Universidad Zaragoza

Trabajo Fin de Máster

Binarización de descriptores en redes neuronales

Binarization of deep embeddings

Autor:

Guillermo García Otin

Director:

Javier Civera Sancho

ESCUELA DE INGENIERÍA Y ARQUITECTURA DE ZARAGOZA

2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. Guillermo Garcia Otin ,en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

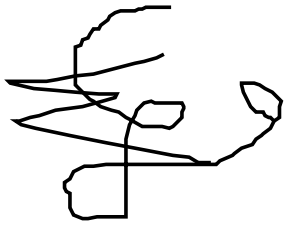
Declaro que el presente Trabajo de Fin de (Grado/Máster) Master

(Título del Trabajo)

Binarización de descriptores en redes neuronales

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 11 de septiembre, 2020



Fdo: Guillermo

AGRADECIMIENTOS

En primer lugar, me gustaría expresar mi más profundo agradecimiento a Javier, por darme la oportunidad de hacer este trabajo con él, a pesar de hacerlo desde Alemania, por su paciencia e incondicional ayuda en mis largos y tediosos mails y su profesionalidad y buena fe a la hora de hacer las cosas. Por otro lado, a mis padres y hermano que han servido de apoyo en todo momento, ayudando a que todo fuera por el buen camino. Por último, a mis amigos de la charanga Os Mozés, que me han enseñado que la música es la mejor forma de cargar las pilas y oxigenar las ideas.

RESUMEN

La binarización de descriptores calculados mediante redes neuronales profundas, el campo de estudio de este trabajo fin de máster, es un problema de investigación con aplicaciones relevantes a medio y largo plazo. Uno de los grandes retos de los modelos de aprendizaje automático actuales es reducir su elevado coste computacional, lo cual posibilitaría su uso a alta frecuencia y/o en plataformas limitadas (por ejemplo, drones pequeños o gafas de realidad virtual y aumentada).

La binarización de descriptores consiste en transformar un descriptor extraído por una red neuronal, compuesto por números reales (con representación en coma flotante), en uno compuestos por números binarios. La ventaja computacional viene aparejada a la reducción de tamaño del descriptor, para el que cada una de sus dimensiones pasa a ser representada por un solo bit (frente a las representaciones en coma flotante, entre las cuales las de doble precisión ocupan generalmente 64 bits). El efecto es una reducción en el espacio de memoria requerido para almacenar el descriptor, y una menor carga al realizar operaciones con dichos descriptores. Esta reducción de tamaño asume que en el descriptor de números reales hay redundancias, que serán usadas por el algoritmo de binarización.

En este trabajo fin de máster se propone, implementa y evalúa un algoritmo de binarización novedoso, que tiene en cuenta la variación de los datos mediante comparaciones entre las dimensiones del descriptor continuo. La implementación del método de binarización conllevó varios desafíos. Por un lado, disponer de datos adecuados para el entrenamiento de la red y para la comparación de los resultados con los del estado del arte, para lo que se han usado tres bases de datos con características diferentes. Por otro lado, la elección de la arquitectura de red, el proceso de entrenamiento y el diseño e la implementación de nuevas funciones de coste, para las cuáles se han evaluado varias alternativas.

Nuestros resultados experimentales demuestran que nuestra propuesta supera al estado del arte en binarización de descriptores en las bases de datos *MNIST* y *CIFAR-10*; e iguala y supera en algunos casos incluso a descriptores de números reales (con mucha menor huella computacional) en la base de datos *Partioned Nordland*.

Índice general

1. INTRODUCCIÓN **6**
1.1. Estructura del trabajo 7

2. REDES NEURONALES **9**
2.1. Neurona Artificial 9
2.2. Estructura y conexiones 10
2.3. Entrenamiento 13
2.4. Redes Siamesas 14
2.5. Redes Triplets 15

3. PROPUESTA DE MÉTODO DE BINARIZACIÓN **18**
3.1. Descriptor real 18
3.2. Principal Component Analisis (PCA) 19
3.3. Binarización por comparación 21
3.4. ¿Por qué funciona? 23

4. BASES DE DATOS UTILIZADAS **25**
4.1. MNIST handwritten digits 25
4.2. CIFAR-10 26
4.3. Partitioned Nordland Dataset 27

5. EVALUACIÓN Y MÉTRICAS **29**
5.1. Information Retrieval 29
5.2. Reconocimiento de lugares 32

6. DISEÑO DEL MÉTODO DE BINARIZACIÓN **33**
6.1. PCA antes de la binarización por comparación 33
6.2. Dimensión después de PCA y método de binarización por comparación . . . 34

