Detección de imágenes alteradas mediante el uso de un *hash* perceptual basado en CNNs

Jorge Froilán Giménez Pérez

Introducción– Con el reciente incremento en la distribución de *fake news* e imágenes ilegales/indeseadas a través de las redes sociales y las aplicaciones de mensajería ha quedado patente que técnicas simples para detectar este tipo de contenido ya no son suficientes a día de hoy. Este artículo introduce y estudia la viabilidad de un sistema basado en redes neuronales convolucionales siamesas y una función de *hash* que permita ser entrenado en un conjunto de imágenes a vetar. Con el objetivo de que sea capaz de detectar estas imágenes aún habiendo sido ligeramente modificadas con la intención de evitar el veto y realizar esta función sin que el usuario del sistema tenga que tomar posesión de dichas imágenes.
Se introducen las distintas fases de la generación de datos necesarios para poder inicializar el sistema así como la estructura del mismo en detalle, resaltando aquellas partes en las que se puede sustituir lo usado en este estudio por una generalización que sea más adecuada a la aplicación en la que se pretende implementar este sistema.
Finalmente, se introducen los resultados obtenidos y se concluye mediante ellos que el sistema propuesto puede ser utilizado en un caso real dado que las distribuciones de distancias entre los conjuntos de imágenes así lo indican y se introducen los pasos que se podrían seguir para ello.

Palabras clave– imágenes, detección, visión por computador, redes neuronales simaesas, redes neuronales convolucionales, SNN, CNN, CSNN, hash, hash neural.

Abstract– With the recent urge on the distribution of fake news and illegal/damaging images through social media and messaging apps it has been proved that naive approaches to ban this kind of content may not be enough in today's world.
This article aims to introduce and study the viability of a siamese convolutional neural networks based system and a hash function that enable it to be trained over a set of images that must be banned. The system is capable of detecting these images even if they're slightly modified to avoid the ban and also does not require the user implementing it to take possession of the images it works on.
The different phases necessary to generate the data that bootstraps the system are introduced along with a detailed explanation of its structure and modular sections where different approaches can be adapted to better fit a possible production application that may desire to implement the proposed system.
Lastly, the obtained results are introduced and discussed and the viability of the proposed system is concluded give that the observed results show that the distributions between the two sets of images can be differentiated. Also the possible steps to follow if the system wants to be implemented in a production application are discussed.

Keywords– images, detection, computer vision, siamese neural networks, convolutional neural networks, SNN, CNN, CSNN, hash, neural hash.

1 OBJETIVO

El objetivo principal de este proyecto es introducir y estudiar la viabilidad de un sistema robusto de detección de imágenes vetadas y que además no requiera de la toma de posesión de las imágenes a procesar. El sistema propuesto se centra en el uso de redes neuronales convolucionales siamesas[2][3] para conseguir el objetivo de la robustez a modificaciones y en el uso de un *hash* sobre las características extraídas de las imágenes sin la necesidad de tener que tomar posesión de las imágenes que se quieren detectar.

1.1. Detección robusta a modificaciones de imágenes vetadas

El primer objetivo que el sistema introducido persigue y que es esencial para el resto es, que el sistema sea capaz de detectar si una imagen que se le proporciona como entrada se encuentra en el conjunto de imágenes que se pretenden vetar. Esto se consigue gracias a la extracción de características de las imágenes. Se detalla más en el apartado 1.2.
La parte más importante de este objetivo principal es que imágenes que han sido creadas mediante la modificación de alguna de las imágenes dentro del conjunto a vetar sean detectadas de todas formas.
Es decir, por ejemplo: Si una empresa que provee de una aplicación de mensajería quiere evitar que los usuarios compartan una imagen I, los usuarios podrían producir una imagen I' aplicando ligeras modificaciones en la imagen original I para conseguir compartir ese contenido sin ser bloqueados. El sistema propuesto sería capaz de detectar que esta imagen es una modificación de una de las imágenes a bloquear y la aplicación podría actuar en consecuencia.

1.2. No necesidad de la toma posesión de las imágenes a detectar

Otro de los objetivos del sistema propuesto es que no sea necesaria la toma de posesión de las imágenes pertenecientes al conjunto a vetar. Esto es gracias al uso de la generación de un *hash* basado en las características extraídas por el modelo de red neuronal.
Volviendo a un ejemplo: La empresa quiere específicamente bloquear aquellas imágenes que una institución pública determina como inapropiadas y de las que mantiene una base de datos. La institución pública podría entrenar el sistema para esta base de datos y generar los *hashes* pertenecientes a todas esas imágenes y compartirlos con dicha empresa. Posteriormente, la empresa puede ejecutar el sistema para las imágenes que los usuarios comparten y emplear el *hash* generado para compararlo con los que la institución pública les ha proporcionado. De esta manera, la empresa no tiene que tomar posesión ni de la base de datos que la institución mantiene ni de la imagen que el usuario pretende enviar, simplemente trabaja con los *hashes* generados por el sistema (de los que no hay manera de extraer la imagen de nuevo) para detectar posibles envíos de imágenes prohibidas.

