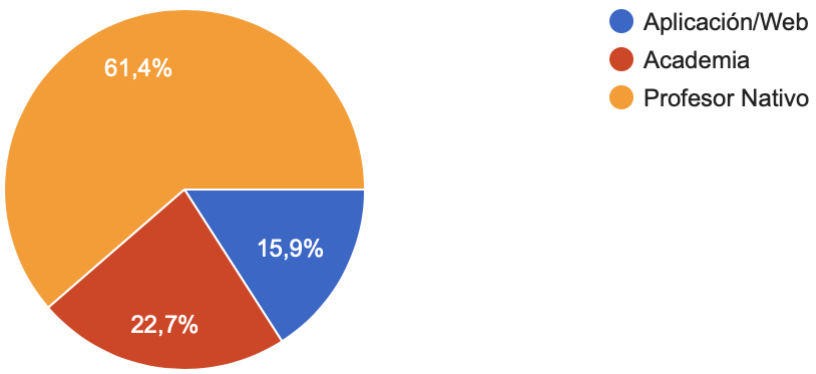


Figura 4.11. Proporción entre métodos didácticos

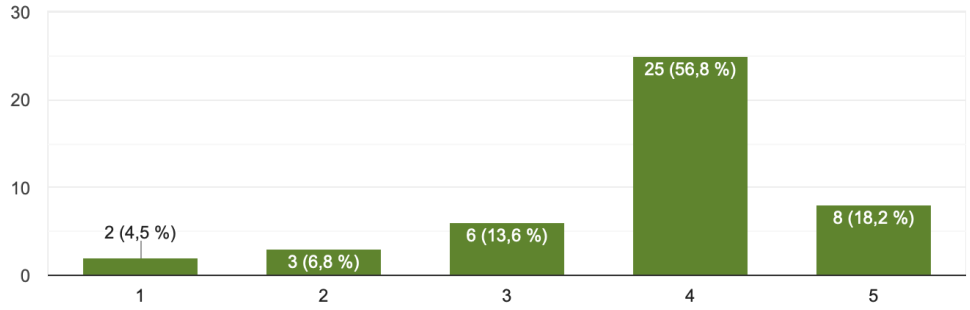
Esto es lógico, ya que un profesor nativo es la mejor opción para aprender cualquier idioma, pero este aprendizaje no es gratuito y tampoco lo es en las academias. Por esto, se cree que la intención de los encuestados es la de aprender a escribir con un profesor nativo o academia, pero no todos ellos podrían estar dispuestos a pagar por aprender a escribir. Posiblemente se decanten más por una alternativa gratuita e igual de buena que un profesor nativo y, además, no se contempla la disponibilidad de la aplicación a cualquier hora y lugar, frente al horario de un profesor o academia.

En la siguiente cuestión se pregunta a los encuestados si les gustaría aprender a escribir en una aplicación *web* mediante plantillas correctoras o con su propia caligrafía. En esta pregunta clave, por sorpresa, se encuentra una equidad en cuanto a respuestas: un 34,1 % prefiere aprender con plantillas, un 27,3 % prefiere con su propia caligrafía y un 38,6 % le da igual con tal de aprender. Esto demuestra que a la gente no le importa aprender a escribir de un modo u otro con tal de hacerlo. Puede que haya más gente que se decante por el uso de plantillas pero para compensar esa falta de usuarios potenciales, existe una parte casi proporcional y equivalente de usuarios que les gustaría aprender con su propia grafía.

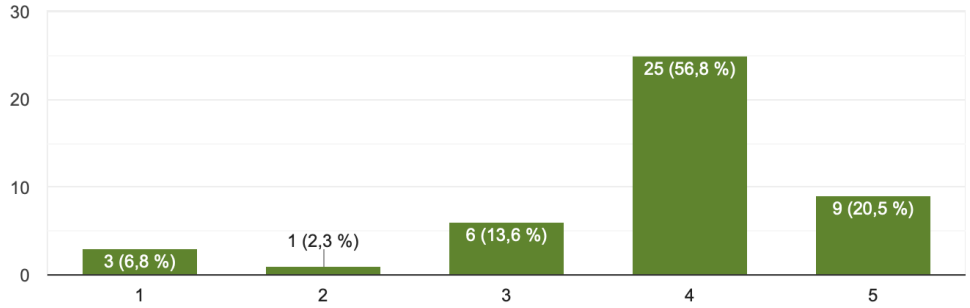
La última fase de la encuesta es en la que a los usuarios se les invita a **probar la aplicación** creada. Lo primero que hacen la mayoría de usuarios (56,8 %) es utilizar la página de pruebas sin registro previo, frente a un 27,3 % que sí se registra. Esto se puede deber a que la gente cada vez es menos propensa a dar sus datos en internet debido a los últimos escándalos con Facebook y similares. Por lo tanto, la mayoría de usuarios ha probado la aplicación únicamente en el ámbito de las prácticas. Independientemente de esto, la aplicación ha sido un éxito en cuanto a intuición con un 79,6 % de los encuestados satisfechos con el fácil manejo y accesibilidad de esta. Este es un dato a resaltar ya que los tutoriales que hay implementados en la página *web*

que guían al usuario para un entendimiento total de la aplicación, han sido obviados por casi un 40,9 % de encuestados.

En cuanto al tiempo de respuesta de la predicción del modelo y al acierto que demuestra este en sus predicciones, los encuestados han quedado bastante satisfechos.



(a) Tiempo de respuesta de predicción



(b) Satisfacción de los resultados obtenidos

Figura 4.12. Evaluaciones del modelo de predicción

Siendo una escala del 1 al 5 en donde 1 es muy poco satisfecho y 5 muy satisfecho. Otra dato a resaltar es que casi un 85 % de los encuestados estarían interesados/as en adaptar la aplicación a otros idiomas como el ruso, chino o tailandés. Esto puede abrir las puertas a un futuro próximo de la aplicación. Por último, y no menos importante, es que a un 75 % de los encuestados les ha resultado correcto y adecuado el aprendizaje que se da en la aplicación *web* de la escritura japonesa, aunque a un 11.4 % también les ha resultado interesante el método de aprendizaje pero prefieren que se enseñe de una manera alternativa: con el uso de plantillas o añadiendo ejercicios de asociación. Al resto de encuestados no les ha servido para nada la aplicación ya que no tienen interés alguno en escribir japonés o porque simplemente no les gusta (Figura 4.13).

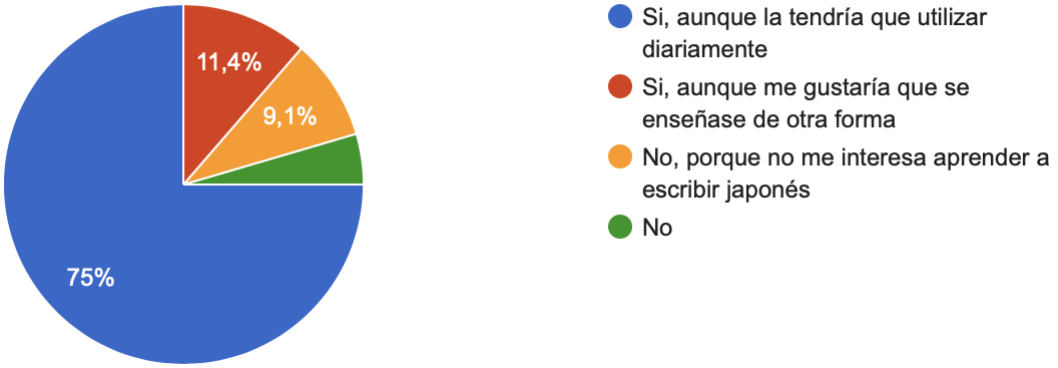


Figura 4.13. Conclusión final: ¿es útil la aplicación?

4.4. Conclusiones sobre la aplicación *web*

Como conclusión, la interfaz de usuario ha resultado ser intuitiva y manejable para los usuarios encuestados y ha cumplido su principal misión de ayudar a aprender a escribir japonés mediante una caligrafía libre y sin restricciones. La funcionalidad de la *web* ha sido revisada y pensada para que los usuarios se limiten a aprender a escribir japonés. Esto ha quedado demostrado con la gran acogida que ha tenido ¡Aprende! Japonés entre los encuestados, identificándola como una *web* intuitiva, fácil de usar y con la que se puede aprender a escribir japonés con un uso adecuado.

La integración de los modelos en la *web* ha resultado todo un éxito, verificándose con el uso diario de la aplicación por parte del desarrollador, además del uso de la misma por parte de los encuestados. Aparte, gracias a la encuesta, se ha podido recoger información acerca de lo que los potenciales usuarios de la aplicación quieren o les gustaría ver integrado en esta, como por ejemplo, la inclusión de nuevos idiomas como el ruso o el chino, y el añadido de más ejercicios para los tests. Este *feedback* es bueno para el desarrollador ya que recibe críticas reales que sirven para poder mejorar la aplicación *web* y, así, contentar a los usuarios habituales y motivar a los nuevos para su uso.

Con todo ello, se obtiene una visión prometedora y positiva sobre ¡Aprende! Japonés como nuevo paradigma para la enseñanza de la escritura japonesa.

5. MARCO REGULADOR

En esta sección se explicarán las cuestiones referentes a la legislación vigente en cuanto al conjunto de datos utilizado (ETL *Character Database*), la protección de datos de los usuarios de la aplicación *web* y a las licencias de las herramientas utilizadas.

5.1. Conjunto de datos

Las imágenes utilizadas han sido recogidas y administradas por el *Electrotechnical Laboratory* (ahora el *National Institute of Advanced Industrial Science and Technology* − AIST) en cooperación con el *Japan Electronic Industry Developement Association* (ahora el *Japan Electronics and Information Technology Industries Association*) [48]. Las imágenes no contienen ningún tipo de dato sensible al tratarse de caracteres básicos del lenguaje. El conjunto está disponible para la descarga en la página oficial bajo una aprobación previa del AIST. La principal condición para el uso gratuito del conjunto es el uso de este en investigaciones que no hagan un uso comercial de él. Ya que en este trabajo se quiere realizar una aplicación para enseñar a escribir japonés, se deberá negociar con el AIST las condiciones legales y comerciales.

5.2. Protección de datos

Dentro de la aplicación, los datos comprometedores son el correo electrónico, la contraseña del usuario, las estadísticas de los tests y las imágenes de los caracteres dibujados realizadas. Como se ha explicado en la sección anterior, la administración de los usuarios se realiza mediante Firebase, concretamente mediante Firebase *Authentication*, para las estadísticas de los tests, se utiliza Firebase *Realtime Database* y para el almacenamiento de todas las imágenes de caracteres que se dibujen en la aplicación se utiliza Firebase *Storage*.

El uso de los datos de usuario confidenciales y personales se hace acorde a la *Ley Orgánica de Protección de Datos* [55]. Ya que se trata de una ley, se da por hecho que la plataforma de Firebase cumple con la legislación vigente y que hace un buen uso de los datos. Independientemente, al usuario se le informa en la aplicación *web* qué uso se le dan a todos sus datos en la misma y qué datos se le podrán extraer, como el almacenamiento de los dibujos.

5.3. Herramientas de *software*

Se han utilizado varias herramientas tanto para la creación de los modelos de predicción como para el desarrollo de la aplicación *web*. A continuación, se detallan las licencias de cada una de ellas.

- Python: El lenguaje de programación Python se distribuye como *software* de libre uso bajo la licencia GPL [56].
- Tensorflow y Tensorflow.js: Este producto de *software* está bajo la licencia de Apache 2.0 [57] que permite el desarrollo y uso de *software* gratuito de manera colaborativa.
- Angular, Ionic y Keras: Estas tres herramientas utilizan la licencia MIT [58], que permite la modificación, distribución y uso del *software*.
- Firebase: Se utiliza el plan *Spark* bajo la licencia de Apache 2.0. Este plan es gratis y viene con unas cuotas fijas de datos para todas las funciones disponibles.
- Herramientas auxiliares: Las herramientas de uso como GitHub, Trello, LaTeX, Google Drive, Git Kraken y Visual Studio son *software* libre bajo licencias de uso público. Algunas tienen planes de pago, para mejorar aún más la experiencia pero en este trabajo se utilizan los planes gratuitos.

6. ENTORNO SOCIOECONÓMICO

En esta sección se detalla la planificación del proyecto, así como el presupuesto establecido para la realización del trabajo. También se habla del impacto que podría tener la aplicación en el mercado y el impacto que podría conllevar en un futuro.

6.1. Planificación

El proyecto se ha realizado en cuatro etapas principales: decisión del **enfoque del trabajo**, la **creación de los modelos de predicción**, la **creación de la aplicación *web*** y la **elaboración de la memoria**.

La primera toma de contacto con los tutores y el proyecto se establece el 5 de septiembre de 2018. Desde ese día hasta el 12 de octubre se investigan varias alternativas para enfocar el trabajo en ellas como la aplicación del *deep learning* a reconocimiento de expresiones faciales. Este periodo de tiempo constituye la primera etapa del proyecto: se deciden e instalan las herramientas necesarias para el trabajo, se estudian las redes convolucionales y las diferentes alternativas de trabajo (conjuntos de datos para enfocar el proyecto). Una vez se han definido las herramientas a utilizar y se entiende el funcionamiento de las redes convolucionales, se realizan los experimentos con los conjuntos de enfoque (FERC, *Dogs vs. Cats* y CIFAR10) para aplicar los conocimientos extraídos del estudio. Es durante esta etapa cuando se modifica el enfoque del trabajo y se propone estudiar el reconocimiento de caracteres japoneses básicos mediante imágenes. Por lo tanto, el 8 de noviembre comienza la preparación del conjunto final, en paralelo a las pruebas de los demás conjuntos.

Tras la finalización de las pruebas de los conjuntos de enfoque, las pruebas del conjunto final elegido comienzan el 28 de noviembre, y estas finalizan el 22 de enero de 2019. Seguidamente, con los mejores modelos listos del conjunto de caracteres japoneses, se pone en marcha la tercera etapa de la planificación que consiste en la elaboración de la aplicación *web* ¡Aprende! Japonés para la enseñanza de la escritura japonesa. El desarrollo de la aplicación finaliza el 19 de mayo y, de forma paralela, se desarrolla la fase de pruebas y experimentación de la aplicación y se avanza con la memoria, desde el 1 de abril al 5 de junio.

Para una visualización completa y más detallada de toda la planificación se crea un **diagrama de Gantt** (véase Figura 6.1).

	Tarea	Duración Estimada	Inicio Estimado	Finalización Estimada
1	Ejecución del proyecto completo	9 meses	5/09/2018	5/06/2019
2	Enfoque del trabajo	37 días	5/09/2018	12/10/2018
3	Instalación de herramientas	15 días	5/09/2018	20/09/2018
4	Búsqueda de enfoques para el trabajo	*13 días*	12/09/2018	25/09/2018
5	Planificación de proyecto	1 día	25/09/2018	26/09/2018
6	Aprendizaje de Deep Learning con Keras y RNC	16 días	26/09/2018	12/10/2018
7	Experimentación previa	48 días	12/10/2018	28/11/2018
8	Experimentación de Gatos vs. Perros	23 días	12/10/2018	3/11/2018
9	Experimentación de FERC	*13 días*	30/10/2018	12/11/2018
10	Experimentación de CIFAR10	*19 días*	9/11/2018	28/11/2018
11	Experimentación con el conjunto ETL	75 días	8/11/2018	22/01/2019
12	Obtención y extracción del conjunto	8 días	8/11/2018	16/11/2018
13	Ordenación del conjunto en Hiragana y Katakana	12 días	16/11/2018	28/11/2018
14	Experimentación de Katakana	29 días	28/11/2018	27/12/2019
15	Experimentación de Hiragana	*29 días*	24/12/2019	22/01/2019
16	Desarrollo de la aplicación web	117 días	22/01/2019	19/05/2019
17	Desarrollo de interfaz	47 días	23/01/2019	20/03/2019
18	Desarrollo de funcionalidad	*56 días*	3/02/2019	1/04/2019
19	Evaluación de la aplicación y arreglo de errores	49 días	1/04/2019	19/05/2019
20	Desarrollo de la memoria	66 días	1/04/2019	5/06/2019

Figura 6.1. Diagrama de planificación de Gantt del proyecto

6.2. Presupuesto

Como todo proyecto, este también tiene un presupuesto que ha de tomarse en cuenta a la hora de la planificación. Aquí se van a presupuestar los gastos de *hardware*, *software*, de plantilla y otros gastos variados, para finalmente agregar todos ellos y dar un presupuesto final del proyecto. Nótese que se aplica el impuesto del valor añadido (IVA) para los diferentes productos: 21 % para el *hardware* y *software* y un IRPF del 19 %. Para los costes totales se recurre al cálculo del coste aplicado al uso que viene dado por: el uso y la vida útil de un material determinado para los costes de materiales, y por las horas útiles realizadas durante el proyecto para el coste del personal. En la Tabla 6.1 se recoge el presupuesto único para los materiales utilizados.