7. RESULTADOS **36**
7.1. MNIST handwritten digits 36
7.1.1. Arquitectura de red y entrenamiento 36
7.1.2. Resultados 37
7.2. CIFAR-10 38
7.2.1. Arquitectura de red y entrenamiento 38
7.2.2. Resultados 39
7.3. Partitioned Nordland Dataset 40
7.3.1. Arquitectura de red y entrenamiento 40
7.3.2. Resultados 41

8. CONCLUSIÓN **47**

9. HERRAMIENTAS **48**

Capítulo 1

INTRODUCCIÓN

Este trabajo se enmarca dentro del campo de la visión por computador, disciplina que estudia el análisis, procesado e interpretación de las imágenes mediante sistemas computacionales. La visión por computador abarca, entre otros, aplicaciones como el reconocimiento y seguimiento de personas,o la localización tridimensional y construcción de mapas simultáneos (SLAM). En este trabajo vamos a centrarnos en el estudio de las redes neuronales profundas para aplicaciones de reconocimiento visual. La figura 1.1 ilustra dicho problema con tres imágenes extraídas de las bases de datos usadas en este trabajo. A la izquierda se muestra una imagen de la base de datos *MNIST*, cuyo objetivo es reconocer qué digito manuscrito se ve en la imagen (en este caso, el 3). En el centro se muestra una imagen de la base de datos *CIFAR-10*, cuyo objetivo es reconocer la categoría a la que pertenece dicha imagen (en este caso, un caballo). Finalmente, a la derecha, se observa una imagen de la base de datos *Nordland*, cuyo objetivo es reconocer lugares ante cambios grandes de apariencia.

Figura 1.1: Ejemplo de reconocimiento de imágenes

Las redes neuronales tienen la capacidad de reconocer patrones en imágenes similares a aquellas con las que han sido entrenadas. De manera general, en problemas de reconocimiento visual, se realiza en primer lugar el proceso conocido como entrenamiento, usando grandes bases de datos, a partir de las cuales la red aprende los patrones más representativos para

clasificar las imágenes en distintas categorías. En nuestra aplicación concreta, enmarcada dentro del problema conocido como *image retrieval*, la red comprime la información de los datos de entrada en descriptores (que son vectores de números reales), que reducen la dimensión de los datos y tienen cierta invarianza ante transformaciones dentro de la misma categoría.

La comparación de descriptores se realiza mediante su distancia (generalmente la distancia euclídea, aunque también se han usado otras en la bibliografía), perteneciendo a la misma categoría aquellos cuya distancia sea pequeña. En redes neuronales uno de los retos a resolver es reducir el coste computacional en las comparaciones entre estos descriptores, y el espacio en memoria que ocupan. Con este objetivo, en este trabajo se propone un algoritmo de binarización de descriptores, que transforma su representación de vectores de elementos tipo *float* o *double* a vectores binarios cuyos elementos son *bits*. La comparación de descriptores binarios se realiza mediante la distancia de Hamming, que es el número bits en el que ambos vectores difieren. Al trabajar con distancias de Hamming en lugar de con distancias euclídeas el coste computacional por emparejamiento puede verse reducido del orden de 35 veces [2].

La desventaja radica en que pasamos de un espacio continuo en el que cada dimensión tiene 32 o 64 bits de información (según si es *float* o *double*) a uno discreto donde cada dimensión tiene 1 bit (0 o 1), por lo tanto la capacidad representativa de cada descriptor se ve mermada. El objetivo dentro de esta línea de investigación es plantear algoritmos de binarización que compriman la información con la menor pérdida posible.

El objetivo de este trabajo es plantear un nuevo método de binarización flexible, fácil de implementar y manteniendo la precisión del descriptor continuo en bases de datos de complejidad baja, media y alta, en términos de dificultad de reconocimiento de patrones.

1.1. Estructura del trabajo

En este capítulo se han introducido el problema y los objetivos. En el capítulo 2 se introducen los conceptos de redes neuronales necesarios para comprender el resto del trabajo. En el capítulo 3 se explica en detalle el funcionamiento del método de binarización. En el capítulo 4 se presentan las bases de datos utilizadas. En el capítulo 5 se explican las métricas y como se lleva a cabo la evaluación de los resultados en cada base de datos.En el capítulo 6 se muestran los experimentos realizados para el diseño del método de binarización .En el capítulo 7 se presentan los resultados del trabajo y se comparan con los mejores métodos de binarización del mundo. En el capítulo 8 se ofrece la conclusión del trabajo y las líneas

de futuro . Finalmente, en el capítulo 9 se aporta información acerca de las herramientas utilizadas.

Capítulo 2

REDES NEURONALES

En este apartado se explican los conceptos básicos necesarios para la comprensión de nuestra propuesta de binarización y su evaluación.