2 METODOLOGÍA

En este apartado se procede a explicar la metodología seguida en el desarrollo del proyecto, así como los detalles relevantes respectivos a los datos y el modelo utilizados. Durante la explicación se aportan especificaciones de los datos y la arquitectura del modelo, pero también justificaciones que ayudan a visualizar como este entorno puede ser generalizado para usar otros datos o arquitecturas más complejas para solucionar el mismo problema, pero en distintas aplicaciones.
Todo es introducido en el orden en el que ha sido desarrollado. Mantiene un sentido y orden lógico en relación con como los datos fluyen desde el entrenamiento del modelo hasta la obtención del resultado final.

2.1. Datos

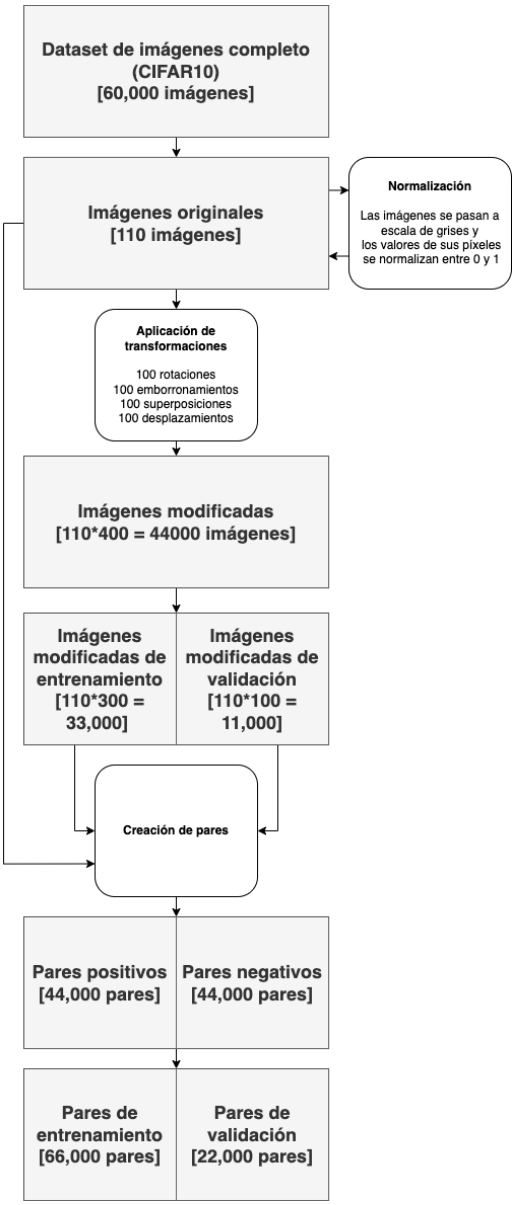


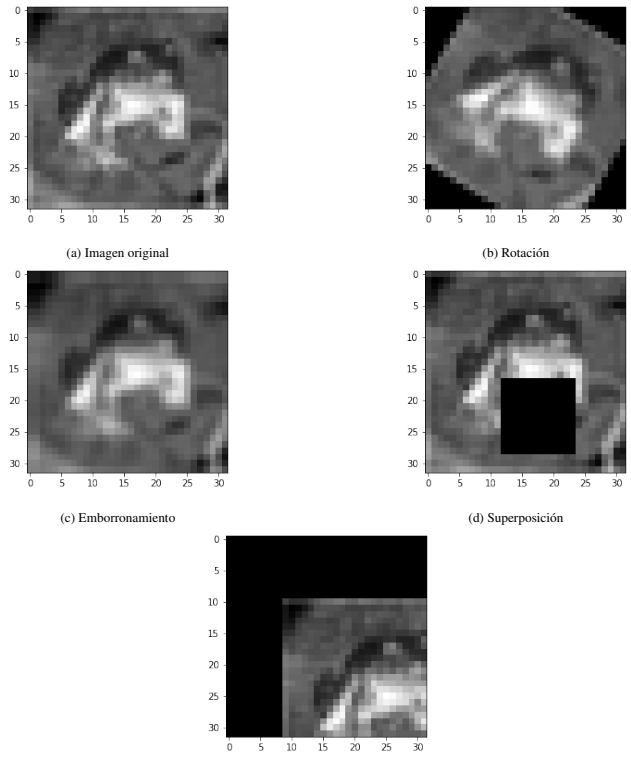
Fig. 1: Generación de datos de entrada a la red

Para hacer funcionar el sistema se requiere de un conjunto de imágenes, las cuales se quieren detectar aún habiendo sido modificadas. Para cada una de estas imágenes pertenecientes al conjunto se van a crear versiones alteradas que

posteriormente se van a organizar y etiquetar en pares para entrenar el modelo que aprenderá a detectar las posibles modificaciones de estas imágenes. (ver figura 1).
En la implementación concreta de este proyecto se ha usado la base de datos CIFAR10 [1]. Que se compone de 60,000 imágenes en color de 32x32 píxeles de tamaño y 10 clases distintas. Sin embargo, en este tipo de sistema, el número de clases es indiferente. Del total de imágenes únicamente se utilizan 110, ya que el *hardware* del que se dispone no es demasiado potente. Como se puede ver en la figura 1 un pequeño número de imágenes de entrada genera luego un número considerable de imágenes dependiendo de cuantas modificaciones realicemos. Evidentemente, esto puede ser generalizado a cualquier número de imágenes (y tamaño de estas si se adaptan al modelo) y modificaciones.