Útil	Coste bruto (€)	Tiempo de uso (meses)	Vida útil (meses)	Coste aplicado al uso (€)
Hardware				
Ordenador personal	1.386€	9	120	95,95€
Ordenador de experimentación	700€	3	120	17,5€
Teléfono personal	395€	3	36	32,91€
Coste total del *hardware*				**146,36€**
Software				
Angular	Gratis	4	-	0€
Ionic	Gratis	4	-	0€
Tensorflow	Gratis	3	-	0€
Trello	Gratis	7	-	0€
Google Drive	Gratis	9	-	0€
Keras	Gratis	3	-	0€
Firebase	Gratis	4	-	0€

Visual Studio	Gratis	6	-	0€
GitKraken	Gratis	4	-	0€
Overleaf (LaTeX)	Gratis	2	-	0€
Ubuntu 16.04	Gratis	5	-	0€
macOS Mojave	Gratis	9	-	0€
Coste total del *software*				**0,00€**
Costes Indirectos				
Electricidad	47€	9	1	423€
Internet	27€	9	1	243€
Transporte	20€	9	1	180€
Coste total de los costes indirectos				**846,00€**

Tabla 6.1. Presupesto de los materiales

Tras haber expuesto los diferentes costes en lo que se refiere a los materiales utilizados durante el desarrollo del proyecto, a continuación se presentan los costes derivados de la plantilla.

Rol	Persona a cargo	Coste por hora (€)	Horas de trabajo útiles	Coste aplicado al uso (€)
Desarrollador de *FrontEnd*	Gabriel García	12€	150h	1.800€
Desarrollador de *BackEnd*	Gabriel García	12€	180h	2.160€
Experto en IA	Gabriel García	13€	240h	3.120€

Analista	Gabriel García	12€	155h	1.860€
Jefes de Proyecto	Juan Manuel Alonso y David Griol	12,5€	85h	1.062,5€
Coste total del personal				**10.002,50€**

Tabla 6.2. Presupesto de la plantilla

Con todo desglosado, se muestra el coste total del proyecto en la Tabla 6.3. Se le añade al total del proyecto un fondo de prevención destinado para imprevistos o cambios del 10 %.

Tipo de coste	Coste bruto (€)	Impuestos (€)	Coste neto (€)
Materiales	992,36€	208,39€	1.200,75€
Personal	10.002,50€	1.900,47€	11.902,97€
Fondo de prevención del 10 %			1.310,34€
Coste total del proyecto			**14.413,79€**

Tabla 6.3. Presupuesto total del proyecto

6.3. Impacto socioeconómico

Muchas de las empresas punteras dedicadas al desarrollo de aplicaciones móviles para el aprendizaje de idiomas están enfocadas al habla y reconocimiento de formas verbales, sustantivos o adjetivos. Muy pocas de estas empresas invierten en el aprendizaje de la escritura de los idiomas que imparten. La entrada de esta aplicación *web* en el mercado puede suponer un cambio de paradigma para estas empresas, las cuales podían apostar por estas pequeñas aplicaciones para implementar en su propio sistema de aprendizaje. Esto aportaría una gran ventaja tanto para los usuarios, ya que podrían aprender un idioma en su totalidad, como para la empresa, que tendría más usuarios y un sistema más completo.

Pero la entrada de la aplicación en el mercado también puede suponer un gran ries-

go: puede suponer que las empresas desarrollen su propio sistema de aprendizaje de escritura. Ya que estas empresas tienen más recursos que los disponibles durante el desarrollo de esta, es muy probable que su sistema desbanque al de este proyecto. Por esto, esta *web* se concibe de dos maneras diferentes: como una *web* de reconocimiento de caracteres independiente y como una posible *web* auxiliar o de apoyo para las grandes empresas. El objetivo principal de la aplicación es que los usuarios puedan llegar a aprender de manera completa un idioma mediante el uso de esta y otras aplicaciones, por lo que las alianzas y clientes objetivo son abundantes: desde las grandes aplicaciones (*Duolingo, Babbel, LingoDeer...*), academias de idiomas y hasta ONGs (como ayuda gratuita para aprender a escribir a personas que no tienen los recursos suficientes).

El hecho de que la aplicación *web* utilice un sistema de reconocimiento basado en imágenes mediante una técnica de inteligencia artificial, hace que la aplicación aplique una innovación puntera dentro del mercado de este tipo de aplicaciones. Solo unas pocas aplicaciones apuestan por el uso de modelos de predicción para el aprendizaje. Esto se puede deber a dos factores: al porcentaje de incertidumbre que existe al hacer una predicción en los modelos, ya que no son perfectos, y a los recursos económicos que se deberían de invertir como el reclutamiento de especialistas en inteligencia artificial, equipo de experiencia de usuario o desarrolladores *web*.

Y, aunque exista esta innovación dentro de la aplicación *web*, hay que tener en cuenta que las empresas objetivo son reacias a sistemas no desarrollados por ellas mismas. Estas pueden no estar interesadas en los sistemas con este tipo de aprendizaje para la escritura, por lo que la aplicación se tendría que mantener en el pequeño mercado de las aplicaciones dedicadas a la escritura de idiomas. Dada la buena respuesta de los modelos de reconocimiento y la buena acogida de la aplicación para el aprendizaje con su uso diario, la aplicación *web* no tendría problema para subsistir en el mercado actual consiguiendo ganancias mediante anuncios y patrocinios en otras *webs*.

6.4. Impacto futuro

Hoy en día, con el auge que ha tenido la inteligencia artificial en los últimos años y con la poca innovación que hay en el mercado actual, esta aplicación representa toda una novedad en el campo de la enseñanza de la escritura. Una evidencia de esto es que la mayoría de las aplicaciones estudiadas no muestran signos de inteligencia artificial. Pero, en el futuro, las tecnologías tienden a mejorar y a ser más accesibles a los desarrolladores y, en definitiva, tienden a ser más estables, por lo

que esta innovación que ahora supone el uso de redes de neuronas convolucionales aplicadas a la enseñanza de la escritura puede llegar en un futuro a ser la norma general del mercado.

La consultora Gartner, en su artículo "5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018" [59], define un gráfico mostrando la esperanza de vida útil que se espera que tenga una tecnología puntera como puede ser el *deep learning*. Este gráfico muestra todas las posibles etapas que una tecnología emergente pueda tener: la etapa de innovación, la de las sobreexpectativas, la de la desilusión y por último las del resurgimiento y la productividad.

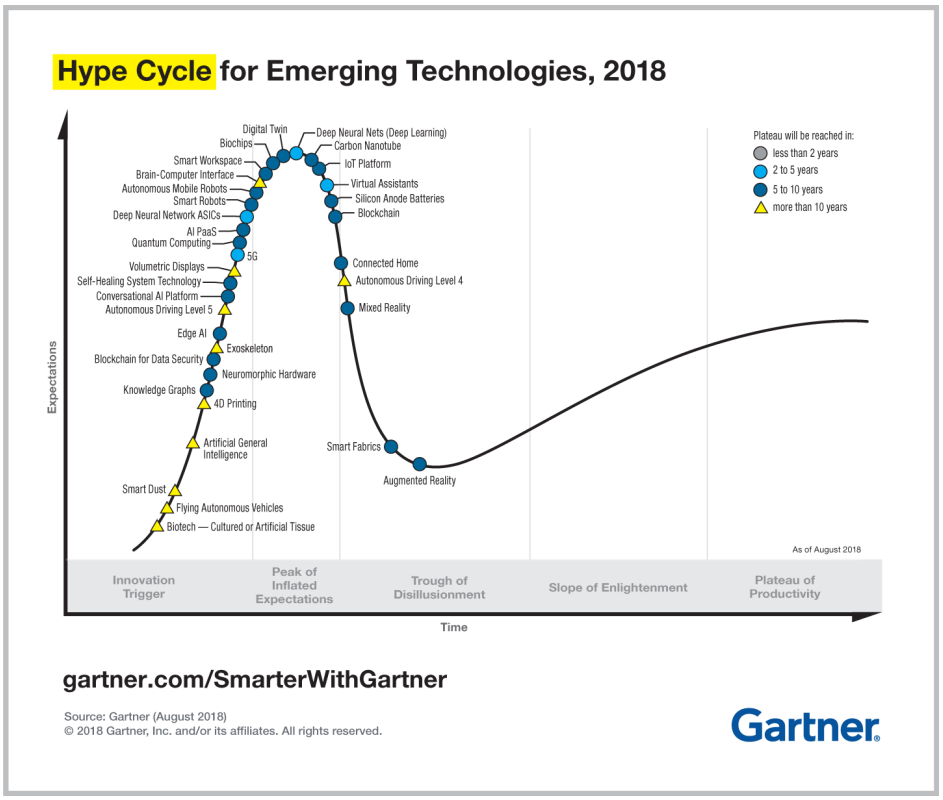


Figura 6.2. *Hype Cycle for Emerging Technologies, 2018*

En el caso del *deep learning*, la tecnología utilizada en este proyecto, se observa que está en lo más alto de la sobreexpectación, está en su época dorada. Esto quiere decir que el *deep learning* se está aplicando en todo tipo de campos consiguiendo grandes resultados. Ahora, lo que ha de resaltarse es que esta tecnología alcanzará su época de productividad y estabilidad de 2 a 5 años, por lo que las aplicaciones que hoy en día se están desarrollando mediante *deep learning* todavía tienen posibilidades de mejora.

Por lo tanto, se ha de seguir desarrollando el modelo de predicción para conseguir, si cabe, unos mejores resultados que los descritos en este trabajo. Esto se hará para que la innovación que supone la inclusión de esta aplicación *web* en el mercado no se quede rezagada, y tenga un futuro de productividad y eficiencia adecuado y competitivo.

7. CONCLUSIONES Y TRABAJOS FUTUROS

En esta sección final se recogen las conclusiones extraídas de todo el trabajo realizado a lo largo de los 9 meses de desarrollo. Finalmente, se exploran los posibles trabajos futuros a desarrollar en la aplicación *web*.

7.1. Conclusiones

La necesidad de enseñar la escritura como parte del aprendizaje de un idioma es una tarea fundamental y más cuando se trata de un idioma que utiliza caracteres diferentes como el árabe, el japonés, el cirílico, etc. El hecho de que las principales aplicaciones dedicadas a la enseñanza de idiomas no apliquen métodos de escritura es un grave error que se ha tomado a la ligera. Otras aplicaciones menos conocidas sí tienen en cuenta este importante paso en el aprendizaje pero el sistema implementado no es el correcto. Estas aplicaciones ayudan al usuario a recordar trazos, formas y estilos de los caracteres mediante plantillas que eliminan o adaptan el trazo realizado por el propio usuario si este no dibuja *exactamente* el carácter como se le pide. Con este método, el usuario aprende de manera "fraudulenta", ya que nunca se le dará por válido un dibujo de un carácter aparentemente correcto si no es exactamente igual que la plantilla de la aplicación. De estos dos problemas nace la necesidad y motivación de la elaboración de este proyecto.

Mediante el uso del *deep learning* como método de reconocimiento de caracteres, se ofrece al usuario un método alternativo más eficiente y adaptable para aprender a escribir los diferentes caracteres del japonés. Este método permite el reconocimiento de caracteres japoneses con el estilo caligráfico que se desee. Los mejores modelos de *hiragana* y *katakana*, los dos silabarios del japonés, obtienen una tasa de acierto de 99,61 % y 99,23 % respectivamente. Estos resultados tan competitivos se deben a la experimentación previa realizada con diferentes dominios (*Dogs vs. Cats*, FERC y CIFAR10) que nada tienen que ver con caracteres japoneses, en donde: se tratan los diferentes parámetros de las CNN, se exploran técnicas de mejora de la predicción (*data augmentation* y más imágenes de entrenamiento) y se estudia la importancia de un correcto balance de clases.

La aplicación brinda la posibilidad de practicar los caracteres básicos del japonés, además de ofrecer pequeños tests aleatorios para mejorar la asociación entre pronunciación y carácter. Con las evaluaciones realizadas a usuarios reales, queda demostrado que ¡Aprende! Japonés, junto con los modelos de predicción, cumplen la

función principal de este proyecto de manera sobresaliente: un 75 % de las personas encuestadas han quedado satisfechas con los servicios proporcionados, tanto con las respuestas o reconocimiento de caracteres de los modelos como por haber utilizado una aplicación intuitiva que les sirve para aprender a escribir japonés.

Finalmente, gracias a la integración de los modelos de predicción en la *web* y a la forma en la que esta se presenta a los usuarios, ¡Aprende! Japonés consigue una utilidad real en el campo del aprendizaje de la escritura japonesa con métodos innovadores de la inteligencia artificial.

7.2. Trabajos futuros

Una de las principales mejoras que se quieren desarrollar en este proyecto es la inclusión de nuevos idiomas para aprender a escribir. Esto tiene su origen en la evaluación de la aplicación con usuario reales, en las que se observa que un 84,1 % de los encuestados están dispuestos a probar o aprender a escribir los caracteres de un nuevo idioma como el ruso, el chino o el tailandés. Esto conlleva directamente a una búsqueda de conjuntos de datos del idioma en cuestión y a un nuevo desarrollo de un modelo de predicción y a un nuevo tratamiento de datos. Con esta adición de nuevos idiomas a la aplicación *web*, esta será capaz de aumentar su número de usuarios y clientes objetivo y, por tanto, representa una ampliación de mercado considerable.

Además de la adición de nuevos idiomas, como parte de la vida de toda aplicación, la constante mejora de los servicios que se ofrecen es fundamental. Por esto, pese a tener unos modelos que alcanzan más del 99,2 % de acierto, hay un margen de 0,8 % que se puede alcanzar mediante un estudio de las redes de neuronas convolucionales más en profundidad. Además, se deberá modificar la interfaz de usuario a medida que se añadan nuevas funcionalidades como la escucha de las pronunciaciones de los caracteres o la inclusión de nuevos ejercicios a añadir en los tests para mejorar la asociación entre carácter, pronunciación y escritura. También, como mejora del servicio, la *web* se migrará a una aplicación móvil para iOS y Android con la incorporación de Cordova en Ionic. Ya que muchos de los usuarios que realizaron la encuesta, la hicieron desde su teléfono personal (84,1 %), es conveniente pensar en una adaptación a una app pese a que la web está adaptada para móviles desde el navegador.

Como parte complementaria a la mejora del modelo, en vez del uso de redes convolucionales, se pueden utilizar redes de neuronas recurrentes o LSTM. Este tipo de

redes de neuronas son semejantes al perceptrón multicapa pero se caracterizan por que todas las neuronas de la red están interconectadas entre sí. Esta conectividad permite que la red "recuerde" las predicciones anteriores para usarlas como datos de entrada completamente nuevos. Esto se traduce a que con el uso de estas redes tan complejas, se puede hacer un seguimiento del orden del trazado de los caracteres, habilitando así, una aplicación que permite el reconocimiento de caracteres tanto por su forma como por su orden de trazado. La implementación de esta red se consideró en un primer momento pero debido a la alta complejidad de estas estructuras y al poco conocimiento sobre el *deep learning* que se tenía en esos momentos, se descartó. Ahora que se han adquirido unos buenos cimientos sobre el *deep learning* aplicado al procesamiento de imágenes, es posible que en un futuro cercano, se pueda implementar este tipo de redes en la aplicación.