Las redes neuronales forman parte del aprendizaje automático (*machine learning*); la rama de las ciencias de la computación que trabaja con algoritmos capaces de aprender a realizar ciertas tareas a partir únicamente de datos. Las redes neuronales aparecieron en la década de 1940 [9], pero han resurgido en la última década gracias al desarrollo tecnológico, especialmente al incremento de la potencia y memoria de los ordenadores, desarrollo de las tarjetas gráficas (GPUs) y a la creciente disponibilidad de grandes conjuntos de datos.

Las redes neuronales, así como otros métodos de *machine learning*, aprenden patrones en los datos de entrada, para seguidamente, hacer predicciones en datos nunca antes vistos. El potencial de las redes neuronales en concreto reside en que son capaces de aprender funciones más complejas que ningún otro método.

2.1. Neurona Artificial

La unidad básica de las redes neuronales son las neuronas artificiales. Cada neurona es un elemento capaz de procesar individualmente un vector n-dimensional de entrada, $x = (x_1, x_2, ..., x_n)^T$, para producir una salida y. La salida de la neurona viene determinada por el producto escalar de los datos de entrada por un vector de pesos $w = (w_1, w_2, ..., w_n)^T$ más un sesgo (o bias) b. Por último, se aplica una función de activación no lineal f para poder modelar relaciones no lineales entre los datos y la salidad de la red. La formulación completa es la siguiente:

$$y = h_{w,b}(x) = f\left(b + \sum_{n=1}^{N} w_i x_i\right) \tag{2.1}$$

El valor de los pesos como el sesgo se determinan mediante el aprendizaje. Se puede ver un ejemplo de la representación típica de una neurona artificial en la Figura 2.1. En la imagen se pueden apreciar las entradas, la salida y la función matemática completa que implementa.

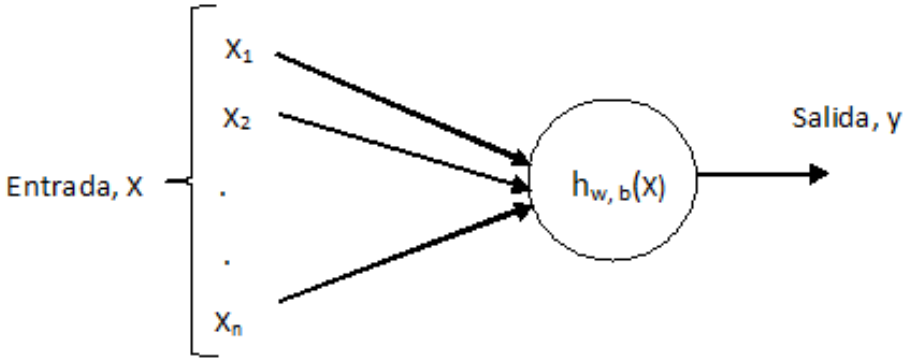


Figura 2.1: Neurona artificial.

2.2. Estructura y conexiones

En la estructura más básica de una red neuronal (*feedforward neural network*), un conjunto de neuronas artificiales en paralelo forma una capa de la red neuronal, y una red neuronal se forma mediante la concatenación en serie de diferentes capas.

Los datos de entrada se introducen a la red por la llamada *capa de entrada*, la cual consta de tantas neuronas como dimensiones tengan nuestros datos (en nuestro caso tantas neuronas como píxeles tenga la imagen de entrada). Seguidamente los datos pasan por una serie de capas intermedias, *capas ocultas* (*hidden layers* en inglés), las cuales realizan las operaciones para la extracción de patrones relevantes en la imagen. Finalmente, la *capa de salida* nos aporta la información requerida, dependiendo del problema que se afronte (en este caso obtendremos el descriptor de cada imagen).

A continuación, se explica y se ilustra los distintos tipos de capas que se han utilizado en las redes neuronales del trabajo:

- **Capa totalmente conectada (*fully conected* en inglés):** En estas capas, cada neurona se conecta a todas las de la capa anterior. Esto permite que se puedan tener en

cuenta todas las posibles relaciones entre los datos de entrada en la salida. Su desventaja es que tienen un número elevado de parámetros y por lo tanto el entrenamiento es más costoso.

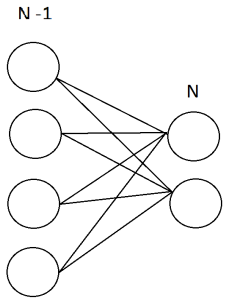


Figura 2.2: Ejemplo de capa totalmente conectada

- **Capa localmente conectada:** Es un tipo de capas en las que sus neuronas se conectan solo a un sub-conjunto de las de la capa anterior. La finalidad es distinguir patrones locales dentro de los datos de entrada con un coste de entrenamiento menor ya que el número de parámetros utilizado es menor que en las totalmente conectadas.