2.1.1. Generación de alteraciones

Para poder entrenar el sistema para que sea capaz de detectar las imágenes incluidas en el conjunto, aunque se les haya practicado alguna alteración, debemos generar alteraciones para cada una de ellas.
Para ello, se generan un número determinado de rotaciones, traslaciones en ambos ejes, aplicaciones de ruido gausiano, etc. Estas modificaciones se mantienen en una estructura de datos que posteriormente va a permitir recuperar todas las modificaciones generadas para cada imagen original.
La tabla 1 recoge las especificaciones de las transformaciones que se han aplicado a cada una de las imágenes originales para generar las imágenes modificadas en este proyecto en concreto.



(a) Imagen original

(b) Rotación

(c) Emborronamiento

(d) Superposición

(e) Traslación

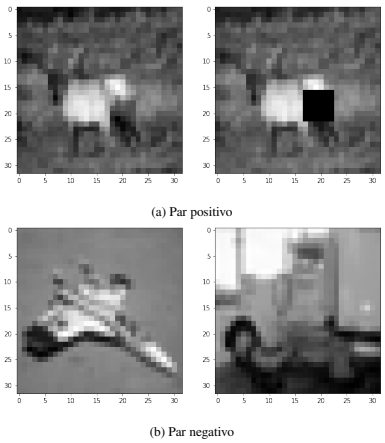
Fig. 2: Ejemplo de transformaciones

Se pueden visualizar algunos ejemplos de las transformaciones aplicadas a una imagen original en la figura 2.

Al generar 100 imágenes modificadas con cada una de las 4 transformaciones se obtienen 400 imágenes modificadas para cada una de las imágenes originales. Como se puede ver en la figura 1 esto resulta en un total de 44,000 imágenes que son alteraciones del conjunto inicial.
Sin embargo, se podrían generar modificaciones de otro tipo y en distintas cantidades si la aplicación final del modelo así lo requiere. De hecho, puede resultar interesante realizar un estudio concreto de la aplicación en la que se quiere aplicar este proyecto y realizar modificaciones que se adapten a como los usuarios de la aplicación normalmente realizan estas transformaciones con intenciones maliciosas para que el modelo sea mucho más robusto al caso concreto. (ver sección 5).

2.1.2. Generación de pares

Para entrenar el modelo va a ser necesario generar pares de imágenes etiquetados para luego usar cada una de las imágenes pertenecientes a los pares como entrada del modelo (ver 2.2.2).
Los pares etiquetados como positivos son todos aquellos en los que aparece una de las imágenes originales pertenecientes al conjunto de entrada junto a alguna modificación generada a partir de ella misma. Los pares negativos son aquellos que contienen dos imágenes originales del cojunto (no existe el par en el que ambas imágenes originales son la misma). Se repite este proceso para todas las imágenes del conjunto. Algunos de estos pares serán usados para entrenar la red y el resto como validación. El objetivo es comprobar que la red está generalizando y es capaz de detectar modificaciones de las imágenes originales que no ha visto durante el entrenamiento.
En este proyecto se han generado 44,000 pares positivos (ya que se han generado un total de 44,000 imágenes modificadas) y para mantener los datos equilibrados se han generado otros 44,000 pares negativos. En concreto se han destinado un total de 66,000 pares para el entrenamiento de la red y 22,000 para la validación en cada una de las iteraciones del entrenamiento (ver figura 1.
Se puede visualizar un ejemplo de los pares positivos y negativos que se generan en la figura 3. Los pares positivos son etiquetados con un 0 y los pares negativos son etiquetados con un 1 ya que es la distancia que queremos que el sistema tienda a predecir para estas combinaciones de



(a) Par positivo

(b) Par negativo

Fig. 3: Ejemplo de pares

Transformacion	Rango	Número	Descripción
Rotación	[-90, 90]	100	La imagen se rota un número determinado de grados.
Emborronamiento	[1, 10]	100	La imagen se emborrona usando ruido gausiano de un determinado tamaño en píxeles.
Superposición	[5, 15]	100	Se aplica un cuadrado negro de un cierto tamaño en píxeles en una posición aleatoria de la imágen.
Traslación	[-10, 10]	100	Se traslada la imagen en el eje X/Y o ambos un determinado núemor de píxeles.

TABLA 1: TRANSFORMACIONES PARA LA GENERACIÓN DE IMÁGENES MODIFICADAS

imágenes de entrada (ver 2.2.3).