Por último, una vez que la aplicación haya alcanzado un número considerable de usuarios y se hayan almacenado un número suficientemente alto de imágenes de caracteres, se puede crear un *corpus* o conjunto de datos nuevo de caracteres básicos japoneses abierto a la comunidad científica para su libre uso. Mediante estas mejoras, tanto de interfaz y funcionalidad como del modelo de predicción, y contribuciones a la comunidad científica, harán que la aplicación *web* aquí presentada llegue a convertirse en un referente del aprendizaje de la escritura.

I. SUMMARY

I.1. Motivation & objectives

Learning a language from scratch with the help of applications and web pages have been a very important trend over recent years. In these sites, associative methods are used in order to establish automatic relations between concepts of the language. This means that the user is required to learn based on the repetition of grammatical structures and word phrasing without having any previous knowledge of the matter.

In the case of the Japanese language, this repetition method is not sufficient. It could be good enough for learning new vocabulary, question and sentence phrasing, but not for the learning how to write. For this reason, the fact that a person is capable of differentiating a character, or its pronunciation, from another one is not enough to learn the language on its entirety: it is needed to know how to properly write the basic characters in order to understand the basis of the language. Applications like *Duolingo*, recognized internationally as one of the best apps for learning languages, do not focus in the importance of writing when learning a new language, and this is a severe handicap for languages such as Japanese.

This problem is not easy to solve though. It is very difficult to create an app or website that adapts and generalizes the writing of a language for every person. Some of the applications that are already dedicated to teach Japanese writing choose to work with a template where the user is obligated to draw exactly the character as it is stated on the template itself. With this method, these applications solve the generalization problem, as every user will draw the same character without fail. However, using this method if the user draws the character correctly but it does not match the template to perfection, the app will automatically classify it as being incorrectly drawn (see Figure I.1).

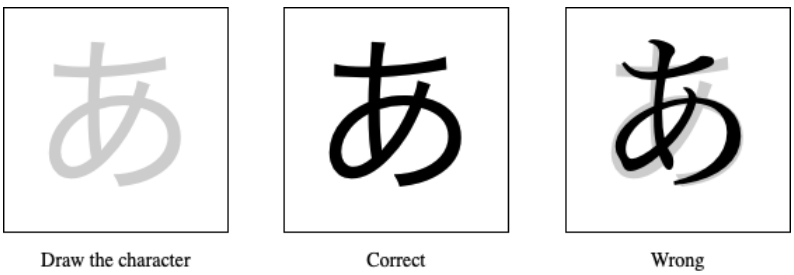


Figure I.1. Functioning of templates in apps

Another related problem with these applications is that they do not take into account that users have, individually their own calligraphic style, all of them unique. With these templates, the apps are teaching the writing of a language in an specific style without any margin of error, as shown in Figure I.1.

In order to reflect this diversity when it comes to writing and with the help of artificial intelligence, the proposed website will try to leave aside this template method and will let the user draw the characters as they want to, with their own calligraphic style. With this in mind, the characters shown in Figure I.1 will both be correct. Thus, this work makes possible the creation of a website that will be entirely dedicated to the teaching of Japanese writing. It also serves as an autodidact method to learn the basis of Japanese as well as a complement to learning the rest of the language.

I.2. Prediction model

There are various machine learning-based methods for character recognition purposes like support vector machines, multilayer perceptrons or k-nearest neighbours. However, for the characteristics and the outstanding results shown in image recognition, the use of convolutional neural networks (CNN) has been selected for recognizing handwritten Japanese characters. These networks have been coded entirely in Python. The friendly and well-documented Keras interface [60] has been used to build the model, using Tensorflow as back-end for all the calculations needed.

Before the experimentation with the Japanese handwritten database, other data sets were initially selected in order to gather information about CNNs behavior and parameters and how to treat CNNs on the Keras environment.

The first data set that has been employed is Dogs vs. Cats data set fom Kaggle [61]. From it, it is gathered that the application of a vast data augmentation (application of transformation operations to the original images to create new ones) is harmful for the model: it applies to much noise and randomness to the images creating new samples that are not representative in the data set thus not contributing to the model. With the application of a moderate data augmentation on the samples of the data set, the model achieves a 70 % up to a 90 % accuracy recognizing between images of cats and dogs. With this, the data augmentation technique shows its importance of appliance in this kind of experimentation. Another important conclusion that is extracted from this data set is that the more images used for training

the model, the better result the model will have, as it has more samples to work with.

The second data set that it has been worked on is the Emotion Detection from Facial Expressions (FERC) also from Kaggle [62]. At glance, this peculiar data set was promising as in the first experiments the model would classify +90 % of the samples correctly. But this result was strange as in the Kaggle competition the accuracy mean was around 52 %. In order to depict what was happening with the model, a confusion matrix was added to the evaluation process of the model, showing us the following matrix:

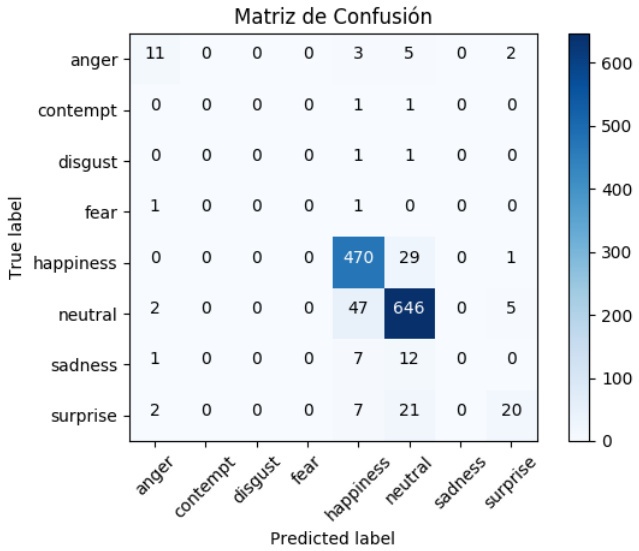


Figure I.2. Confusion matrix of experiment 0018-FERC-VGG16-0001

It is observed that almost every sample of the data set is gathered on two of the eight classes this data set has. This means that the whole data set is deeply unbalanced (see Figure I.3). To put this into perspective, both classes have around 93 % of accuracy while the rest obtain almost 0 %, giving the model a total of an 88 % of accuracy. The simple fact that these two classes obtain such good results shadows the bad ones giving the model a fake result. This problem could be resolved by oversmapling or undersampling the data set.

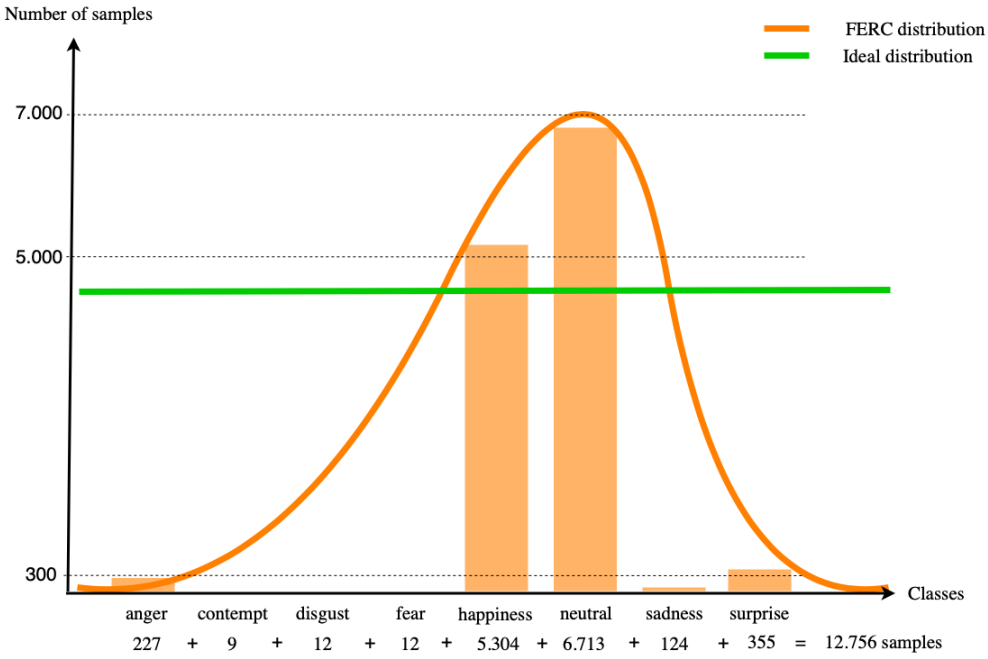


Figure I.3. FERC data distribution

In summary, what is learned from this data set is that every data set must be previously analyzed to verify if all samples are balanced within the classes in order to not have fake results when experimenting. An example of this can be seen in the Dogs vs. Cats data set, which contains a perfectly balanced data set with 12,500 images of cats and 12,500 of dogs.

The last data set before the Japanese one is CIFAR10 [30]. This data set is perfectly balanced along its ten classes with 6.000 samples each. Here the extraction of information of every sample improves as well as the convergence fastness of the model to a specific peak accuracy value, leading to a slightly better generalization and behavior of the model.

Once all this information is processed and learned, the experiments with the Japanese handwritten data set begins. The data set used in this work is the ETL Character Database. The ETL Character Database (hereinafter referred to as "Database") is composed of 9 data sets of scanned images of handwritten and printed characters supplied formerly by the Electrotechnical Laboratory (ETL) and currently by its reorganized successor the National Institute of Advanced Industrial Science and Technology (AIST) [48]. With this data set, two major subsets will be created: one for *hiragana* characters and another for *katakana* ones, the two syllabaries of Japanese.

After analyzing the whole data set of ETL Character Database and extracting only the needed characters (with the help of python scripts provided in the official website of the data set), the two sets are ready to be fed into the CNNs.

Both sets have 46 classes defined: *katakana* including 64,763 images with 1,407 images per class, and *hiragana* with 38,180 samples with 830 per class, showing an almost perfect balance between images and classes on both data sets.

First, the *katakana* experimentation is performed. The first test inherits all the major improvements found on the previous experimentation with other data sets: the moderated data augmentation and gathering a sufficiently large amount of samples to train from Dogs vs. Cats and the improvements of information extraction and behavior of the model of CIFAR10 experiments are also taken into account. With this, the experiment 0034-KATA-VGG16-0005 reaches an accuracy value of 99.18 %, observing a super-fast converging model with no downsides.

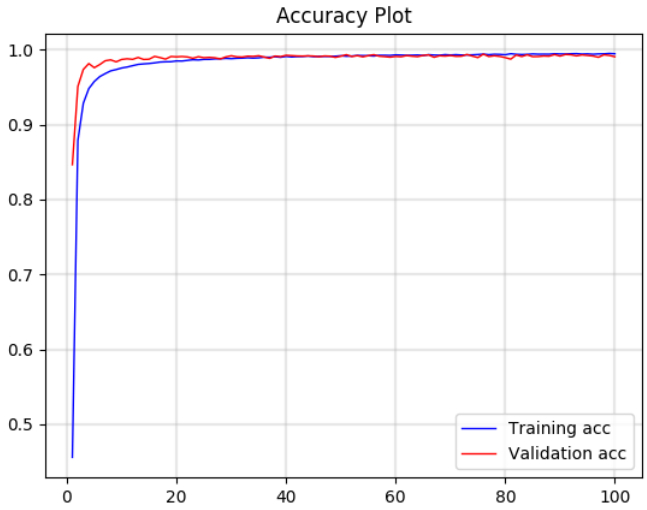
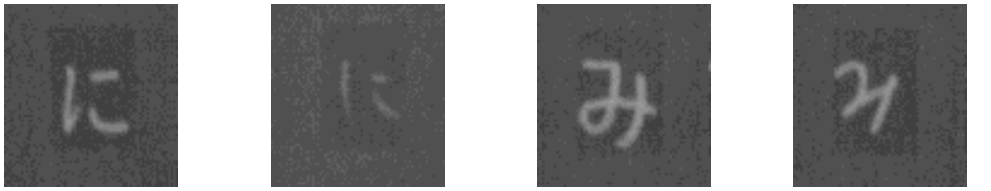


Figure I.4. Experiment 0034-KATA-VGG16-0005

The input image dimension used in the CNN is 64×64 and it is reduced to 32×32, reducing the information of the samples. This change comes along with the reduction of the training samples in order to see if any of it affects to the information extraction capabilities of the model. After the experiment we obtain a 98.94 % mean of accuracy, showing that the time of training is reduced by almost half thanks to the reduction of the size sample while maintaining the model accuracy despite of the cut of information.

Another two experiments where performed with the above specifications, obtaining an accuracy of 98.99 % and 99.15 %. There is almost no variation on the results independently of the modifications applied to the model. This leads to assume that further techniques of experimentation like cross validation are of no use, as it will not be possible to extract any significant conclusions.

The *hiragana* model consists of four different subsets of the ETL Character Database. One of them because of the bad resolution and bad representation of the character on the images may be harmful for the model and is considered for its inclusion in the experiments (see Figure I.5).



(a) に character **(b)** に outlier **(c)** み character **(d)** み outlier

Figure I.5. Outliers of the ETL-4 Character Database subset

In order to decide if the outliers conform the final set or not, a set of experiments were completed. The first experiment 0039-HIRA-VGG16-0003 was performed without the outliers and a 99.53 % of accuracy is obtained, showing the same behavior as the *katakana* models. Now, for comparison, the outliers are introduced into the *hiragana* set and the 0040-HIRA-VGG16-0004 experiment obtains a 99.18 % of accuracy. There is a certain margin between both experiments and because of that they are repeated in order to see if that margin maintains (see Figure I.6).

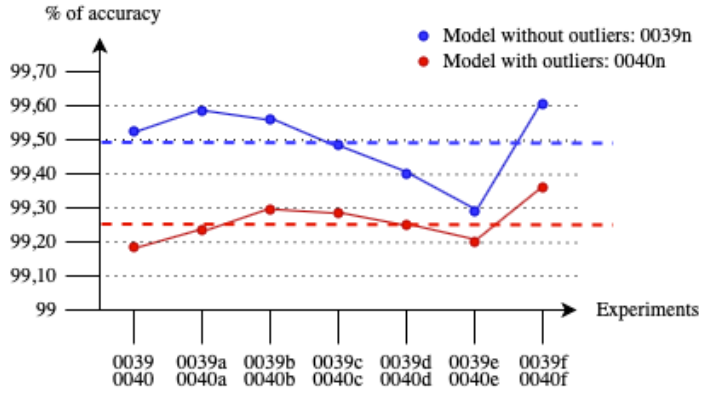


Figure I.6. Tendency between models of *hiragana*

The tendency, as observed, roughly maintains, confirming that the outliers were slightly harming the prediction of the model and they will not be included in the final model. For a final experimentation, the *katakana* model stays as it is with 64,763 images and the *hiragana* model has 32,660 samples instead of 38,180. In the following tests, only the dimension of the input samples varies along with the structure of the CNN (see table I.1).