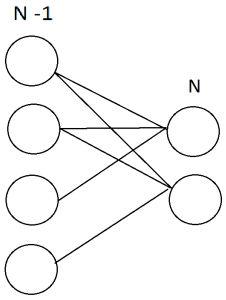


Figura 2.3: Ejemplo de capa localmente conectada

- **Capa convolucional:** Es una de las capas localmente conectadas mas usadas. Su aplicación en el tratamiento de imágenes es ya estándar, ya que su estructura hace que posea invarianza espacial, es decir, es capaz de reconocer determinadas características u objetos en cualquier lugar de la imagen. Esto lo logra mediante la operación de convolución que se ilustra a continuación (Figura 2.6). Los únicos parámetros que aprende esta capa son los pesos de un número de filtros que son desplazado por toda la imagen. Son considerablemente menos costosas computacionalmente que las capas totalmente conectadas, en las cuales cada píxel tendría un peso asociado.

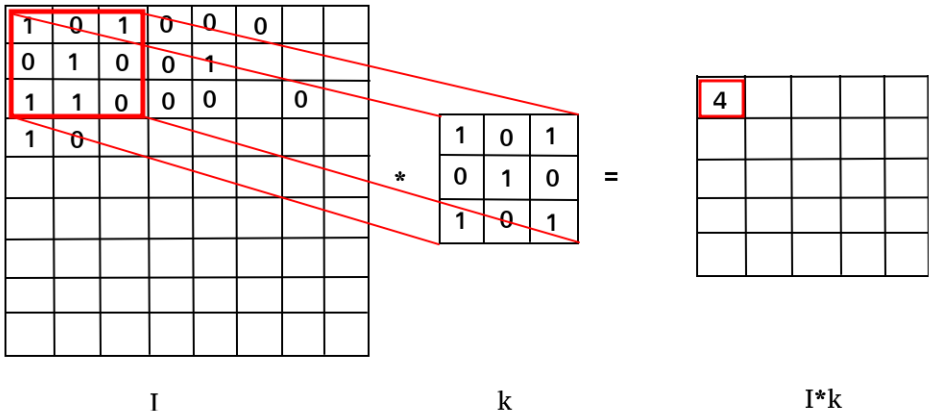


Figura 2.4: Ejemplo de funcionamiento de la operación de convolución llevada a cabo por la capa convolucional. El filtro es desplazado por toda la imagen multiplicando todos los píxeles del Input y resultando la nueva matriz ilustrada como I * K

- **Capa reductora (*pooling* en inglés):** es otro tipo de capa localmente conectada que aplica directamente una función conocida sobre la entrada y no tiene parámetros para aprender. Su aplicación mas frecuente y para lo que se usan en el trabajo consiste en su colocación justo después de las capas convolucionales para reducir la dimensión de estas.

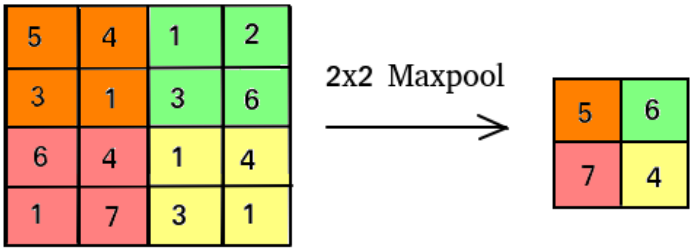


Figura 2.5: Ejemplo de capa reductora. Se ilustra la operación de *max pooling* la cual consiste en escoger el máximo de una ventana.

- **Capa *Flatten***: Se trata de una capa que transforma un tensor o matriz de entrada en un vector unidimensional. Se suele utilizar al inicio de las redes neuronales que trabajan con imágenes para transformar la entrada (por ejemplo, una imagen de 28×28) a un vector unidimensional (siguiendo con el ejemplo anterior, de 784 dimensiones) y trabajar con la red neuronal con las capas indicadas anteriormente.