2.2. Modelo

Una vez introducida la manera en que se generan y manipulan los datos que se usan para entrenar el modelo y hacer funcionar el sistema, se procede a introducir su estructura así como la manera en la que este es entrenado y evaluado. Por lo tanto, en esta sección se procederá a detallar como se estructura el modelo esencial del sistema basado en redes neuronales y el resto de partes del sistema, así como los cálculos de distancias entre vectores y la función de pérdida necesaria para poder utilizar el algoritmo de *backpropagation* [4].
Se detallarán también las características específicas implementadas en este proyecto, aunque de nuevo, no son necesarias para hacer funcionar un sistema como el que se introduce, ya que todo es generalizable como se presentará a continuación.

2.2.1. Extracción de características - Redes Neuronales Convolucionales

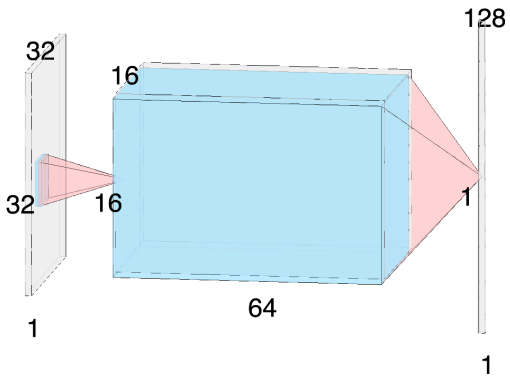


Fig. 4: Visualizacion extraccion de caracteristicas

La primera fase del sistema, es la parte más importante, ya que es la encargada de realizar la extracción de características de las imágenes de entrada para que luego se puedan ser utilizadas por las subsecuentes etapas.
Dentro del ámbito del aprendizaje profundo se encuentran un tipo de modelos de redes neuronales que son especialmente eficientes y que se especializan en la tarea de extraer características de imágenes mediante entrenamiento[5]. Evidentemente, se podría intentar usar otro tipo de algoritmos para extraer características, en esencia cualquier algoritmo que sea capaz de extraer un vector de características de una imagen puede servir como sustituto a esta primera fase del sistema. Sin embargo, las redes neuronales convolucionales (CNNs a parir de ahora), como el resto de redes neuronales, son valiosas por su capacidad de aprender de los datos que se les proporcionan como entrada y su gran poder de generalización. Por eso, el uso de CNNs es el más indicado para el sistema propuesto.
Se pretende aprovechar el gran poder de generalización que tienen estos modelos para que sea capaz de producir vectores de características similares (cercanos en el espacio vectorial) para imágenes originales y modificaciones realizadas sobre estas, algo que sería muy complicado o directamente inviable de perseguir con otros algoritmos de extracción de características.
En este proyecto se ha usado una CNN que consiste en una primera capa de entrada que admite imágenes de un solo canal de tamaño 32x32 píxeles, seguida de una capa de 64 filtros de convolución *zero-padding*[6] de tamaño 2x2 píxeles con activación *relu*[7]. Seguidamente, se aplica un *average-pooling*[8] de tamaño 2x2 píxeles. Esto resulta en un vector de tres dimensiones de tamaño 16x16x64 sobre el que se aplica un *global-average-pooling* que es inyectado en una capa densa que nos proporciona un vector de 1x1x128 que que es interpretado como el vector de características generado para la imagen de entrada (ver figura 4). También se añade una capa de *dropout*[9] que tiene efecto durante el entrenamiento del modelo, se puede ver en detalle en la figura 6.
Es importante ver como se puede sustituir esta CNN tan sencilla por una mucho más compleja o incluso por modelos existentes ya probados como *MobileNet*[10], *AlexNet*[11], etc. Para resolver escenarios que requieran de mayor o menor complejidad y/o mayor o menor tiempo de cómputo sin afectar a ninguna otra fase del sistema (ver sección 5).

2.2.2. Redes Neuronales Siamesas

Aunque esta no es la siguiente fase por la que pasan los datos después de la extracción de características (de hecho, no es una ”fase”del sistema de por sí) es importante mostrar este concepto referente a la arquitectura del modelo (ver figura 5) para poder entender las siguientes secciones.
El concepto de Redes Neuronales Siamesas (SNNs a partir de ahora)[12] se refiere a la utilización de dos modelos de redes neuronales idénticos de forma paralela durante la fase de entrenamiento y al hecho de que estas dos comparten los pesos (y las actualizaciones de estos). Gracias a ello, es

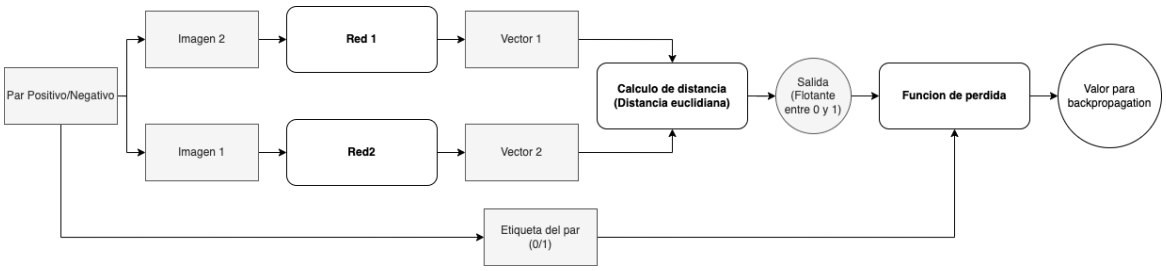


Fig. 5: Entrenamiento del modelo

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 32, 32, 1)]	0
conv2d (Conv2D)	(None, 32, 32, 64)	320
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
global_average_pooling2d (Gl	(None, 64)	0
dense (Dense)	(None, 128)	8320