Structures of the final models									
64 × 64	32 × 32	32 × 32	32 × 32	32 × 32	32 × 32	64 × 64	32 × 32	32 × 32	64 × 64
Conv2D(32)	Conv2D(32)	Conv2D(32)	Conv2D(16)	Conv2D(32)	Conv2D(32)	Conv2D(32)	Conv2D(16)	Conv2D(32)	Conv2D(32)
Conv2D(32)	Conv2D(32)	MaxPooling2D	MaxPooling2D	Conv2D(32)	Conv2D(32)	Conv2D(32)	Conv2D(16)	Conv2D(32)	Conv2D(32)
MaxPooling2D	MaxPooling2D	Conv2D(64)	Conv2D(32)	MaxPooling2D	MaxPooling2D	MaxPooling2D	MaxPooling2D	Conv2D(32)	Conv2D(32)
Conv2D(64)	Conv2D(64)	MaxPooling2D	MaxPooling2D	Conv2D(64)	Conv2D(64)	Conv2D(64)	Conv2D(32)	MaxPooling2D	MaxPooling2D
Conv2D(64)	Conv2D(64)	Conv2D(128)	Conv2D(64)	MaxPooling2D	Conv2D(64)	Conv2D(64)	Conv2D(32)	Conv2D(64)	Conv2D(64)
MaxPooling2D	MaxPooling2D	MaxPooling2D	MaxPooling2D	Conv2D(128)	MaxPooling2D	MaxPooling2D	MaxPooling2D	Conv2D(64)	Conv2D(64)
Conv2D(128)	Conv2D(128)		Conv2D(128)	MaxPooling2D	Conv2D(128)	Conv2D(128)	Conv2D(64)	Conv2D(64)	Conv2D(64)
Conv2D(128)	Conv2D(128)		MaxPooling2D		MaxPooling2D	MaxPooling2D	MaxPooling2D	MaxPooling2D	MaxPooling2D
Conv2D(128)	Conv2D(128)						Conv2D(128)	Conv2D(128)	Conv2D(128)
MaxPooling2D	MaxPooling2D						MaxPooling2D	Conv2D(128)	Conv2D(128)
								MaxPooling2D	MaxPooling2D
Flatten									
Dense(1024)									
Dense(1024)									
Dense(46)									
Softmax									
0042-KATA	0043-KATA	0052-KATA	0053-KATA	0054-KATA	0055-KATA	0056-KATA	0057-KATA	0058-KATA	0059-KATA
99,21 %	99,18 %	98,98 %	98,65 %	99,11 %	99,10 %	99,17 %	98,99 %	98,80 %	99,23 %
0039-HIRA	0041-HIRA	0044-HIRA	0045-HIRA	0046-HIRA	0047-HIRA	0048-HIRA	0049-HIRA	0050-HIRA	0051-HIRA
99,53 %	99,42 %	99,10 %	99,18 %	99,40 %	99,51 %	99,29 %	99,37 %	99,34 %	99,40 %

Table I.1. Final experiments of *katakana* and *hiragana*

The best models of each set are chosen as the ones to be included in the website: for *katakana*, the model 0059-KATA-CCCMCCCMCCM-0018 (99.23 %) and for *hiragana*, the model 0039f-HIRA-VGG16-0002 (99.61 %). All results have been validated with the help of the article written by Charlie Tsai "Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks" [63].

I.3. Web application

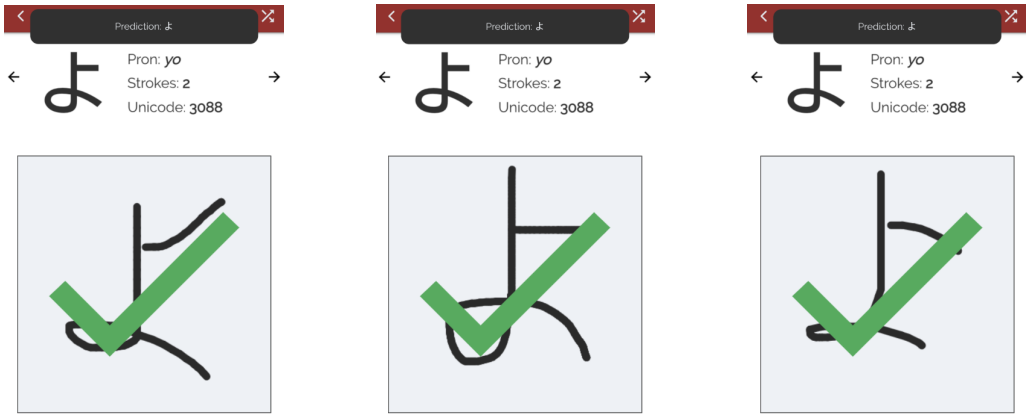
In order to create the website, Angular [64], Ionic [65] and Firebase [66] were chosen as the framework because of the user experience focus, beauty and simplicity of Ionic and because of the whole core of tools Angular itself provides to create a perfectly complete website application, along with the power of Firebase to manage data bases and hosting. The models obtained on the previous section are migrated into Angular thanks to Tensorflow.js [67], a tool that translates the models from the Python environment to a Javascript/Typescript one, like the one Angular offers.

The main objective of the website, Learn! Japanese, is creating a new teaching paradigm for writing Japanese powered with artificial intelligence techniques, specifically convolutional neural networks, in order to recognize handwritten Japanese characters in images (with the desired calligraphic style).

The website consists of a section for practicing and another for testing the user's knowledge on the basic characters of Japanese: *hiragana* and *katakana*. It does not require registration, as there is an option to practice all the basic Japanese characters without any restriction, but when registered, the user is able to make tests and keep track of his/her results on them. Only the registered user will be able to see his/her progress on his/her learning process with test-only data such like: number of test performed and accuracy percentage on each syllabary. Furthermore, the website is available in two languages: Spanish and English, setting that can be changed in the website as well as − only for registered users − modifying the username and password, and even deleting the account.

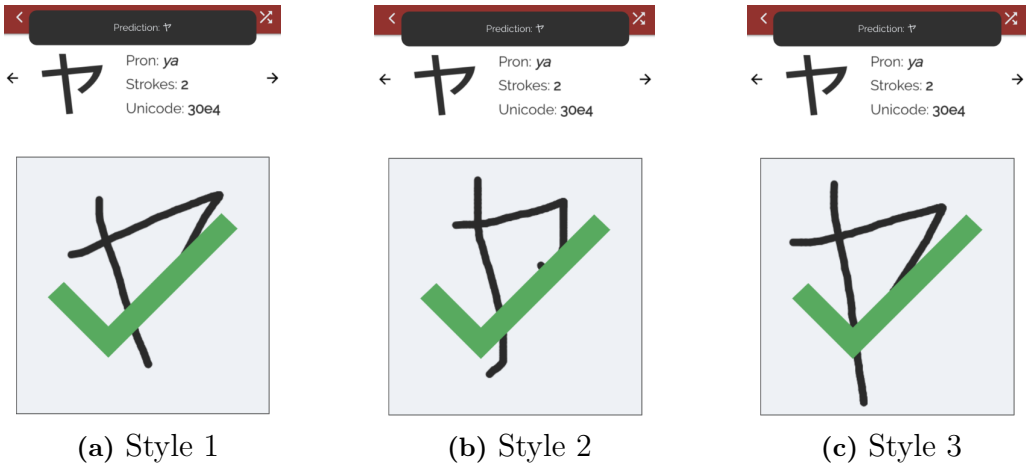
In the practice modality, the user can see the character pronunciation, the number of strokes and the Unicode code, along with a short preview on how the stroke order of the character is. The user is also able to select the character and the syllabary to practice at will. When the user draws the character, he/she will be able to delete the drawing or to evaluate it with the prediction models. When evaluating the drawn character in practice, an image will be taken of the drawing and it will be sent to

the model of *hiragana* or *katakana* (depending on the selected syllabary) and it will provide a prediction with the recognized character. Then, an automatic feedback will be displayed on the page telling the user if the drawn character is correct or not, showing the character the model predicted as an additional information to the user. To prove that the accuracy levels the models showed in the experimentation match reality, the Figures I.7 and I.8 display two different characters with different calligraphic style: the *hiragana* character よ and the *katakana* ヤ respectively, showing an impressive performance on recognizing the chosen characters.



(a) Style 1 **(b)** Style 2 **(c)** Style 3

Figure I.7. Verification of different calligraphic styles for *hiragana*



(a) Style 1 **(b)** Style 2 **(c)** Style 3

Figure I.8. Verification of different calligraphic styles for *katakana*

In the test modality, the user is able to select whether to do the test on *hiragana* or *katakana* characters. The test consists of ten random timed questions in which characters are requested to be drawn based on their pronunciation. During the test, no feedback will be displayed. When the user finishes the test, all the results will be

shown (along with the feedback) to the user in order to compare the answers given to the correct results.

As additional information, all drawings made by non-registered and registered users will be saved in order to contribute to the scientific community with the making of a public database of basic Japanese characters.

After the main definition of the website, a survey for evaluating the application with 44 real users is performed. Most of the users made use of the practice section without registering (56.8 %) while the 27.3 % do register. Either way, the majority of the users (79.6 %) only tested the practice section, giving as feedback a good user experience as the website displays a simple and intuitive interface even though there are tutorials to guide the users (which a 40.9 % of the surveyed obviated them). With respect of the time of response and load of the model, the surveyed were satisfied for the most part (75 %), confirming the competent results of the models. And, for last, a 75 % of the users were determined to acknowledge the utility of the website for learning how to write the basic Japanese characters, showing that the proposed method is successful.

I.4. Conclusions & future work

As a conclusion, using deep learning as a method for character recognition, the user is offered a more efficient and adaptive alternative method to learn to write the different characters of the Japanese language. This method allows the recognition of the characters with the desired calligraphic style, getting rid of templates. Thanks to an extensive previous study with different domains (Dogs vs. Cats, FERC and CIFAR10), two models of the ETL Character Database have achieved very competitive results and this allows the website to have a real use in the field of language learning. Lastly, with the evaluations performed on the website, it has been demonstrated that the application, together with the predictive models, fulfills the main function in an outstanding way: teaching how to write basic Japanese characters without any restrictions on the calligraphy.

For future works, the implementation of new languages, apart from Japanese, will be included, such as Russian, Chinese or Thai. This idea comes out of the user evaluation as almost a 84,1 % of the surveyed agreed on enlarging the website scope with more languages. This will carry a new search of a data set of characters of the desired language and a new experimentation to find the most suitable model. With

the addition of languages to the application, it will be easier to gain more users and potential clients.

Besides the addition of new languages into the website, the constant improvement of the website is a must in this kind of projects. Even though the models results are very competitive, they have an improvement margin of 0,8 % that can be difficult to achieve but not impossible through the deep study of convolutional neural networks. In addition, there is room for improvement in the user interface too. As time passes by, the interface may need modifications with the inclusion of new functionalities like listening to the pronunciation of the characters or the addition of more exercises to randomly do on the tests. Following the same path, as an 84,1 % of the users that answered the survey on their mobile phones, it will be a profiting move to migrate the website to an iOS and Android mobile app.

As a complementary improvement of the models, instead of using CNNs, recurrent neural networks could also been used. This type of networks are characterized by the fact that they can "remember" previous predictions to use them as completely new input data. This means that with the use of these complex networks, the stroke order of the characters can be tracked, thus enabling a website that allows the recognition of characters both by their shape and by their stroke order. The implementation of this network was considered at first but due to the high complexity of it and the little knowledge about the deep learning, it was discarded. Now that it has been acquired a good foundation on deep learning applied to image processing, it is completely possible that in the near future, this type of networks could be implemented in the website.

Finally, once the application reaches a considerable number of users and a sufficiently number of characters drawn, a new *corpus* or data set of basic Japanese characters can be created. This database will be open to the scientific community for free usage. Through these improvements, both interface and functionality as well as the prediction model, and contributions to the scientific community, they will make the web application presented here become a reference for teaching on how to write.

BIBLIOGRAFÍA

[1] M. G. Aller, «El Fin de la Paciencia», *El Independiente*, `https://www.elindependiente.com/economia/2018/04/27/el-fin-de-la-paciencia/`, 2018.

[2] W. S. McCullogh y W. H. Pitts, «A Logical Calculus Of The Ideas Immanent In Nervous Activity», *Bulletin of Mathematical Biophysics*, vol. 5, págs. 115-133, 1943.

[3] D. O. Hebb, *The Organization of Behaviour: A Neuropsychological Theory*. John Wiley & Sons, 1949.

[4] F. Rossenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain», *Psychological Review*, vol. 65, n.º 6, págs. 386-408, 1958.

[5] B. Widrow y T. Hoff, *Adaptative Switching Circuits*. 1960.

[6] M. Minsky y S. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[7] J. L. McClelland, D. E. Rumelhart y G. E. Hinton, *The appeal of parallel distributed processing*. Morgan Kaufmann, 1988.

[8] Y. LeCun, «Une procédure d'apprentissage pour réseau a seuil asymmetrique», *Proceedings of Cognitiva*, vol. 85, págs. 599-604, 1985.

[9] Y. LeCun, Y. Bengio y G. Hinton, «Deep Learning», *Nature*, vol. 521, págs. 436-444, 2015.

[10] K. Fukushima, «Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position», *Biological Cybernetics*, vol. 36, págs. 193-202, 1980.

[11] D. Hubel y E. Wiesel, «Receptive fields of single neurones in the cat's striate cortex», *J. Physiol*, vol. 148, n.º 3, págs. 574-591, 1959.

[12] Y. LeCun y col., «Backpropagation Applied to Handwritten Zip Code Recognition», *Neural Computation*, vol. 1, n.º 4, págs. 541-551, 1989.

[13] S. Sudholt y G. A. Fink, «PHOCNet: A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents», 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), Shenzhen, China, 2016.

[14] R. F. Nogueira, R. de Alencar Lotufo y R. C. Machado, «Fingerprint Liveness Detection Using Convolutional Neural Networks», *IEEE Transactions on Information Forensics and Security*, vol. 11, n.º 6, págs. 1206-1213, 2016.

[15] W. Sun, T. L. Tseng, J. Zhang y W. Qian, «Enhancing deep convolutional neural network scheme for breast cancer diagnosis with unlabeled data», *Computerized Medical Imaging and Graphics*, vol. 57, págs. 4-9, 2017.

[16] P. Isasi e I. M. Galván, «Redes de Neuronas Artificiales: Un enfoque práctico», en. Pearson-Prentice Hall, 2004, págs. 63-64.

[17] V. Patil y S. Shimpi, «Handwriting English Character Recognition Using Neural Network», *Elixir Computer Science Engineering*, vol. 41, págs. 5587-5591, 2011.

[18] I. Colombet y col., «Models to predict cardiovascular risk: comparison of CART, multilayer perceptron and logistic regression», *Proceedings of the AMIA Symposium*, vol. 156, 2000.

[19] S. Lee y J. Y. Choeh, «Predicting the helpfulness of online reviews using multilayer perceptron neural networks», *Expert Systems with Applications*, vol. 41, n.º 6, págs. 3041-3046, 2014.

[20] C. Cortes y V. N. Vapnik, «Support-vector networks», *Machine Learning*, vol. 20, n.º 3, págs. 273-297, 1995.

[21] S. Ray, *Analytics Vidhya*, `https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/`, accedido por última vez el 11 de abril de 2019, sep. de 2017.

[22] K. Kim, K. Jung y H. Kim, «Support Vector Machines: Theory and Applications. Studies in Fuzziness and Soft Computing», en. Berlin, Heidelberg: Springer, 2005, cap. Fast Color Texture-Based Object Detection in Images: Application to License Plate Localization, págs. 297-320.

[23] K. Kyoung-jae, «Financial time series forecasting using support vector machines», *Neurocomputing*, vol. 55, n.º 1-2, págs. 307-319, 2003.

[24] Z. Sun y H. Sun, «Support vector machine approach for protein subcellular localization prediction», *Bioinformatics*, vol. 17, págs. 721-728, 2001.

[25] S. J. Smith, M. Bourgoin, K. Sims y H. Voorhees, «Handwritten character classification using nearest neighbor in large databases», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, n.º 9, págs. 915-919, 1994.

[26] S. B. Imandoust y M. Bolandraftar, «Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events: Theoretical Background», *Journal of Engineering Research and Applications*, vol. 3, n.º 5, págs. 605-610, 2013.

[27] M. Maltamo y A. Kangas, «Methods based on k-nearest neighbor regression in the prediction of basal area diameter distribution», *Canadian Journal of Forest Research*, vol. 28, n.º 8, págs. 1107-1115, 1998.

[28] K. Nakai y P. Horton, «Better Prediction of Protein Cellular Localization Sites with the k Nearest Neighbors Classifier», Proceedings of Intelligent Systems in Molecular Biology, Halkidiki, Greece, 1997.

[29] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-Based Learning Applied to Document Recognition», *Proceedings of the IEEE*, vol. 86, n.º 11, págs. 2278-2324, 1998.

[30] A. Krizhevsky, «Learning Multiple Layers of Features from tiny Images», 2009.

[31] Y. LeCunn, C. Cortes y C. J. C. Burges, *THE MNIST DATABASE of handwritten digits*, `http://yann.lecun.com/exdb/mnist/`, accedido por última vez el 17 de abril de 2019.

[32] wikiabhi, *Github*, `https://www.github.com/wikiabhi/Cifar-10/`, accedido por última vez el 20 de abril de 2019.

[33] Y. Abouelnaga, O. S. Ali, H. Rady y M. Moustafa, «CIFAR10 KNN-based Ensemble of Classifiers», *arXiv:1611.04905 [cs.CV]*, 2016.