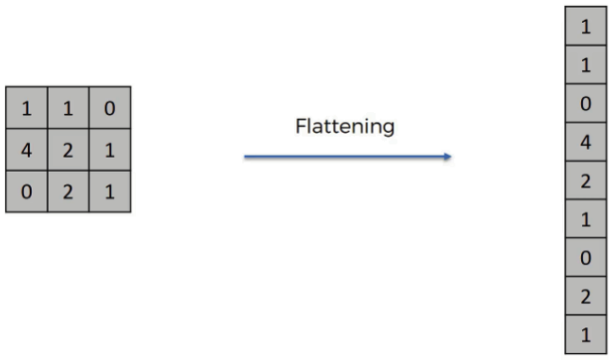


Figura 2.6: Ejemplo de funcionamiento de la capa *flatten*

2.3. Entrenamiento

El entrenamiento de una red neuronal es el proceso mediante el cual la red aprende a realizar la tarea para la cual ha sido diseñada. Consiste en iterar sobre los valores de los parámetros de la red (pesos y sesgos) de manera que el error de predicción en los datos de entrenamiento sea mínimo. Antes de explicar el proceso de entrenamiento en detalle, es necesario conocer una serie de términos:

Número de épocas: es un hiperparámetro que representa el número de veces que se recorre el conjunto de datos de entrenamiento completo.

Batch size: es un hiperparámetro que define el número de ejemplos del conjunto completo de datos que se van a usar en una actualización de los pesos de la red. Una de las principales ventajas de usar un batch size de tamaño pequeño es la menor necesidad de memoria durante el proceso de propagación hacia atrás.

Step size: es un hiperparámetro que define la magnitud de actualización de los parámetros entrenables durante el proceso de *back propagation*, explicado más adelante. La elección correcta de este parámetro es determinante para lograr la convergencia del algoritmo.

Función de coste: conocida en inglés como *loss*. Es la función que obtiene el valor del error entre la salida de la red y la salida que se esperaba. Es la función a optimizar durante el proceso de entrenamiento.

$$J(\theta) = \frac{1}{m}\sum_{n=1}^{m} Coste(h_\theta(x^{(i)}), y^{(i)}) \tag{2.2}$$

Habitualmente,las redes neuronales, se entrenan mediante el método *Stochastic Gradient Descent, (SGD)* [9] el cual consiste en aplicar el método de descenso de gradiente, *Gradient Descent,* a cada batch, con el objetivo de reducir la función de coste. El proceso para cada

batch es el siguiente:

Cada imagen del batch es propagada hacia adelante, de tal manera que, atraviesa la red siendo multiplicada por los parámetros entrenables (pesos y sesgos) de cada capa. El resultado predicho por la red es comparado con el resultado deseado y se obtiene la función de pérdida (véase Figura 2.7), la cual depende de todos los parámetros entrenables de la red. Para cada imagen en particular y con el objetivo de reducir la función de pérdida, se aplica el algoritmo de propagación hacia atrás (*back propagation*), en el cual se calcula la derivada de la función de coste con respecto a cada parámetro entrenable $\nabla(w_1, w_2...w_n)$. De aquí obtenemos un vector fila con tantas dimensiones como parámetros entrenables, el cual nos indica en qué dirección habría que variar cada parámetro para conseguir reducir la función de coste de esta imagen en concreto.

Ahora bien, no solo tenemos una imagen. En el entrenamiento, se suman las contribuciones de cada imagen en el batch y se aplica la actualización a los parámetros entrenables. El proceso se repite iterativamente para todos los batchs (SGD) y tantas veces como épocas haya, $w_{new} = w_{old} - \eta\nabla(w_1, w_2...w_n)$. Al final se habrá reducido la función de coste total, es decir, de todos los datos, y, por tanto, el error entre la predicción de la red y la realidad será más pequeño. De esta manera la red aprende a hacer predicciones cada vez más acertadas en los datos de entrenamiento. Asumiendo que los datos de test son similares a los datos de entrenamiento, las predicciones también tendrán una alta precisión en ellos.

2.4. Redes Siamesas

En este trabajo se utilizan redes siamesas, las cuales se componen de dos redes neuronales idénticas que comparten sus pesos. Al introducir dos imágenes distintas como entradas de las redes (una por cada red), los descriptores de las capas de salida deberían ser parecidos si pertenecen a la misma clase y diferentes si pertenecen a clases distintas. Un esquema ilustrativo de dichas redes se puede ver en la Figura 2.8:

El proceso de entrenamiento de las redes neuronales difiere del estándar. La única información disponible es si los pares son el mismo (pares positivos) o no (pares negativos). Por tanto, es necesario una función de coste acorde a estas características. En este trabajo se usa la función de coste *contrastive* [10], la cual acerca los descriptores cuando su etiqueta es 1, pares positivos, y los aleja cuando su etiqueta es 0, pares negativos.

$$L = yd^2 + (1 - y)max(m - d, 0)^2 \tag{2.3}$$

Donde L es el coste, d es la distancia euclídea entre los descriptores de la pareja, y es la