Total params: 8,640
Trainable params: 8,640
Non-trainable params: 0

Fig. 6: Modelo de estracción de características en detalle

posible usar dos entradas distintas al mismo tiempo en dos modelos exactamente idénticos y posteriormente operar con ambas salidas.
En el caso del sistema propuesto, la fase de extracción de características durante el entrenamiento es una SNN en la que dos CNNs como la introducida en el apartado anterior son entrenadas conjuntamente y generan vectores de características para parejas de imágenes (ver 2.1.2) sobre los que posteriormente se calcula una métrica de distancia en el espacio vectorial.
La idea esencial que conduce al uso de SNNs es que utilizando dos redes exactamente iguales en todo momento, pero proporcionándoles dos imágenes distintas, se pueden obtener obtener dos vectores de características que seguidamente se pueden operar conjuntamente para realizar un cálculo de distancia en el espacio o diferencia entre ambos (ver 2.2.3). Si además, como es el caso, sabemos cuál debería ser la distancia esperada entre esos dos vectores (1 o 0 en este caso) es posible comparar esa distancia obtenida con la esperada (ver 2.2.4).

2.2.3. Función de distancia

Fig. 7: Visualización 2D de la función de distancia Euclidiana

Como se predentará posteriormente, para poder entrenar ambas redes se va a requerir de una función de pérdida que permita realizar *backpropagation* y actualizar los pesos de estas redes. Pero antes de ello, se debe generar un único valor escalar operando sobre los dos vectores de características que resultan después de que cada una de las CNNs que componen la SNN procesen las imágenes del par que se les proporciona.
No se debe perder de vista que, para el sistema, dos imágenes que se ”parecen”son en realidad dos imágenes que generan vectores de características, los cuales resultan en valores cercanos a 0 cuando son operados con la función de distancia que el sistema desee usar.
Cuando se opera con vectores multidimensionales se dispone de una gran cantidad de funciones que se pueden aplicar para determinar la distancia entre dos vectores. Por lo tanto, esta fase puede ser concebida como una función matemática que recibe como entrada dos vectores de igual dimensión (en el caos del sistema propuesto, dos vectores de características) y genera como salida un valor escalar que determina la distancia entre ambos.
En este proyecto se ha optado por utilizar la distancia euclidiana, aunque, de nuevo, cualquier otra función que pueda operar sobre dos vectores de igual dimensión para generar un entero que represente de alguna manera su similitud es válida. Pueden ser intercambiadas y estudiad su comportamiento para elegir la más adecuada a los datos para los que se desee emplear el sistema propuesto. En este proyecto se ha optado por la distancia euclidiana que sigue la fórmula de la ecuación 1 (se puede visualizar una versión en dos dimensiones en la figura 7).

$$\sqrt{max((vec_a - vec_b)^2, \epsilon)} \tag{1}$$

2.2.4. Función de pérdida

Cuando se trabaja con modelos de redes neuronales es esencial que al final del proceso siempre se disponga de una función de pérdida que genere un valor escalar con respecto a al valor esperado por los datos para que sea posible usar el algoritmo de *backpropagation* y proceder a optimizar los pesos de la red neuronal.
En el caso del sistema propuesto se empelean pares de dos imágenes que tienen asignada una etiqueta (ver 2.1.2). Por ello se va a utilizar la función de pérdida conocida como *Contrastive Loss*[13], esta es su ecuación:

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

Donde Y representa la etiqueta del par de imágenes (Es 1 para imágenes que deben estar cerca en el espacio y 0 para imágenes que deben estar lejos), D representa el resultado de aplicar la función de distancia a los vectores de características y $margin$ es un parámetro ajustable que indica el valor máximo de distancia que queremos que pueda haber

entre dos vectores.
Se puede observar como gracias a esta función de pérdida se deberían obtener valores altos a más juntas estén dos imágenes que deberían estar lejos en el espacio así como si se obtienen valores de distancia altos para dos imágenes que deberían estar cerca en el espacio. El efecto que esto produce durante el entrenamiento es que la red tienda a empujar juntas los pares de imágenes etiquetados como positivos y a empujar lejos de sí los pares de imágenes etiquetados como negativos. De esta manera, si el entrenamiento se realiza adecuadamente, se debería obtener un modelo que genera vectores de características cercanos en el espacio (notablemente parecidos) para imágenes etiquetadas como positivas entre sí y vectores de características lejanos en el espacio (notablemente distintos) para imágenes marcadas como negativas entre sí.