[34] Z. Lin, R. Memisevic y K. Konda, «How far can we go without convolution: Improving fully connected networks», *arXiv:1511.02580 [cs.LG]*, 2015.

[35] R. Beneson, *Classification Datasets Results*, `rodrigob.github.io/are_we_there_yet/build/classification_dataset_results.html`, accedido por última vez el 19 de abril de 2019, feb. de 2016.

[36] A. Eyerly, *UCLA Researchers Map Brain Development in Four Dimensions, Revealing Stage-Specific Growth Patterns in Children*, `http://users.loni.ucla.edu/~thompson/MEDIA/ucnewswire.html`, accedido por última vez el 28 de abril de 2019, mar. de 2000.

[37] K. G. Henshall y T. Takagaki, *Learning Japanese Hiragana And Katakana*. Tuttle Publishing, 2014.

[38] *LingoDeer: Aprende Inglés, Japonés, Coreano o Chino*, `https://play.google.com/store/apps/details?id=com.lingodeer`, accedido por última vez el 29 de abril de 2019.

[39] Jernung, *Write It! Japanese*, `https://play.google.com/store/apps/details?id=com.jernung.writeit.jpn`, accedido por última vez el 29 de abril de 2019.

[40] Learning and Writing Studio, *Japanese Writing*, https://play.google.com/store/apps/details?id=com.writing.learn.japanese, accedido por última vez el 29 de abril de 2019.

[41] D. Meeuwis, *Write Japanese*, https://play.google.com/store/apps/details?id=dmeeuwis.kanjimaster, accedido por última vez el 29 de abril de 2019.

[42] *Keras: The Python Deep Learning library*, https://keras.io, accedido por última vez el 23 de abril de 2019.

[43] *Tensorflow*, https://www.tensorflow.org, accedido por última vez el 23 de abril de 2019.

[44] *Pytorch*, https://pytorch.org, accedido por última vez el 23 de abril de 2019.

[45] Microsoft, *Dogs vs. Cats*, https://www.kaggle.com/c/dogs-vs-cats/overview, accedido por última vez el 26 de abril de 2019.

[46] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», *arXiv:1409.1556 [cs.CV]*, 2014.

[47] *Emotion Detection From Facial Expressions*, https://www.kaggle.com/c/emotion-detection-from-facial-expressions, accedido por última vez el 1 de mayo de 2019.

[48] Electrotechnical Laboratory, Japanese Technical Committee for Optical Character Recognition, ETL Character Database, 1973-1984.

[49] C. Tsai, «Recognizing handwritten Japanese characters using deep convolutional neural networks», *Universidad de Stanford, California*, 2016.

[50] *What is Angular?*, https://angular.io/docs, accedido por última vez el 8 de mayo de 2019.

[51] *What is Ionic Framework?*, https://ionicframework.com/docs/intro, accedido por última vez el 8 de mayo de 2019.

[52] *Getting Started with React*, https://reactjs.org/docs/getting-started.html, accedido por última vez el 8 de mayo de 2019.

[53] *Tensorflow For Javascript*, https://www.tensorflow.org/js, accedido por última vez el 9 de mayo de 2019.

[54] *Firebase*, https://firebase.google.com, accedido por última vez el 9 de mayo de 2019.

[55] *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*, https://www.boe.es/eli/es/lo/2018/12/05/3, accedido por última vez el 4 de junio de 2019.

[56] *GNU General Public License*, https://www.gnu.org/licenses/gpl-3.0.html, accedido por última vez el 4 de junio de 2019.

[57] *APACHE LICENSE, VERSION 2.0*, https://www.apache.org/licenses/LICENSE-2.0, accedido por última vez el 4 de junio de 2019.

[58] *MIT License (Expat)*, https://tldrlegal.com/license/mit-license#summary, accedido por última vez el 4 de junio de 2019.

[59] K. Panetta, *5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018*, https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/, accedido por última vez el 23 de mayo de 2019.

[60] *Keras: The Python Deep Learning library*, https://keras.io, last accessed May 25th, 2019.

[61] Microsoft, *Dogs vs. Cats*, https://www.kaggle.com/c/dogs-vs-cats/overview, last accessed May 25th, 2019.

[62] *Emotion Detection From Facial Expressions*, https://www.kaggle.com/c/emotion-detection-from-facial-expressions, last accessed May 25th, 2019.

[63] C. Tsai, «Recognizing handwritten Japanese characters using deep convolutional neural networks», *university of Stanford, California*, 2016.

[64] *What is Angular?*, https://angular.io/docs, last accessed May 25th, 2019.

[65] *What is Ionic Framework?*, https://ionicframework.com/docs/intro, last accessed May 25th, 2019.

[66] *Firebase*, https://firebase.google.com, last accessed May 25th, 2019.

[67] *Tensorflow For Javascript*, https://www.tensorflow.org/js, last accessed May 25th, 2019.

A. ANEXO I: PROGRAMA DE EXPERIMENTACIÓN

En este anexo se va a comentar en detalle cómo Keras trata las redes de neuronas convolucionales y cuál es la estructura base de todos los programas usados para la experimentación con los distintos dominios estudiados.

Toda la información básica de los parámetros y funciones explicadas se basa en la documentación oficial de Keras y en el libro de François Chollet, creador de Keras, *Deep Learning with Python* [1].

A.1. Preprocesamiento de imágenes

Keras y Tensorflow trabajan con **tensores**. Los tensores no son más que *arrays* o listas n-dimensionales. Por esto, lo primero que se debe de hacer para entrenar cualquier red con Keras, es convertir las imágenes a tensores tridimensionales de la forma $W_I \times H_I \times C_I$ o $H_I \times W_I \times C_I$, donde W_I y H_I es la anchura y altura en píxeles respectivamente, y C_I es la profundidad (formato de color) de la imagen. Esta última dimensión hace referencia a la gama de colores que se usa, como puede ser RGB ($C_I = 3$) o escala de grises ($C_I = 1$).

Esta conversión, necesaria en todos los conjuntos, se realiza con la ayuda de la librería de visión por computador **openCV** y con la librería dedicada a las operaciones matemáticas **numpy**. En los conjuntos, normalmente las muestras están almacenadas por clase. Cada clase está diferenciada en una carpeta con el mismo nombre de la clase: `claseX`, por lo que en el conjunto existen tantas carpetas como clases haya, conteniendo cada una de ellas un número determinado de imágenes. Teniendo esto en cuenta, lo primero que se hace en los programas de experimentación es almacenar como *string* las rutas completas de las diferentes imágenes clasificadas del conjunto en un *array* `imgsClaseX` con ayuda del paquete **os**. Ya que las imágenes se extraen por clase, habrá tantos *arrays* como clases tenga el conjunto. Después, se realiza la separación del conjunto total en entrenamiento, validación y prueba mediante las operaciones de partición que ofrece python. Para crear el conjunto de entrenamiento, se extraen N rutas de imágenes de cada `imgsClaseX`, para validación $M - N$ y para el conjunto de prueba M imágenes, siempre siguiendo una proporción apropiada y declarando $M > N$.

```
imgsClaseX = ['dataset/claseX/'+i for i in os.listdir('dataset
   /claseX/')]
train_images = imgsClaseX[:N]
val_images = imgsClaseX[N:M]
test_images = imgsClaseX[M:]
```

Código A.1. Lectura de imágenes y separación del conjunto de datos

Seguidamente, se extraen las clases correspondientes a las imágenes de cada conjunto con la función `extract_labels`. Esta función saca partido de los nombres de las rutas extraídas para distinguir si una imagen pertenece a una clase o a otra. Esta distinción se hace mediante un *string* `CLASS` que debe de ser común para todos los patrones de una clase. Este identificador suele corresponder o con el nombre de la carpeta `claseX` visto anteriormente o con los nombres de las propias muestras, por lo que la extracción es inmediata. Por ejemplo: tenemos un conjunto de perros y gatos en los que cada imagen se denota por el nombre *"<animal><número de imagen>.png"* como `dataset/perro12443.png` y `dataset/gato443.png`. El *string* de distinción sería "perro" y "gato", por lo que si se está extrayendo la clase de cualquier imagen y *animal* = *perro*, se sabe que pertenece a la clase "perro" y si *animal* = *gato*, pertenece a "gato".

```
def extract_labels(imgs,labels):
    for i in imgs:
        if CLASS in i:
            labels.append([0,1])
        else:
            labels.append([1,0])

train_labels = []
val_labels = []
test_labels = []
extract_labels(train_images,train_labels)
extract_labels(train_images,val_labels)
extract_labels(train_images,test_labels)
```

Código A.2. Extracción de clases de las imágenes

Las clases de las imágenes se guardan en `train_labels`, `val_labels` y `test_labels` utilizando el método *one hot encoding*. Este método establece un *array* de *arrays* de dimensión $I_L \times R_L$, siendo I_L el número de imágenes del conjunto de datos L y R_L el número de clases del conjunto. En este *array* se traducen las clases nominales, por ejemplo: se tienen dos imágenes clasificadas como "perro" y "gato", por lo que el *array* queda: $[[0,1],[1,0]]$, siendo $[0,1]$ "perro" y $[1,0]$ "gato". Ya que una red

de neuronas no puede trabajar más que con números, este es un paso vital a realizar.

Posteriormente, una vez extraídas las clases de todo el conjunto y separado correctamente en entrenamiento, validación y prueba, en `prep_data`, se crea el tensor de 4 dimensiones que va a albergar todas las imágenes convertidas: $N \times W_I \times H_I \times C_I$, donde N es la cantidad de imágenes almacenadas en `train_images`, `val_images` y `test_images` respectivamente (Código A.3, línea 9). Ahora, por cada imagen se llama a la función `transform_image`. Esta función lee la imagen en la posición `i` almacenada en `images` y la divide en tres canales: rojo, verde y azul (RGB), creando 3 *arrays* diferentes con dimensión $W_I \times H_I$ conteniendo en cada posición el valor píxel de la imagen del canal extraído (Código A.3, línea 3). Este paso es esencial, ya que las redes de todos los experimentos se han diseñado para que acepte únicamente imágenes tipo RGB. Si en alguna ocasión, se tratan imágenes de otro tipo, como GRAYSCALE, el programa es capaz de convertir esa imagen a RGB para que la imagen pueda ser correctamente analizada. Por último, los 3 *arrays* de $W_I \times H_I$ se unen en uno solo, creando un tensor $W_I \times H_I \times 3$ equivalente a la imagen y se aplica una redimensión de $W'_I \times H'_I$ a este, ya que es posible que no todas las imágenes del conjunto tengan las mismas dimensiones (Código A.3, línea 5). Se necesitan unas dimensiones constantes $W'_I \times H'_I$ a lo largo del conjunto para que la red funcione correctamente.

```
1  def transform_image(file_path):
2      img = cv2.imread(file_path, cv2.IMREAD_COLOR)
3      b,g,r = cv2.split(img)
4      img2 = cv2.merge([r,g,b])
5      return cv2.resize(img2, (WIDTH, HEIGHT), interpolation=cv2
           .INTER_CUBIC)
6
7  def prep_data(images):
8      count = len(images)
9      data = np.ndarray((count, WIDTH, HEIGHT, CHANNELS), dtype=
           np.uint8)
10     for i, image_file in enumerate(images):
11         image = transform_image(image_file)
12         data[i] = image
13     return data
14
15 train = prep_data(train_images)
16 val = prep_data(val_images)
17 test = prep_data(test_images)
```

Código A.3. Conversión de imágenes a tensor tridimensional

En este proceso de redimensión se aplica una operación clave: **la interpolación**. La interpolación es un proceso que se aplica en imágenes cuando estas sufren transformaciones para que se pierda la menor información posible. Se basa en la aproximación de valores no conocidos a partir de valores conocidos. En este caso, los valores conocidos son los píxeles y los valores a aproximar aparecen cuando se aplica cualquier transformación a las imágenes, como puede ser una redimensión. Los nuevos píxeles de la imagen deben ser aproximados para que la imagen no pierda sus características, adecuándose en función de sus vecinos. Se puede observar un ejemplo de redimensión en la Figura A.1 con una imagen de 400×400.

(a) Foto original

(b) Foto redimensionada con interpolación cúbica

(c) Foto redimensionada con interpolación básica

Figura A.1. Aplicación de interpolación en imagen redimensionada

Al redimensionar la imagen de la Figura A.1a a 50×50 con interpolación, como la Figura A.1b, se mantiene la forma nítida del personaje y las formas de las prendas y cara conforme a la original. Con todo esto, la imagen interpolada parece la imagen original pero borrosa. Mientras que si se realiza con una interpolación básica, como en la Figura A.1c, el resultado es más brusco, se notan demasiado los píxeles, no hay punto intermedio entre las prendas, y es menos real, perdiendo información vital de sombreado, formas y rasgos faciales. Se ha elegido la interpolación cúbica ya que, a pesar de que la carga computacional es mayor, ofrece unos resultados mucho más nítidos, aunque **openCV**, librería que permite realizar la operación, ofrece varias opciones: *lanczos*, *linear*, *nearest*, *cubic* y *area*.

Tras la redimensión, el contenido del tensor se debe **normalizar**. Este proceso es muy importante ya que permite la no discriminación entre valores de intensidad del píxel altos y pequeños, ya que pueden sesgar la extracción de información. El valor máximo de la intensidad de un píxel es 255, por lo que la normalización consiste en la división total del tensor por 255, dando lugar a nuevos tensores con valores

comprendidos entre 0 y 1.

```
train = train.astype('float32')/255
val = val.astype('float32')/255
test = test.astype('float32')/255
```

Código A.4. Normalización de los tensores

Una vez convertidas y normalizadas todas las imágenes, el proceso se repite para cada imagen de los tres conjuntos creados, creando un conglomerado de tensores 3D almacenados en: `train`, `val` y `test`.

A.2. Generadores y *data augmentation*

Antes de transmitir los tensores normalizados a la red, se crean unos **generadores de imágenes** a partir de los propios tensores calculados previamente. Un generador es una estructura necesaria para el entrenamiento de la red que genera lotes de imágenes aplicando transformaciones en tiempo real. Estas modificaciones son vitales para que la red de neuronas produzca un buen resultado, aunque son opcionales. La aplicación de variaciones a las imágenes produciendo como resultado nuevos datos es conocido como ***data augmentation***. En este caso, estas transformaciones vienen aplicadas como rotaciones, translaciones o emborronado de imágenes. Keras, con sus generadores, ofrece varias opciones:

- `rotation_range`: rango de grados de rotación que se aplica aleatoriamente.
- `width_shift_range` y `height_shift_range`: número de píxeles o proporción relativa a las dimensiones del patrón con las que desplazar la imagen.
- `brightness_range`: rango de valores para aplicar un determinado brillo.
- `shear_range`: rango de valores relativo al recorte aleatorio a aplicar a la imagen.
- `zoom_range`: valor para aplicar un zoom aleatorio.
- `vertical_flip` y `horizontal_flip`: valor booleano para aplicar un volteo vertical u horizontal respectivamente.

En este proceso, las transformaciones pueden llegar a ser radicales, por lo que es conveniente no aplicar *muchas* de ellas, ya que afectaría a la relevancia de la imagen. Con un *data augmentation* moderado se pueden conseguir resultados bastante

buenos. Establecidas algunas de las opciones que ofrece Keras para el *data augmentation*, son estas las que se van a aplicar a cada imagen que pase por el generador.

En el Código A.5, se aplica `ImageDataGenerator`, el generador que va a aplicar las transformaciones expuestas, que quedan guardadas en `train_gen`. Es con la función `.flow` del generador con la que se va a leer de los tensores `train` y `train_labels` (o `val` y `val_labels`) previamente creados y se van a ir aplicando las consecuentes modificaciones aleatorias definidas en `train_gen`. Se utilizan esos dos tensores para saber a qué clase pertenece cada nueva imagen generada con el *data augmentation*.

```
train_gen = ImageDataGenerator(rotation_range=90,
    horizontal_flip=True)
val_gen = ImageDataGenerator()

train_generator = train_gen.flow(train,train_labels,
    batch_size=NTRAIN)
val_generator = val_gen.flow(val,val_labels,batch_size=NVAL)
```

Código A.5. *Data augmentation* y generadores

Ya que al aplicar el *data augmentation* se parte de la imagen original, las imágenes generadas heredan la clase de la original, clasificándose automáticamente. Gracias al *data augmentation* se añaden nuevas imágenes con ruido que ayudan a la generalización de la red y se crean miles de datos para que esta tenga muchos datos con los que trabajar.