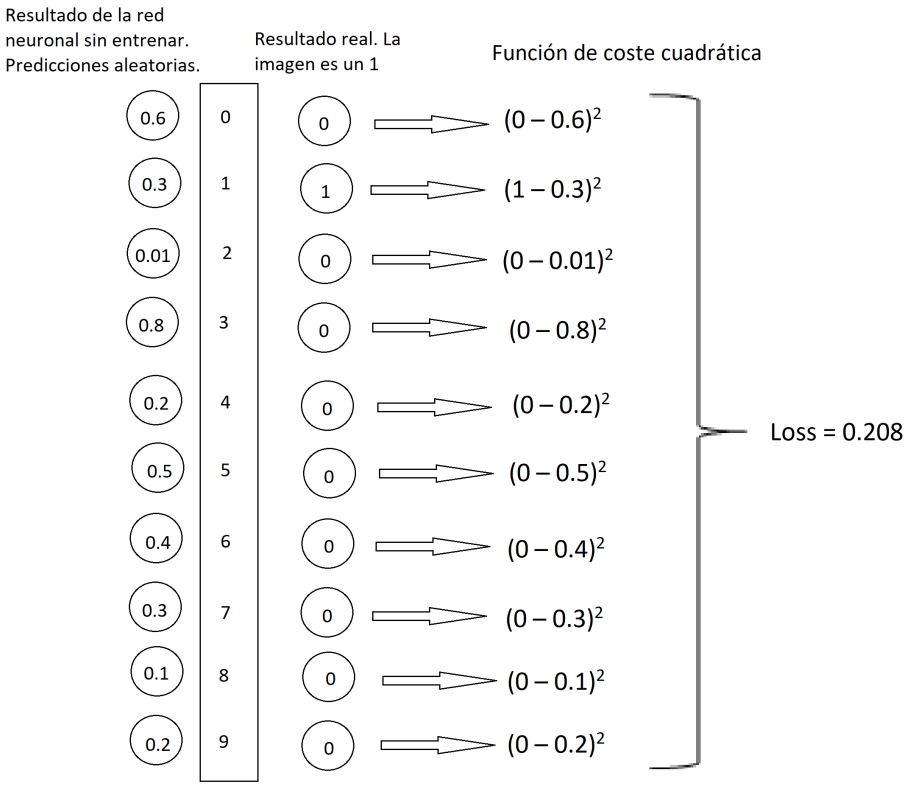


Figura 2.7: Ejemplo numérico de predicción de los dígitos del dataset MNIST. Tras atravesar la red neuronal sin entrenar se puede apreciar que las predicciones no son correctas (primera columnas de neuronas), la siguiente columna de neuronas ilustra lo que idealmente debería de resultar de la predicción. El error se calcula con una función de coste cuadrática.

etiqueta de la pareja y m es el margen.

Acercar los descriptores significa que cuando su distancia d es grande, pero es la misma pareja ($y = 1$) la función de coste es grande, lo cual indica que la distancia de la pareja es errónea. Por eso, en el proceso de entrenamiento, igual que en la red simple, la función de coste es minimizada y en este caso minimizar significa hacer la función de pérdida más pequeña, que a su vez implica que la distancia entre los descriptores sea pequeña, es decir, los acerca. De manera equivalente se alejan los descriptores gracias al max de la función si son una pareja negativa ($y = 0$) pero su distancia es menor que el margen.

2.5. Redes Triplets

Uno de los problemas de las redes siamesas es que, durante su entrenamiento, trabajan sucesivamente con un ejemplo positivo y después con uno negativo. Puede darse el caso

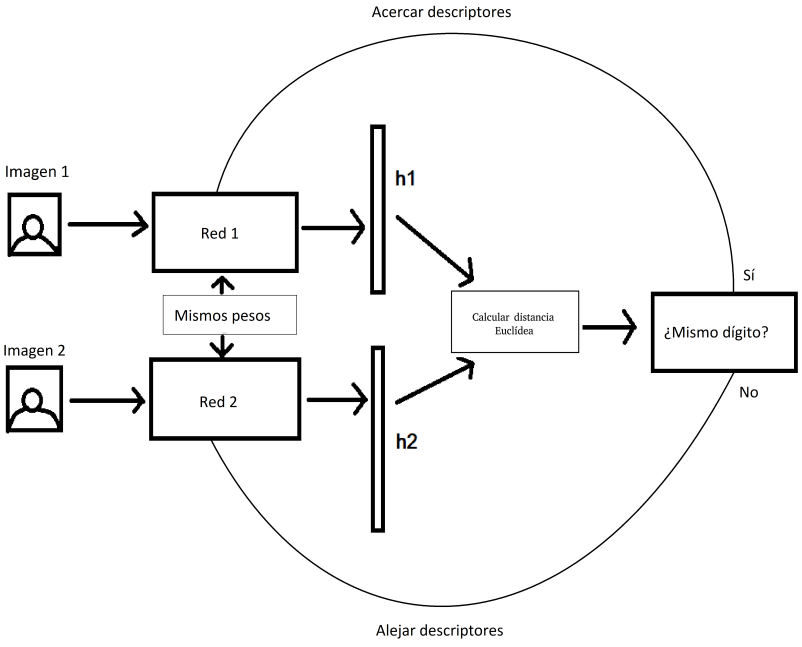


Figura 2.8: Ejemplo de red siamesa y su funcionamiento. h1 y h2 representan los descriptores de cada imagen al final de la red.