2.3. Entrenamiento del modelo

El modelo se entrena como cualquier otro modelo basado en redes neuronales, mediante el algoritmo de *backpropagation*. Se proporcionan como entrada los pares de imágenes generados (ver 2.1.2) y sus respectivas etiquetas. La red generará vectores de características para ambas imágenes, se calculará la distancia entre ellas mediante la función de distancia (ver 2.2.3) y se generará un valor para iniciar el algoritmo de *backpropagation* mediante la función de pérdida (ver 2.2.4).
En el caso concreto de este estudio se han usado 70 *epochs*, como se puede ver en la figura 8, y un *batch size* de 64 pares con el optimizador ADAM para el entrenamiento de la red. De nuevo, y como aplica para la mayoría de los pasos introducidos, estas decisiones pueden y probablemente deban ser distintas y estudiada en el caso de emplear una red de extracción de características distinta y/o un *dataset* diferente.

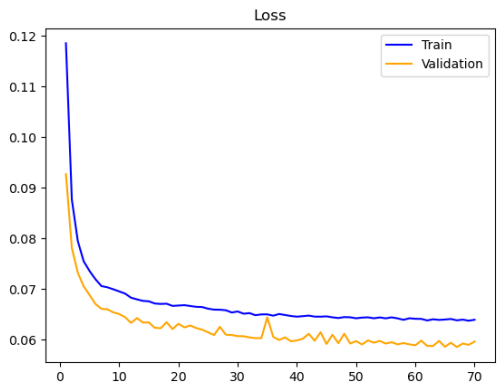


Fig. 8: Visualización del valor de *loss* durante el entrenamiento

2.3.1. Generación de *hashes* y cálculo de distancia entre ellos

El paso final por el que pasan nuestros datos es la generación de *hashes*. Con el modelo ya entrenado, se prescinde de una de las dos redes de la arquitectura siamesa y con únicamente una de ellas entregarle imágenes (tanto modificadas como no modificadas) para que esta genere un vector de características para dicha imagen. Para poder distribuir, almacenar y comparar estas imágenes de una forma más conveniente, segura y rápida debemos convertir los vectores de números flotantes producidos mediante alguna función de *hash*.
En este estudio se ha optado por una sencilla función que convierte los números en coma flotante del vector de características en binario, asignando 1 si el valor es mayor a 0 y asignando 0 si el valor es menor o igual a 0. De esta manera se reduce el espacio que ocupa en memoria y se hace más veloz la comparación de distancias con otro *hash*.
En el estudio realizado se usa la distancia de Hamming [14], una función que básicamente produce un entero que representa el número de valores distintos para cada una de las posiciones de dos vectores. (Ver figura 9 para visualizar el proceso completo).
Como en el caso del resto de apartados, en esta fase se puede optar por cualquier función de *hash* que se adecue al caso de uso y de la misma manera a cualquier función de distancia entre *hashes* que se pueda emplear entre los *hashes* generados.