Después de que se hayan generado `NTRAIN` y `NVAL` tensores nuevos mediante *data augmentation*, estos se van guardando por lotes en `train_generator` y `val_generator` respectivamente, los cuales serán los utilizados en el modelo para entrenar. El número de imágenes por lote, `batch_size`, se calcula mediante la fórmula:

$$I_L \leq \texttt{batch\_size} \times P_L \tag{1}$$

Donde L es el conjunto de entrenamiento o validación, I_L son las imágenes de L y P_L es el número total de lotes de imágenes a extraer del generador de L antes de finalizar un ciclo de entrenamiento. Siempre se busca que el producto sea mayor o igual a I_L porque de esta forma se crea espacio extra en los lotes para que todas las imágenes del conjunto L sirvan para el entrenamiento y que no se quede ninguna sin utilizar. Si el producto es menor que I_L, se cogerían menos imágenes por lote, y, por tanto, alguna imagen podría quedarse fuera del entrenamiento.

Por último, resaltar que los únicos conjuntos que se han de utilizar para el entrenamiento son el conjunto de entrenamiento y de validación, por lo que será únicamente a estos dos conjuntos los que se apliquen los generadores, con una excepción: solo se realizará el *data augmentation* al conjunto de entrenamiento. Esto se debe a que las imágenes del conjunto de validación sirven para evaluar el correcto entrenamiento de la red, por lo que deben ser imágenes reales, sin ninguna modificación. Lo mismo les ocurre a las imágenes del conjunto de prueba: se utilizan solamente para evaluar la red una vez a producido un modelo, por lo que ni se necesita un generador, ni las imágenes deben sufrir alteraciones.

A.3. Capas y estructura del modelo

En esta sección se explica la creación de la estructura interna de la red. Keras utiliza una interfaz basada en capas como en la Figura A.2.

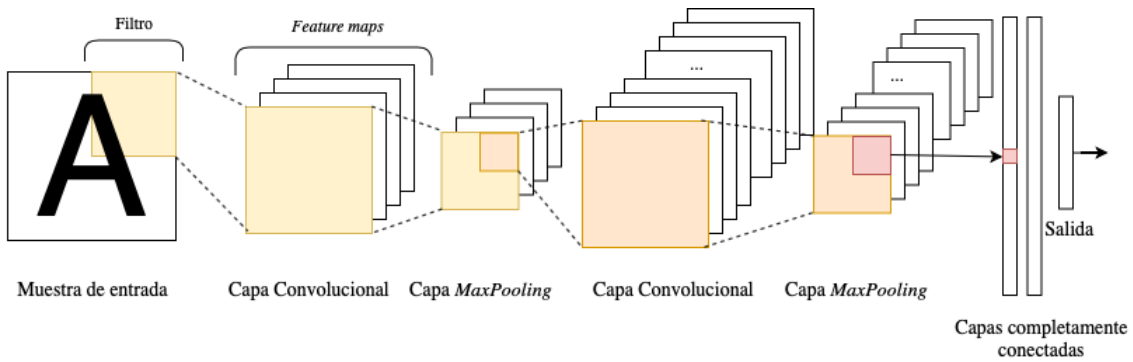


Figura A.2. Estructura básica de una red de neuronas convolucional

Por lo que existen varias capas: `Conv2D` para la operación de convolución, `MaxPooling` para el *pooling*, `Dense` como capa oculta del perceptrón multicapa anclado al final de la base convolucional, y `Flatten`, una capa especial que convierte los *feature maps* a un vector 1D para poder pasarlos al perceptrón multicapa. A continuación, se explican en detalle los parámetros utilizados de cada una de estas capas y su cohesión con el modelo. Un ejemplo de estructura de base convolucional construida con Keras se puede ver en el Código A.6.

```
model = models.Sequential()
model.add(layers.Conv2D(filters=32, kernel_size=(3,3),
    input_shape=(WIDTH, HEIGHT, CHANNELS),
    activation='relu'))
model.add(layers.MaxPool2D((2,2)))
```

```
6  model.add(layers.Conv2D(filters=64, kernel_size=(3,3),
7      activation='relu'))
8  model.add(layers.MaxPool2D((2,2)))
```

Código A.6. Estructura básica de la base convolucional

Se crean pares de capas de convolución-*pooling*, aunque es perfectamente plausible utilizar combinaciones de convolución-convolución-*pooling* o convolución-*pooling*-*pooling*. La capa de convolución viene dada por `Conv2D` con los siguientes parámetros:

- `filters`: número de *feature maps* resultantes de la operación de convolución sobre un campo receptivo.
- `kernel_size`: dimensión $W_F \times H_F \times C_F$ de los filtros a aplicar sobre el patrón de entrada $W_I \times H_I \times C_I$ siempre manteniendo que $W_F \leq W_I$, $H_F \leq H_I$ & $C_F = C_I$.
- `strides`: este parámetro controla cómo el filtro aplicado se traslada sobre el patrón de entrada. Como ejemplo se pone la Figura A.3, en donde se observa la diferencia entre un par de valores de `strides` aplicados a un filtro de 2×2 y su deslizamiento sobre un campo receptivo de 4×4.

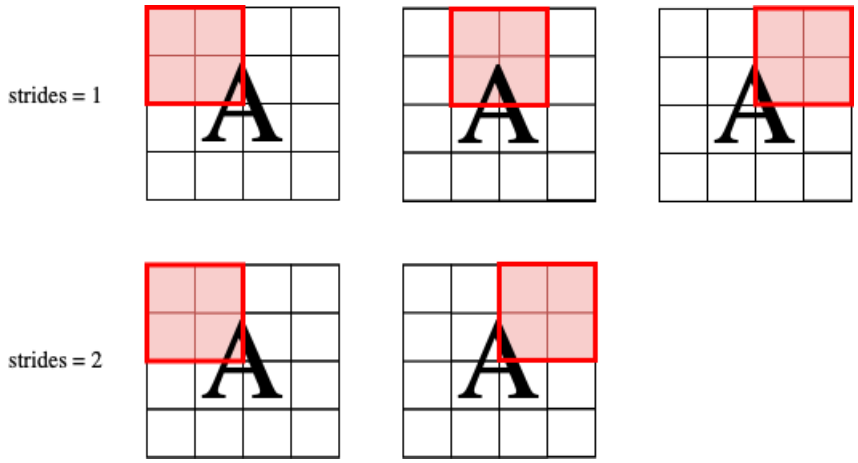


Figura A.3. Aplicación de `strides` con diferentes valores sobre patrón

El valor predeterminado es `strides = 1`.

- `padding`: consiste en aplicar un rellenado artificial en el campo receptivo. Por ejemplo, si se tiene un patrón de entrada de 4×4 y se aplica un filtro de 4×4, este solo se podrá aplicar una vez. Esto puede provocar la pérdida de información ya que se pueden haber perdido detalles al no realizar la traslación del filtro sobre la imagen.

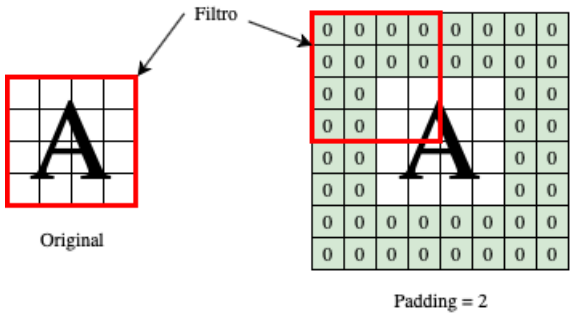


Figura A.4. Diferenciación entre patrones con la aplicación de *padding*

El `padding` ayuda a no perder esa información recubriendo el patrón de entrada con píxeles artificiales (Figura A.4). Si se recubre el patrón con un margen de 2 píxeles, este pasa a tener dimensión 8×8, pudiendo aplicar el mismo filtro 5×5 veces. De esta forma, al tener más *feature maps* se extrae mucha más información de un patrón dado. Si no se especifica su valor, el parámetro no se aplica.

- `data_format`: parámetro que sirve para definir si la dimensión de los patrones de entrada viene la forma $C_I \times W_I \times H_I$ o $W_I \times H_I \times C_I$.
- `activation`: la función de activación a utilizar en cada neurona de la capa de convolución. Esta función decide en base a los parámetros de las neuronas, si estas pasan la información obtenida hacia las demás capas o no. Existen varias funciones aplicables como `ReLU, sigmoid, tanh, softmax` o `linear`. La sigmoidea se solía utilizar como función de activación de capas ocultas, pero quedó demostrado por Krizhevsky que ReLU ofrece una mejor respuesta en estas capas debido a su capacidad de convergencia y rapidez [2].
- `input_shape`: determina las dimensiones del patrón de entrada tal que $W_I \times H_I \times C_I$. Estas dimensiones están definidas en los lotes de tensores de los generadores del Código A.5. Este parámetro solo es necesario en la primera capa convolucional de la red (la de entrada). Para las demás capas convolucionales de la red, Keras calcula automáticamente cual será la dimensión del patrón necesaria para trabajar.

Posteriormente se aplican a los *feature maps* extraídos la operación de *pooling*, definida como `MaxPool2D`. En esta capa se va a especificar la dimensión de la ventana de *pooling* con el parámetro `pool_size`, que en este caso toma el valor de 2×2. También se tiene la opción de poner como parámetros `strides`, `padding` y `data_format`, con el mismo significado que para las capas de convolución a excepción de `strides`. Si no se especifica este parámetro, su valor por defecto será igual al `pool_size`, es

decir, como `pool_size` = `strides` = 2, la ventana no se solapará al trasladarse por el patrón.

Como se ha explicado previamente, ya que las redes de neuronas convolucionales son capaces de aprender patrones espaciales de información, son necesarias una o varias capas completamente conectadas para poder clasificar la información.

```
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(NUM, activation='softmax'))
```

Código A.7. Estructura de las capas completamente conectadas

La información resultante de la última operación de *max pooling* es un tensor de 3 dimensiones, por lo que no se puede trasladar *directamente* a una capa completamente conectada. Estas capas necesitan de un tensor unidimensional para poder trabajar, y para convertir el tensor 3D a 1D se utiliza la capa `Flatten`.

Para definir una capa completamente conectada se utiliza la capa `Dense`. Como parámetros tiene: las `units`, que especifican la dimensionalidad del espacio neuronal (establecen el número de neuronas en la capa) y, por otra parte, el parámetro de `activation` con el mismo significado explicado anteriormente.

Se pueden concatenar tantas capas `Dense` como se desee, pero con una condición: la capa de salida debe tener la misma cantidad de `units` que clases tiene el conjunto estudiado, `NUM`, y, generalmente, con una función de activación tipo `softmax` o `sigmoid`. El tipo función de activación utilizada en la capa de salida viene condicionada por el número de neuronas.

- La función `softmax` establece un rango de valores entre 0 y 1 tal que $[0, 1]$, como una función logística. En el caso de tener $R_L \geq 2$, siendo R_L el número de clases del conjunto, para cada clase se decide un valor V_t dentro de $[0, 1]$ de tal forma que $\sum_{t=0}^{R_L} V_t = 1$. Esto se puede interpretar como la probabilidad de que una imagen pertenezca a una determinada clase. Por sus propiedades esta función se utiliza para problemas de clasificación multiclase.
- La función `sigmoid` es otra modificación de la función logísitica pero que se suele utilizar en problemas de clasificación binarios, ya que pese a tener el mismo funcionamiento que la función `softmax`, esta no requiere que la suma de los valores por clase sea 1.

Todos los conjuntos de datos que aquí se estudian son multiclase, por lo que en todos los experimentos se utilizará la función de activación `softmax` en la capa de salida.

A.4. Compilación y entrenamiento

Una vez establecida la estructura de la red de neuronas convolucional y preparados todos los lotes de imágenes en los generadores, solo queda configurar el modelo para su entrenamiento y ejecutarlo.

```
model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
        loss='binary_crossentropy',
        metrics=['accuracy'])
```

Código A.8. Compilación del modelo

Con la función `compile` se configura el modelo creado para su entrenamiento y se establecen los siguientes parámetros requeridos: la función de pérdida (`loss`), el optimizador del gradiente (`optimizer`) y las métricas a registrar (`metrics`).

El entrenamiento de una red se basa en la actualización de sus pesos. Esta modificación es proporcional al paso del gradiente para encontrar el mínimo de la función de error, siendo el factor de proporcionalidad la razón de aprendizaje (`lr`). Si la razón es grande, las modificaciones de los pesos son más bruscas y la función de error puede oscilar alrededor del mínimo sin llegar nunca a él, aunque la convergencia es mucho más rápida. Mientras que si la razón es pequeña, los desplazamientos son más suaves, implicando una convergencia más lenta, y en muchas ocasiones más eficaz, incluso llegando al mínimo.

Encontrar el valor idóneo de la tasa de aprendizaje es una tarea muy complicada y por eso se obtiene la ayuda de los **optimizadores de gradiente**. Con estos optimizadores se intenta aproximar el valor de la razón de aprendizaje en cada ciclo de tal manera que el gradiente encuentre el mínimo de la función de error rápida y eficazmente, a la vez que los pesos se actualicen de manera adecuada. Keras ofrece varios optimizadores `stochastic gradient descent (SGD)` [3], `AdaGrad` [4], `RMSProp` [5], `AdaDelta` [6] y `Adam` [7] pero no hay una regla genérica que establezca cual es el mejor para optimizar el gradiente. En los experimentos se utilizan los optimizadores `RMSProp` y `AdaDelta`[13].

[13] Para información concreta acerca de las peculiaridades y diferencias de los optimizadores puede visitarse la entrada del equipo CS231N de la universidad de Stanford [8].

La función de pérdida o de error es la que produce una red de neuronas cuando esta clasifica un patrón. En los primeros pasos del entrenamiento, el error puede ser muy grande y eso se va a remediar a lo largo del entrenamiento gracias al descenso de gradiente. Keras ofrece varias funciones para representar el error entre la clase producida y la clase real de un patrón, como por ejemplo: `mean_squared_error`, `mean_absolute_error`, `categorical_crossentropy` o `binary_crossentropy`. En los experimentos para los distintos conjuntos de datos se utiliza, preferiblemente `categorical_crossentropy` y `binary_crossentropy` dependiendo si el problema es de clasificación multiclase o binaria respectivamente, en vez del `mse` o el `mae` [9].

Por último, `metrics` es una función que se usa para juzgar la eficacia del modelo creado. Estas funciones no se utilizan durante el entrenamiento, simplemente sirven para la visualización del progreso del modelo a medida que el entrenamiento se ejecuta. En todos los experimentos se selecciona la métrica `accuracy` que depende del tipo de función de error: si es `categorical_crossentropy`, la métrica será `categorical_accuracy` y si es `binary_crossentropy` será `binary_accuracy`. El uso de una métrica u otra viene determinado, al igual que para la función de error, por el número de clases del conjunto: si $R_L = 2$, se utiliza `binary_crossentropy` y si $R_L \geq 2$, `categorical_crossentropy`.

Ya configurado todo el modelo para su aprendizaje, falta una última función para poder ejecutarlo: `fit_generator`.