de que al tratar de acercar imágenes similares, los parámetros evolucionen de tal manera que parte de las negativas también se acerquen. También puede suceder que al alejar las negativas, se alejen parte de las positivas, dándose un efecto pulsante.

Las redes triplets trabajan a la vez con el ejemplo positivo y el negativo. Estas redes están formadas por tres copias de la misma red. Tienen tres entradas, dos ejemplos que se consideren similares junto a uno que se considere distinto. La red hace evolucionar sus parámetros para acercar unos y alejar otros a la vez. De esta forma, se favorece que la evolución de los parámetros no acerque ni aleje los ejemplos que no se desean. Se puede ver el funcionamiento en la Figura 2.9. Como las tres redes son idénticas, se sigue teniendo un único conjunto de parámetros.

Como función de error se usa la función de error triplet de Wohlhart-Lepetit

$$E = \text{máx}\left(0{,}1 - \frac{d_n}{margen + d_p}\right) \tag{2.4}$$

Donde E es el error, d_p es la distancia entre los vectores de la pareja positiva, d_n la distancia con el ejemplo negativo y el margen es el parámetro que permite limitar la diferencia entre las dos distancias. En esta función, el error es nulo cuando la distancia negativa supera

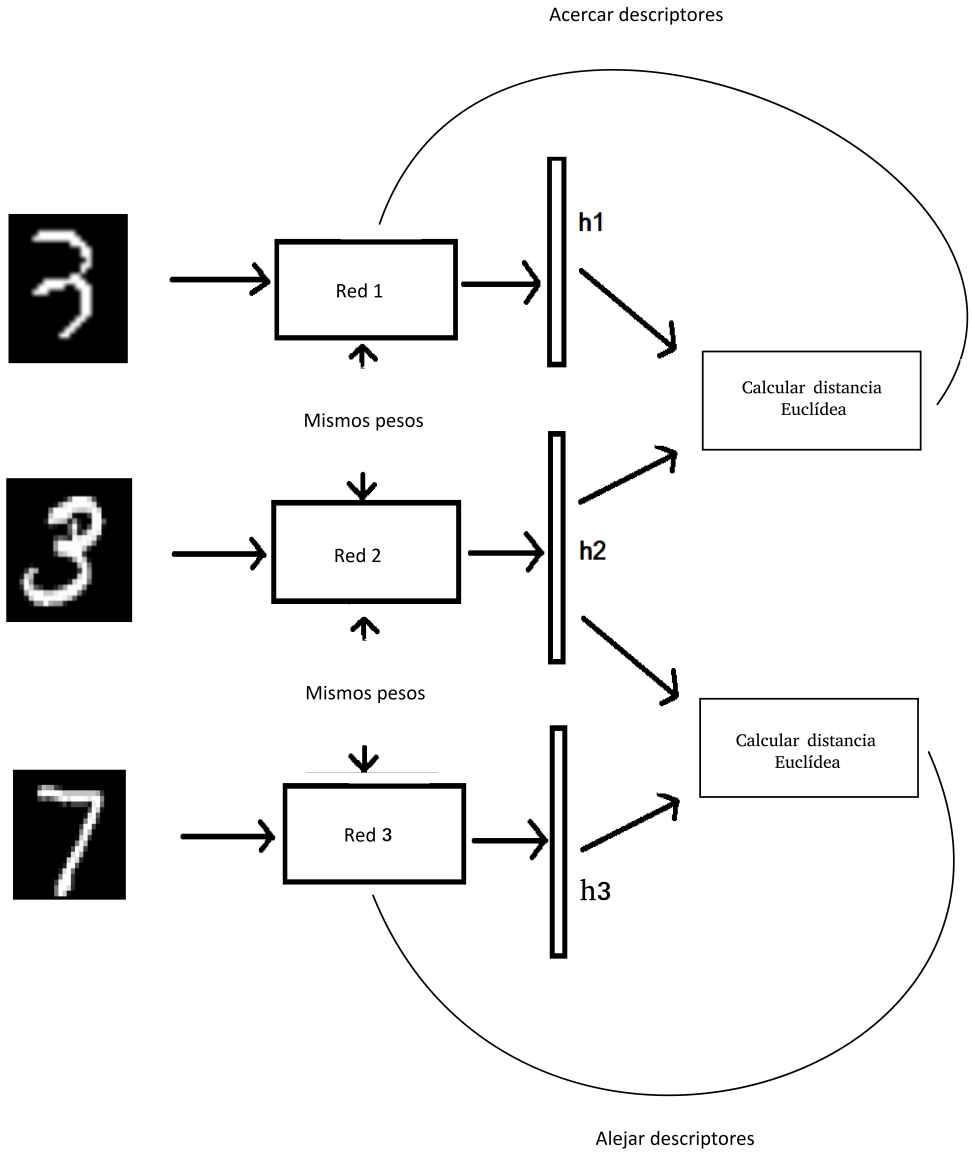


Figura 2.9: Ejemplo de red triplet y su funcionamiento. $h1$, $h2$ y $h3$ representan los descriptores de cada imagen al final de la red.

a la positiva por cierto margen. Además, el error está acotado entre 0 y 1, lo cual puede tener efectos positivos sobre la estabilidad en el entrenamiento.

Capítulo 3

PROPUESTA DE MÉTODO DE BINARIZACIÓN