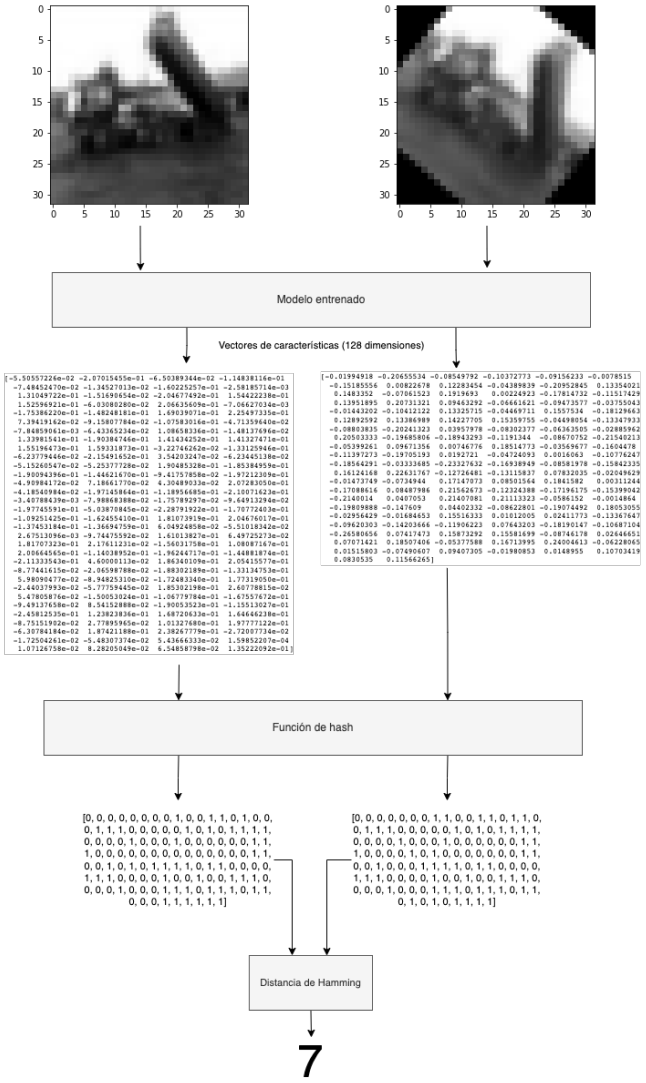


Fig. 9: Ejemplo de cálculo de distancia entre *hashes*

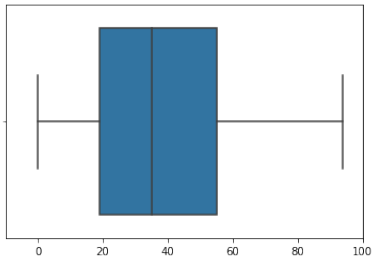
3 Resultados

Para comprobar los resultados producidos por el modelo introducido en el estudio se ha decidido observar las distri-

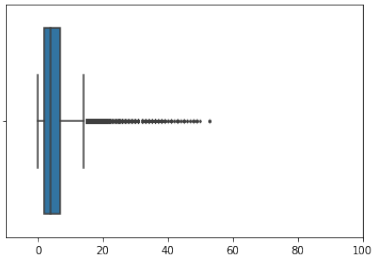
buciones de las distancias en dos ámbitos distintos:

- Distancia entre cada una de las imágenes originales y todas las demás imágenes originales.
- Distancia entre cada imagen original y todas sus versiones modificadas de validación.

En el primer caso, se compara una a una todas las imágenes originales con el resto de imágenes originales del *dataset*. Debemos esperar que la distancia entre estas sea grande en comparación al segundo caso para poder dictaminar que el sistema no las confunde y es capaz de diferencias correctamente entre las imágenes originales de la base de datos sobre la que queremos que actúe. Si se observa la distribución (ver figura 10) se observa que la media se sitúa cercana a las 40 unidades de distancia.
En el segundo caso, en el que se comparan las imágenes originales con cada una de sus imágenes modificadas de test (es decir, imágenes que la red neuronal que actúa como base del sistema nunca ha visto) la distribución sitúa la media cercana a las 5 unidades de distancia (se pueden observar también ciertos *outlayers*).
Esto nos indica que efectivamente el sistema está produciendo *hashes* mucho más cercanos para las imágenes que son modificaciones de sus originarias y, por lo tanto, tenemos la capacidad de marcar un *threshold* para poder dictaminar si una imagen de entrada (sin necesidad de acceder a la imagen como tal, únicamente al *hash* que genera) es una modificación de una de las imágenes pertenecientes o si no tiene relación con ninguna de ellas con un cierto grado de seguridad (este grado de seguridad dependerá de donde marquemos el *threshold* en la aplicación que use el sistema).



(a) Distancias de imagenes originales a imagenes originales



(b) Distancias de imagenes originales a sus modificadas

Fig. 10: Quartiles en las distribuciones de distancias (25 %, 50 %, 75 %)

Si se visualiza la figura 11 es posbible ver como para el sistema y datos utilizados en este estudio sería conveniente usar un *threshold* igual a 20, dictaminando así que si el *hash* de una imagen de entrada tiene una distancia inferior a 20 con alguno de los *hashes* generados para las imágenes a vetar se podría tratar de una modificación de esta.

Estos resultados variarán para todas las posibles combinaciones que el sistema propuesto puede acoger, así como para los datos que se empleen para su entrenamiento, pero siempre se debe apuntar a distribuciones parecidas a las obtenidas en este estudio para que sea posible su uso.