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch = TRAIN_STEPS,
    epochs = EPOCHS,
    validation_data = val_generator,
    validation_steps = VAL_STEPS,
    verbose = 2)
```

Código A.9. Trasmisión de los lotes de imágenes al modelo

La función `fit_generator` es la encargada de suministrar los lotes de imágenes de los generadores al modelo configurado. Por cada ciclo o `EPOCH` del entrenamiento se van a extraer `TRAIN_STEPS` y `VAL_STEPS` lotes de imágenes de `train_generator` y `val_generator` (véase ecuación 1), para poder aprender patrones y verificar el correcto funcionamiento de la red. El número de ciclos totales, `EPOCHS`, es un valor arbitrario que se puede ir ajustando en función de los experimentos. Y, por su parte, el parámetro `verbose` simplemente especifica el nivel de detalle a mostrar por

pantalla mientras el entrenamiento se esté ejecutando.

Una vez que las imágenes estén procesadas y convertidas, que los distintos generadores creen los lotes de imágenes, aplicando el *data augmentation* al conjunto de entrenamiento, que la estructura del modelo esté creada y configurada, y que la duración del entrenamiento esté decidida, se podrá ejecutar un experimento con una red neuronal convolucional con Keras.

A.5. Métodos de evaluación

Cuando el modelo haya terminado de entrenar, este nos devolverá un historial basado en la métrica establecida en la configuración del modelo. Como se ha comentado anteriormente, en los experimentos se han utilizado dos métricas de apoyo a la tasa de acierto del modelo: la matriz de confusión y una lista con todas las imágenes del conjunto con su clase correcta y la clase predicha.

En primer lugar, después de haber entrenado, el modelo se guarda en un fichero H5 con la función `model.save`. Keras automáticamente guarda la estructura del modelo junto con los pesos óptimos conseguidos para su utilización posterior. En segundo lugar, las imágenes que definen las matrices de confusión tanto de validación como de prueba se crean con el código de `scikit-learn` destinado a crear una matriz de confusión [10]. Para poder crear la imagen, se necesita crear una matriz de confusión. Se consigue con la librería `sklearn`.

```
1 Y_pred = model.predict(test, verbose=2)
2 y_pred = np.argmax(Y_pred, axis=1)
3
4 cnf = metrics.confusion_matrix(
5     np.argmax(test_labels, axis=1),
6     y_pred)
```

Código A.10. Creación de la matriz de confusión

La matriz de confusión representa el contraste entre clases de tal manera que:

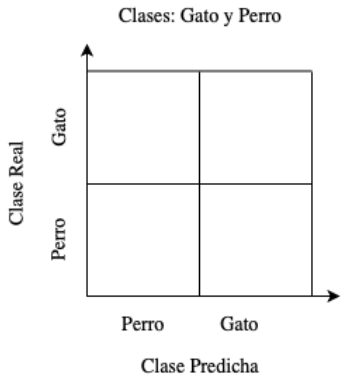


Figura A.5. Matriz de confusión

Como se requiere saber únicamente la clase de todas las imágenes del conjunto para crear esta matriz, es necesario crear unos *arrys* con tantas posiciones como imágenes tenga el conjunto, teniendo en cada una de ellas un número asociado a la clase que pertenece la imagen: 0 para "perro" y 1 para "gato". Por lo tanto, se necesitan dos *arrays* 1D: un *array* de las clases predichas por la red y otro de las clases reales de cada imagen en el conjunto deseado, en este caso, el de prueba. El primer *array* se crea con la función `predict` que devuelve un *array* de *arrays* con las probabilidades asociadas a cada imagen del conjunto de prueba (`softmax`), denotado por `Y_pred`. Ya que se requiere un *array* unidimensional, se realiza la operación `argmax` para escoger la posición del mayor valor de cada uno de esos *arrays* de probabilidades. Para entender mejor esta operación se estudia la Figura A.6.

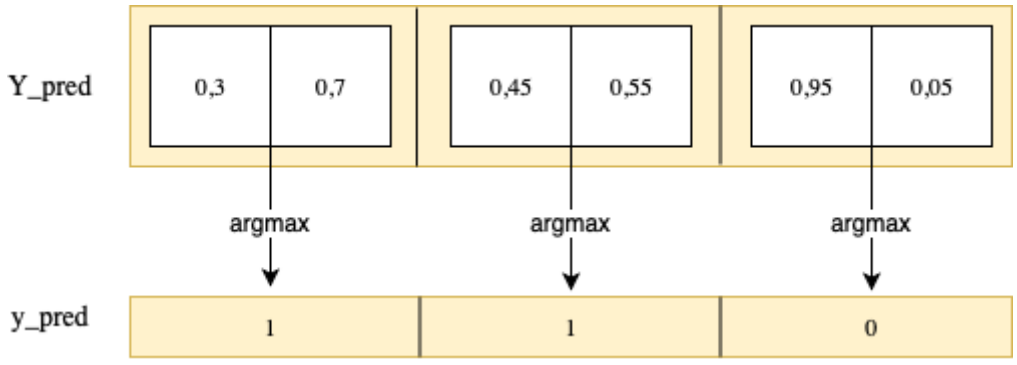


Figura A.6. Visualización de la operación `argmax`

En la figura se presenta un *array* de salida de red (`Y_pred`) que se ha evaluado con 3 imágenes, y en cada posición hay un *array* de dos posiciones con las probabilidades de que cada imagen del conjunto pertenezca a una u otra clase (clasificación binaria). Para convertir el *array* de salida en uno unidimensional acorde a las necesidades de la matriz de confusión, se escoge de cada *array* de probabilidades la posición del valor más alto (la clase ganadora) y se queda como único valor, obteniendo así un

array 1D con las posiciones de las clases ganadoras predichas de todas las imágenes del conjunto (`y_pred`). Por lo tanto, el número asociado a cada clase es la posición del *array* de salida de la clase ganadora. Para el *array* de las clases extraídas en el Código A.2, `test_labels`, se realiza el mismo procedimiento y, con el resultado de esta operación y `y_pred` se crea la matriz de confusión. Este proceso se realiza de igual manera con el conjunto de validación.

En tercer lugar, para evaluar el modelo con los distintos conjuntos creados, se utiliza la función `evaluate`. La función evalúa cada imagen de cada conjunto y la compara con su clase real, devolviendo el valor de la función de acierto.

```
1 score = model.evaluate(test, test_labels, verbose=2)
2 score_val = model.evaluate(val, val_labels, verbose=2)
3 score_train = model.evaluate(train, train_labels, verbose=2)
```

Código A.11. Cálculo de la tasa de acierto

Se debe atender a que se evalúa en función de las imágenes normalizadas y no de las imágenes de los generadores (véase Código A.4). Como se quiere observar cómo influye cada conjunto creado en el modelo, se evalúan los tres conjuntos, pero solo se creará un apoyo visual al progreso del entrenamiento entre el conjunto de entrenamiento y validación. La gráfica se crea con ayuda de `matplotlib.pyplot` y se construye de tal manera que se muestre el valor de la función de acierto para ambos conjuntos por cada `epoch` o ciclo de entrenamiento.

Como medida preventiva y para tener los datos del acierto y de las matrices de confusión más detallados, se crea un fichero de texto que contiene: las tres tasas de acierto `score, score_val` y `score_train`, junto con las matrices de confusión en texto plano. De esta manera, al hacer anotaciones de los experimentos resulta mucho más llevadero consultar este fichero que las imágenes.

Por último, se crea el archivo que contiene el contraste de clases por imagen. Este archivo cumple la misma función que la matriz de confusión pero con mucho más detalle. La matriz de confusión hace un recuento general de las imágenes mal o bien clasificadas, mientras que en este archivo se muestran todas las imágenes del conjunto con su clase real y su clase predicha por el modelo. De esta forma, se pueden evaluar casos puntuales y anómalos y tratar de buscar una solución a la mala clasificación de estos.

Referencias

[1] F. Chollet, *Deep Learning with Python*. Manning, 2018.

[2] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», en *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ép. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, págs. 1097-1105.

[3] J. Kiefer y J. Wolfowitz, «Stochastic Estimation of the Maximum of a Regression Function», *The Annals of Mathematical Statistics*, vol. 23, n.º 3, págs. 462-466, 1952.

[4] J. Duchi, E. Hazan e Y. Singer, «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization», *Journal of Machine Learning Research*, vol. 12, págs. 2121-2159, 2011.

[5] T. Tieleman y G. Hinton, «Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude», *COURSERA: Neural Networks for Machine Learning*, 2012.

[6] M. D. Zeiler, «ADADELTA: An Adaptive Learning Rate Method», *arXiv:1212.5701 [cs.LG]*, 2012.

[7] D. P. Kingma y J. Ba, «Adam: A method for stochastic optimization», *arXiv:1412.6980 [cs.LG]*, 2014.

[8] A. Karpathy, *CS231n Convolutional Neural Networks for Visual Recognition*, `http://cs231n.github.io/neural-networks-3/#update`, accedido por última vez el 24 de abril de 2019.

[9] J. D. McCaffrey, *Why You Should Use Cross-Entropy Error Instead Of Classification Error Or Mean Squared Error For Neural Network Classifier Training*, `https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/`, accedido por última vez el 24 de abril de 2019.

[10] E. de Desarrollo de scikit-learn, *Confusion Matrix*, `https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py`, accedido por última vez el 25 de abril de 2019.

B. ANEXO II: RECOPILACIÓN DE EXPERIMENTOS

En este anexo se expone la recopilación de todos los experimentos realizados con los distintos dominios: *Dogs vs. Cats*, FERC, CIFAR10, *Katakana* y *Hiragana*.

Experimentación	
Dogs vs. Cats	
Nombre	Porcentaje de acierto
0004-CvD-VGG16-0001	$\approx 68\,\%$
0005-CvD-VGG16-0002	69,56 %
0006-CvD-VGG16-0003	$\approx 80\,\%$
0007-CvD-VGG16-0004	$\approx 83\,\%$
0008-CvD-VGG16-0005	$\approx 85\,\%$
0009-CvD-VGG16-0006	$\approx 85\,\%$
0010-CvD-VGG16-0007	$\approx 85\,\%$
0011-CvD-VGG16-0008	$\approx 90\,\%$
0012-CvD-VGG16-0009	87,17 %
0013-CvD-VGG16-0010	90,37 %
0014-CvD-VGG16-0011	88,39 %
0015-CvD-VGG16-0012	90,26 %
0016-CvD-VGG16-0013	91,17 %
0017-CvD-VGG16-0014	90,93 %
FERC	
Nombre	Porcentaje de acierto
Prueba 1	$\approx 97\,\%$
Prueba 2	$\approx 97\,\%$
Prueba 3	$\approx 97\,\%$
Prueba 4	$\approx 97\,\%$
Prueba 5	$\approx 97\,\%$
Prueba 6	$\approx 97\,\%$
Prueba 7	$\approx 97\,\%$
Prueba 8	$\approx 95\,\%$
Prueba 9	$\approx 95\,\%$
Prueba 10	$\approx 96\,\%$
Prueba 11	$\approx 96\,\%$

Prueba 12	$\approx 97\,\%$
Prueba 13	$\approx 97,5\,\%$
Prueba 14	$\approx 97\,\%$
Prueba 15	$\approx 98\,\%$
Prueba 16	$\approx 97\,\%$
0018-FERC-VGG16-0017	89,30 %
CIFAR10	
Nombre	Porcentaje de acierto
0020-CIFAR-VGG16-0003	78,78 %
0021-CIFAR-VGG16-0004	79,91 %
0022-CIFAR-VGG16-0005	$\approx 81\,\%$
0023-CIFAR-CCMCMCMDD-0006	75,80 %
0024-CIFAR-CCMCMCMDD-0007	$\approx 75\,\%$
0025-CIFAR-CCMCMCMDD-0008	81,40 %
0026-CIFAR-CCMCMCMDD-0009	-
0027-CIFAR-CCMCMCMDD-0010	88,28 %
0028-CIFAR-CCMCMCMDD-0011	85,24 %
0029-CIFAR-CCMCMCMDD-0012	-
0030-CIFAR-CCMCMCMDD-0013	-
Katakana	
Nombre	Porcentaje de acierto
0019-KATA-VGG16-0001	98,01 %
0031-KATA-VGG16-0002	98,67 %
0032-KATA-VGG16-0003	99,14 %
0033-KATA-VGG16-0004	99,26 %
0034-KATA-VGG16-0005	99,18 %
0035-KATA-VGG16-0006	98,99 %
0036-KATA-VGG16-0007	99,15 %
0037-KATA-VGG16-0008	98,94 %
0042-KATA-VGG16-0009	99,21 %
0043-KATA-VGG16-0010	99,18 %
0052-KATA-CMCMCM-0011	98,98 %
0053-KATA-CMCMCMCM-0012	98,65 %
0054-KATA-CCMCMCM-0013	99,11 %
0055-KATA-CCMCCMCM-0014	99,10 %
0056-KATA-CCMCCMCM-0015	99,17 %

0057-KATA-CCMCCMCMCM-0016	98,99 %
0058-KATA-CCCMCCCMCCM-0017	98,80 %
0059-KATA-CCCMCCCMCCM-0018	99,23 %
Hiragana	
Nombre	Porcentaje de acierto
0038-HIRA-VGG16-0001	99,23 %
0039-HIRA-VGG16-0002	99,53 %
0039a-HIRA-VGG16-0002	99,59 %
0039b-HIRA-VGG16-0002	99,56 %
0039c-HIRA-VGG16-0002	99,48 %
0039d-HIRA-VGG16-0002	99,40 %
0039e-HIRA-VGG16-0002	99,29 %
0039f-HIRA-VGG16-0002	99,61 %
0040-HIRA-VGG16-0003	99,18 %
0040a-HIRA-VGG16-0003	99,23 %
0040b-HIRA-VGG16-0003	99,30 %
0040c-HIRA-VGG16-0003	99,28 %
0040d-HIRA-VGG16-0003	99,25 %
0040e-HIRA-VGG16-0003	99,20 %
0040f-HIRA-VGG16-0003	99,36 %
0041-HIRA-VGG16-0004	99,42 %
0044-HIRA-CMCMCM-0005	99,10 %
0045-HIRA-CMCMCMCM-0006	99,18 %
0046-HIRA-CCMCMCM-0007	99,40 %
0047-HIRA-CCMCCMCM-0008	99,51 %
0048-HIRA-CCMCCMCM-0009	99,29 %
0049-HIRA-CCMCCMCMCM-0010	99,37 %
0050-HIRA-CCCMCCCMCCM-0011	99,34 %
0051-HIRA-CCCMCCCMCCM-0012	99,40 %

Tabla B.1. Compilación de los experimentos con diferentes dominios

Como nota adicional, el experimento 0033-KATA-VGG16-0004 pese a ser el que mejor acierto obtiene del conjunto de *katakana*, la adición del *padding* en las imágenes no está implementada en el modelo. Debido a esto, este experimento no se tiene en cuenta para la elección de los modelos a integrar en la aplicación *web*.

C. ANEXO III: REQUISITOS Y CASOS DE PRUEBA

En este anexo se exponen los distintos requisitos de la aplicación *web* ¡Aprende! Japonés.

C.1. Requisitos

Se distinguen dos tipos de requisitos: funcionales y no funcionales.

- Los **requisitos funcionales** son aquellos que definen el comportamiento y las necesidades de la aplicación.
- Los **requisitos no funcionales** son aquellos que definen las capacidades y restricciones de la aplicación.

A continuación, se muestra la plantilla (véase Tabla C.1) de la que se va a hacer uso para la exposición de los distintos requisitos, tanto funcionales como no funcionales. En ella, los campos se definen como:

- **Identificador**: sirve para identificar los distintos requisitos. Vienen dados por un ID: RF (Requisito Funcional) o RNF (Requisito No Funcional) y por un número N de tres dígitos para poder identificar los requisitos por grupo.
- **Prioridad**: define la prioridad para la implementación del requisito en cuestión. Puede tomar tres valores: ALTA, MEDIA o BAJA.
- **Necesidad**: define la necesidad de implantar cierto requisito en la aplicación. Puede tomar tres valores: ALTA (obligación), MEDIA (deseable) o BAJA (opcional).
- **Estabilidad**: define la capacidad de alteración que puede llegar a tomar el requisito en un momento determinado del desarrollo. Puede tomar tres valores: ALTA, MEDIA o BAJA.
- **Verificabilidad**: define si un requisito se puede comprobar en la aplicación en cuestión o no. Puede tomar tres valores: ALTA, MEDIA o BAJA.
- **Autor**: persona creadora del requisito a implementar.
- **Descripción**: breve resumen en el que se explica la funcionalidad o la restricción del requisito.

ID-N			
Prioridad:		Necesidad:	
Estabilidad:		Verificabilidad:	
Autor:			
Descripción:			

Tabla C.1. Plantilla para la definición de requisitos

C.1.1. Requisitos funcionales

Aquí se detallan los requisitos funcionales de la aplicación. Primeramente, se describen los requisitos funcionales dedicados a la **autenticación del usuario**: registro e inicio de sesión del usuario, modificación del nombre y contraseña del usuario, eliminación de la cuenta, cierre de sesión y cambio de idioma de la aplicación.

RF-001			
Prioridad:	MEDIA	Necesidad:	MEDIA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá registrarse con un e-mail, una contraseña y un nombre válidos.			

Tabla C.2. Requisito funcional 1

RF-002			
Prioridad:	MEDIA	Necesidad:	MEDIA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario registrado podrá cambiar su nombre.			

Tabla C.3. Requisito funcional 2

RF-003			
Prioridad:	MEDIA	Necesidad:	MEDIA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario registrado podrá cambiar su contraseña.			

Tabla C.4. Requisito funcional 3

RF-004			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario registrado podrá eliminar su cuenta.			

Tabla C.5. Requisito funcional 4

RF-005			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario registrado podrá cerrar sesión.			

Tabla C.6. Requisito funcional 5

RF-006			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá iniciar sesión con su correo electrónico y contraseña.			

Tabla C.7. Requisito funcional 6

RF-007			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá elegir el idioma de la aplicación: español o inglés.			

Tabla C.8. Requisito funcional 7

Posteriormente, se muestran los requisitos enfocados a la **práctica de los caracteres**: elección del carácter a dibujar y de la secuencia de aparición de los caracteres, evaluación y borrado del carácter dibujado, ayudas al usuario para la comprensión del carácter y *feedback* de la evaluación.

RF-008			
Prioridad:	ALTA	Necesidad:	MEDIA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Usuario		
Descripción:			
El usuario podrá practicar los caracteres disponibles sin necesidad de estar registrado.			

Tabla C.9. Requisito funcional 8

RF-009			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá elegir el carácter (de cualquiera de los dos silabarios) a dibujar en el lienzo de práctica.			

Tabla C.10. Requisito funcional 9

RF-010			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá borrar su dibujo.			

Tabla C.11. Requisito funcional 10

RF-011			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá evaluar su dibujo.			

Tabla C.12. Requisito funcional 11

RF-012			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá observar el carácter, cómo se realizan los trazos del carácter, el número de trazos, la pronunciación y el código unicode de este.			

Tabla C.13. Requisito funcional 12

RF-013			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá elegir el orden en el que van apareciendo los caracteres a practicar: secuencial o aleatoriamente.			

Tabla C.14. Requisito funcional 13

RF-014			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá elegir entre practicar los caracteres o hacer pequeños tests para probar su aprendizaje.			

Tabla C.15. Requisito funcional 14

RF-015			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Usuario		
Descripción:			
El usuario podrá verificar qué carácter ha reconocido la aplicación en base al dibujo realizado.			

Tabla C.16. Requisito funcional 15

Finalmente se muestran los requisitos asociados a la parte de **test de la aplicación *web***: muestra gráfica de los resultados totales, contraste entre las respuestas dadas y las reales y opciones del usuario tras finalizar un test.

RF-016			
Prioridad:	MEDIA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá observar sus avances mediante unos gráficos en la página principal.			

Tabla C.17. Requisito funcional 16

RF-017			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá elegir entre hacer el test del silabario *katakana* o *hiragana*.			

Tabla C.18. Requisito funcional 17

RF-018			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El usuario podrá revisar y comparar las respuestas de los tests tras haber realizado uno.			

Tabla C.19. Requisito funcional 18

RF-019			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Después de realizar un test, el usuario podrá repetir el test del mismo silabario elegido.			

Tabla C.20. Requisito funcional 19

RF-020			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Después de realizar un test, el usuario podrá guardar el resultado antes de volver a la página principal.			

Tabla C.21. Requisito funcional 20

C.1.2. Requisitos no funcionales

Aquí se detallan los requisitos no funcionales de la aplicación. Primero, se muestran los **requisitos base**, detallando las herramientas a utilizar para la administración de datos y usuarios y las restricciones de los campos de texto.

RNF-001			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El administración de los usuarios se realizará mediante Firebase Authentication.			

Tabla C.22. Requisito no funcional 1

RNF-002			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El nombre del usuario tendrá restricciones: mínimo 2 caracteres y no se aceptan símbolos.			

Tabla C.23. Requisito no funcional 2

RNF-003			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
La contraseña del usuario tendrá restricción de caracteres: mínimo 6.			

Tabla C.24. Requisito no funcional 3

RNF-004			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
La administración de los resultados de los tests, así como el idioma seleccionado, se realizará mediante Firebase *Realtime Database*.			

Tabla C.25. Requisito no funcional 4

RNF-005			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Todos los dibujos de cada usuario deberán ser almacenados para la realización de un *corpus* en Firebase *Storage*.			

Tabla C.26. Requisito no funcional 5

RNF-006			
Prioridad:	MEDIA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Al registrarse un usuario, la aplicación mandará un e-mail de confirmación.			

Tabla C.27. Requisito no funcional 6

RNF-007			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Si no se confirma el correo electrónico, los resultados de los tests no se guardarán en ningún caso.			

Tabla C.28. Requisito no funcional 7

RNF-008			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Al eliminar la cuenta, todos los registros de Firebase *Realtime Database* y los datos de usuario de Firebase *Authentication* se eliminarán.			

Tabla C.29. Requisito no funcional 8

RNF-009			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista, Usuario		
Descripción:			
Si el usuario no ha cerrado sesión y sale de la página *web*, su sesión se mantendrá abierta.			

Tabla C.30. Requisito no funcional 9

Seguidamente, se muestran los requisitos relacionados con la **respuesta esperada del modelo de predicción**: Porcentaje de acierto mínimo de los modelos para resultas efectivos para el aprendizaje, tiempo de respuesta de la predicción y tiempo de carga de los modelos.

RNF-010			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Los modelos de reconocimiento implementados tendrán que tener al menos un 99 % de acierto.			

Tabla C.31. Requisito no funcional 10

RNF-011			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El tiempo de predicción de los modelos deberá ser de 1 a 3 segundos.			

Tabla C.32. Requisito no funcional 11

RNF-012			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
El tiempo de carga de los modelos deberá ser de menos de 5 segundos.			

Tabla C.33. Requisito no funcional 12

A continuación, se muestran los requisitos relacionados con la **práctica de los caracteres**: secuencia de los caracteres y comportamiento de la aplicación según la respuesta del modelo.

RNF-013			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
En la práctica, la aparición de los caracteres tiene un orden específico.			

Tabla C.34. Requisito no funcional 13

RNF-014			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
En la práctica, si la predicción del carácter no coincide con el dibujo del usuario, la aplicación permanecerá en el mismo carácter.			

Tabla C.35. Requisito no funcional 14

RNF-015			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
En la práctica, si la predicción del carácter coincide con el dibujo del usuario, la aplicación pasará a mostrar el siguiente carácter según la secuencia seleccionada.			

Tabla C.36. Requisito no funcional 15

RNF-016			
Prioridad:	ALTA	Necesidad:	MEDIA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Al iniciar la práctica de los caracteres, el carácter mostrado será aleatorio.			

Tabla C.37. Requisito no funcional 16

Finalmente, se muestran los requisitos asociados a los **tests de la aplicación**: secuencia de las preguntas, estilo de las preguntas, administración de las respuestas tras la realización de un test y tiempo de respuesta por cada pregunta.

RNF-017			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
En los tests, la secuencia de las preguntas será aleatoria.			

Tabla C.38. Requisito no funcional 17

RNF-018			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Los tests constarán de 10 preguntas, en los que se preguntará por el carácter, dando al usuario únicamente la pronunciación de este.			

Tabla C.39. Requisito no funcional 18

RNF-019			
Prioridad:	ALTA	Necesidad:	MEDIA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Al finalizar un test, si el usuario decide repetir el test, los resultados no se guardarán, pero sí los dibujos realizados.			

Tabla C.40. Requisito no funcional 19

RNF-020			
Prioridad:	ALTA	Necesidad:	ALTA
Estabilidad:	ALTA	Verificabilidad:	ALTA
Autor:	Analista		
Descripción:			
Cada pregunta del test se mostrará un total de 10 segundos para ser respondida.			

Tabla C.41. Requisito no funcional 20

C.2. Casos de prueba

Una vez estudiados los distintos escenarios posibles dentro de la aplicación *web*, a continuación se exponen varios casos de prueba para comprobar el correcto funcionamiento de la *web* en base a los requisitos y escenarios establecidos. Para resumir estos casos de prueba se utiliza una plantilla auxiliar (véase Tabla C.42) en donde se especifica lo siguiente:

- **Identificador**: identificación del caso de prueba con las siglas CP (caso de prueba) seguidas de un número N de tres dígitos.
- **Objetivo**: objetivo que describe el caso de prueba.
- **Precondiciones**: condiciones o estado previo que se tienen que cumplir para poder realizar un caso de prueba específico.
- **Postcondiciones**: condiciones o estado *a posteriori* resultante de la prueba.
- **Requisito(s) asociado(s)**: funcionalidad(es) que se evalúa(n) en el caso de prueba.
- **Evaluación**: resultado de la prueba del requisito, ACEPTADO o RECHAZADO.

CP-N	
Objetivo:	
Precondiciones	Postcondiciones
Requisito(s) asociado(s):	
Evaluación:	

Tabla C.42. Plantilla para la definición de casos de prueba

A continuación se presentan los diferentes casos de prueba de la aplicación *web*, verificando los distintos requisitos establecidos anteriormente. Primero, se exponen los diferentes casos de prueba para la **autenticación y administración de usuario**.

CP-001		
Objetivo:	Registro de un usuario	
Precondiciones		Postcondiciones
Usuario no registrado		Entrada en base de datos, acceso a cuenta y envío de confirmación de e-mail
Requisito(s) asociado(s):	RF-001, RNF-001, RNF-002, RNF-003, RNF-004, RNF-006	
Evaluación:	ACEPTADO	

Tabla C.43. Caso de prueba 1

CP-002		
Objetivo:	Cambio de nombre de usuario	
Precondiciones		Postcondiciones
Usuario con cuenta		Nombre actualizado
Requisito(s) asociado(s):	RF-002, RNF-001	
Evaluación:	ACEPTADO	

Tabla C.44. Caso de prueba 2

CP-003		
Objetivo:	Cambio de contraseña de usuario	
Precondiciones		Postcondiciones
Usuario con cuenta		Contraseña actualizada
Requisito(s) asociado(s):	RF-003, RNF-001	
Evaluación:	ACEPTADO	

Tabla C.45. Caso de prueba 3

CP-004		
Objetivo:	Eliminación de la cuenta del usuario	
Precondiciones		Postcondiciones
Usuario con cuenta		Borrado del usuario de la base de datos y de los registros de usuario
Requisito(s) asociado(s):	RF-004, RNF-001, RNF-004, RNF-008	
Evaluación:	ACEPTADO	

Tabla C.46. Caso de prueba 4

CP-005		
Objetivo:	Cierre de sesión	
Precondiciones		Postcondiciones
Usuario con cuenta		Cierre de acceso a la aplicación
Requisito(s) asociado(s):	RF-005, RNF-001, RNF-009	
Evaluación:	ACEPTADO	

Tabla C.47. Caso de prueba 5

CP-006		
Objetivo:	Inicio de sesión de usuario	
Precondiciones		Postcondiciones
Usuario con cuenta		Acceso a la aplicación y carga de los datos
Requisito(s) asociado(s):	RF-006, RNF-001, RNF-002, RNF-003, RNF-004, RNF-009	
Evaluación:	ACEPTADO	

Tabla C.48. Caso de prueba 6

CP-007		
Objetivo:	Cambio de idioma de la aplicación	
Precondiciones		Postcondiciones
Usuario no registrado o usuario con cuenta		Actualización de campos de base de datos y cambio de idioma
Requisito(s) asociado(s):	RF-007, RNF-004	
Evaluación:	ACEPTADO	

Tabla C.49. Caso de prueba 7

A continuación, se presentan los casos de prueba relacionados con la **práctica de los caracteres**, verificando los requisitos asociados.

CP-008		
Objetivo:	Acceso a la práctica sin registro	
Precondiciones		Postcondiciones
Usuario no registrado		Carga del modelo
Requisito(s) asociado(s):	RF-008, RNF-012, RNF-016	
Evaluación:	ACEPTADO	

Tabla C.50. Caso de prueba 8

CP-009		
Objetivo:	Elección del carácter en la práctica	
Precondiciones		Postcondiciones
Usuario no registrado o usuario con cuenta, página de prácticas y modelo cargado		Cambio del carácter y su información
Requisito(s) asociado(s):	RF-009, RF-012, RNF-012, RNF-016	
Evaluación:	ACEPTADO	

Tabla C.51. Caso de prueba 9

CP-010		
Objetivo:	Borrado de dibujo del carácter	
Precondiciones		Postcondiciones
Usuario no registrado o usuario con cuenta y modelo cargado		Borrado del lienzo
Requisito(s) asociado(s):	RF-010, RNF-012, RNF-016	
Evaluación:	ACEPTADO	

Tabla C.52. Caso de prueba 10

CP-011		
Objetivo:	Evaluación correcta de dibujo del carácter	
Precondiciones		Postcondiciones
Usuario no registrado o usuario con cuenta y modelo cargado		*Feedback* de la evaluación y actualización de la información del carácter
Requisito(s) asociado(s):	RF-011, RF-015, RNF-005, RNF-010, RNF-011, RNF-012, RNF-013, RNF-015	
Evaluación:	ACEPTADO	

Tabla C.53. Caso de prueba 11

CP-012		
Objetivo:	Evaluación incorrecta de dibujo del carácter	
Precondiciones		Postcondiciones
Usuario no registrado o usuario con cuenta y modelo cargado		*Feedback* de la evaluación
Requisito(s) asociado(s):	RF-011, RF-015, RNF-005, RNF-010, RNF-011, RNF-012, RNF-014	
Evaluación:	ACEPTADO	

Tabla C.54. Caso de prueba 12

CP-013		
Objetivo:	Cambio de orden de la aparición de los caracteres en la práctica	
Precondiciones		Postcondiciones
Usuario no registrado o usuario con cuenta y carga del modelo		
Requisito(s) asociado(s):	RF-013, RNF-012, RNF-013, RNF-016	
Evaluación:	ACEPTADO	

Tabla C.55. Caso de prueba 13

Finalmente, se muestran los casos de prueba relacionados con la **realización de tests**, verificando los requisitos asociados.

CP-014		
Objetivo:	Elección de aprendizaje: práctica o test	
Precondiciones		Postcondiciones
Usuario con cuenta		Acceso a la página de práctica o test
Requisito(s) asociado(s):	RF-014, RF-017	
Evaluación:	ACEPTADO	

Tabla C.56. Caso de prueba 14

CP-015		
Objetivo:	Visualización de los resultados de los tests	
Precondiciones		Postcondiciones
Usuario con cuenta, e-mail confirmado y, por lo menos, un test realizado		Gráficos actualizados
Requisito(s) asociado(s):	RF-016, RNF-004, RNF-005, RNF-007	
Evaluación:	ACEPTADO	

Tabla C.57. Caso de prueba 15

CP-016		
Objetivo:	Realización de test, comparación de resultados y guardado de estos	
Precondiciones		Postcondiciones
Usuario con cuenta, e-mail de confirmación aceptado		Almacenamiento de los resultados, actualización de la base de datos y almacenamiento de las imágenes
Requisito(s) asociado(s):	RF-017, RF-018, RF-020, RNF-004, RNF-005, RNF-007, RNF-010, RNF-011, RNF-012, RNF-017, RNF-018, RNF-020	
Evaluación:	ACEPTADO	

Tabla C.58. Caso de prueba 16

CP-017		
Objetivo:	Realización de test, comparación de resultados y repetición del test	
Precondiciones		Postcondiciones
Usuario con cuenta, e-mail de confirmación aceptado		Almacenamiento de las imágenes y comienzo de un nuevo test
Requisito(s) asociado(s):	RF-018, RF-019, RNF-005, RNF-010, RNF-011, RNF-012, RNF-017, RNF-018, RNF-019, RNF-020	
Evaluación:	ACEPTADO	

Tabla C.59. Caso de prueba 17