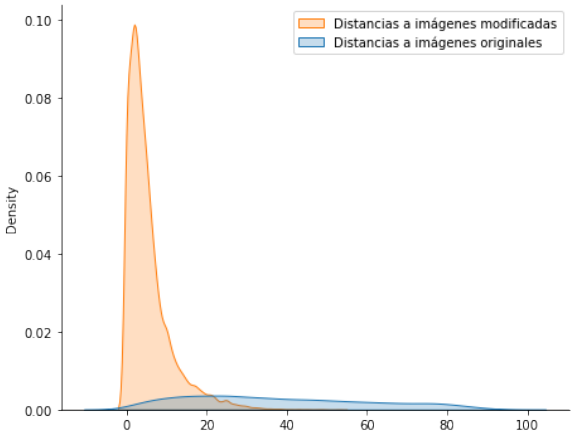


Fig. 11: Superposición de las distribuciones *hashes*

4 Conclusiones

Se puede concluir una vez observados los resultados que el sistema propuesto ha alcanzado los objetivos que se habían planteado en el comienzo del estudio. Por lo tanto, la prueba de concepto para comprobar si el sistema es capaz de detectar imágenes modificadas y permitir que dicho objetivo se cumpla sin la necesidad de tomar posesión sobre estas realizada en este estudio puede considerarse exitosa.
Además, debe destacarse la flexibilidad que esta arquitectura permite y que puede ser extendida a modelos de redes neuronales y funciones de *hash* mucho más complejas y que pueden llegar a generar resultados muy prometedores, así como la posibilidad de usar datos de entrenamiento más complejos y abundantes siguiendo exactamente las fases propuestas.
Es importante también concluir que la principal punto que pretende demostrar este estudio es que un sistema como el sugerido podría emplearse exitosamente en un entorno real, y no se pretende sugerir el modelo empleado concretamente en el estudio para ello. Sin embargo, se brindan todas las fases y posibles modificaciones que pueden ser aplicadas para que pueda efectuarse.
También puede concluirse que el uso de datos y/o modelos de extracción de características más complejos podrían reducir el solapamiento que puede observarse en los resultados (ver figura 11) así como la cantidad de *outlayers*, ya que gran parte de ellos pueden ser debidos a la similitud de las imágenes utilizadas debida a su reducido tamaño.
Para finalizar concluir que la utilidad que el sistema presentado puede llegar a tener si se entrena usando una red de extracción de características de última generación, una abundante cantidad de datos y específicamente datos que sean moramente susceptibles de ser vetados en determinados sistemas puede ser de gran utilidad para la sociedad.

5 TRABAJO FUTURO

El enfoque que se quiere dar a futuros trabajaos que amplíen el estudio presentado en este artículo es hacia la implementación del sistema en una aplicación real en producción.
En este artículo se ha concluido que el sistema propuesto puede cumplir con los objetivos y para ello se ha optado por versiones simples tanto del modelo de extracción de características como de la función de *hash* (y la función que calcula la distancia entre *hashes*) así como por un conjunto de imágenes de tamaño reducido y sin una utilidad real más allá del estudio de la viabilidad del sistema.
Por lo tanto, es conveniente ver que puntos deben considerarse en una aplicación real y como debería abordarse para posibles trabajos futuros.

5.1. Aplicación de diferentes modificaciones para la generación de pares

Esta es probablemente la fase más esencial al buscar que el sistema sea realmente efectivo en un caso de implementación. Es muy importante que las modificaciones que se generen sobre el conjunto de imágenes original sean significativas y tengan una relación directa con lo que se puede esperar que los usuarios que intentar evitar el veto vayan a realizar. Es decir, las modificaciones que se efectúen sobre las imágenes originales para generar los pares positivos y entrenar el modelo deberían ser decididas mediante un estudio de casos previos en los que los usuarios han enviado estas u otras imágenes con ligeras modificaciones. También, es relevante decidir hasta que punto debemos llegar con estas modificaciones, ya que si un usuario ha modificado en gran medida una imagen para saltarse el veto, pero esta imagen ha pasado a ser irreconocible o perder su ”valor”, no hay necesidad de entrenar el modelo para que detecte dicho caso.

5.2. Diferente model de extracción de características

El modelo de red neuronal convolucional utilizado en este estudio es básico y ciertamente suficiente para trabajar con los datos que se le presentan, pero en una aplicación real sería necesario y recomendable optar por modelos más complejos y probablemente pre entrenados.

5.3. Diferente función de hash

La función de *hash* utilizada en este estudio es simple. Aun así, reduce el tamaño de los vectores en memoria, ya que se pasa de guardar 128 números de coma flotante a guardar 128 binarios que pueden guardarse con un solo *bit* además de permitir que el cálculo de la distancia entre *hashes* sea mucho más rápido.
En una futura ampliación de este estudio podrían explorarse otras funciones de *hash* junto a sus respectivas funciones de distancia que hicieran una mayor optimización de espacio y/o de cálculo de distancias en el caso de que haya una gran cantidad de imágenes a vetar. Funciones *LSH*[15] pueden ayudar a hacer la búsqueda en conjuntos de *hashes* muy grandes mucho más eficiente.

REFERENCIAS

[1] Alex Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 8 de Abril del 2009.

[2] Keiron O'Shea, Ryan Nash, *An Introduction to Convolutional Neural Networks*, 26 de Noviembre del 2015.

[3] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, *Siamese Neural Networks for One-shot Image Recognition*, 10 de Julio del 2015.

[4] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, *Learning representations by back-propagating errors*, 31 de Julio del 1986.

[5] Manjunath Jogin et al., *Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning*, 18 de Mayo del 2018.

[6] Mahidhar Dwarampudi, N V Subba Reddy, *Effects of padding on LSTMs and CNNs*, 18 de Marzo del 2019.

[7] Abien Fred M. Agarap, *Deep Learning using Rectified Linear Units (ReLU)*, 2018.

[8] Lin et al., Global Network In Network, 16 de Diciembre del 2013.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 17 de Abril del 2017.

[11] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, 2021.

[12] Davide Chicco, *Siamese Neural Networks: An Overview*, 18 de Agosto del 2020.

[13] Khosla et al., *Supervised Contrastive Loss*, 23 de Abril del 2020.

[14] Abraham Bookstein, Vladimir A. Kulyukin, Timo Raita, *Generalized Hamming Distance*, Octubre del 2002.

[15] Aristides Gionis, Piotr Indyk, Rajeev Motwani, *Similarity Search in High Dimensions via Hashing*, 7 de Septiembre del 1999.