El método de binarización que proponemos en este trabajo consta de tres etapas que se explican y detallan a continuación. Estas etapas se pueden visualizar en la imagen 3.1

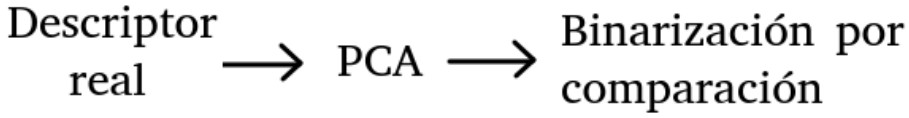


Figura 3.1: Método de binarización

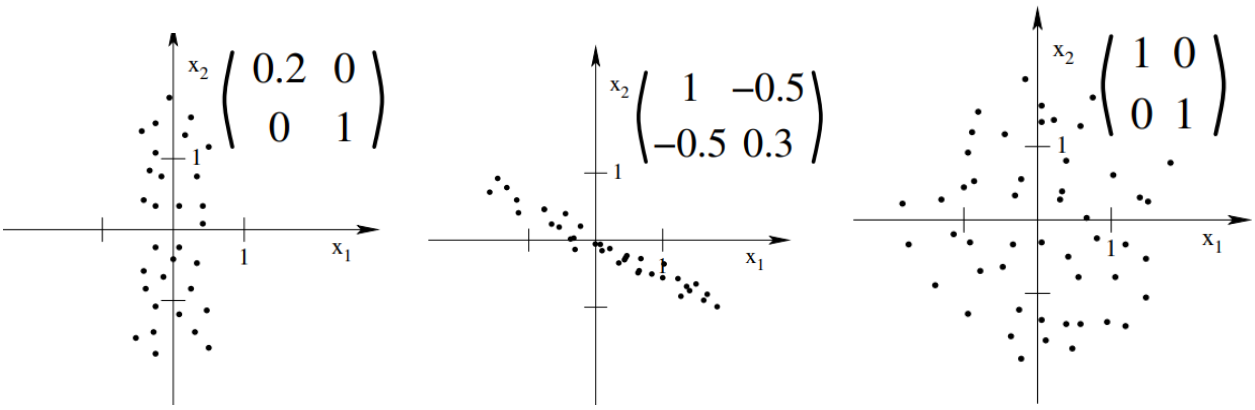
3.1. Descriptor real

En este trabajo se consideran tres formas de entrenamiento de la red neuronal: clasificación con una red neuronal individual, comparación entre imágenes con una red neuronal siamesa y comparación entre imagenes con una red neuronal triplet. Al final del entrenamiento, en los tres métodos, la red neuronal es capaz de aprender parámetros que propician descriptores acordes con la categoría de cada imagen. En el caso de la red siamesa y triplet, estos descriptores se extraen de la última capa de la red, y en el caso de una única red se extraen de la penúltima capa, porque la última es la capa de clasificación. Por lo tanto, el descriptor de cada imagen es un vector con tantas dimensiones como neuronas tengan estas capas, y, cada neurona tendrá como salida un número real que dependerá de los parámetros de la red y la imagen introducida.

3.2. Principal Component Analisis (PCA)

PCA es un procedimiento estadístico que utiliza una transformación ortogonal para convertir un conjunto de observaciones de variables posiblemente correlacionadas en otro conjunto sin correlación lineal compuesto por las llamadas componentes principales. Esta transformación se define de tal manera que el primer componente principal tiene la mayor varianza posible (es decir, representa la mayor variabilidad posible en los datos), y cada componente subsiguiente a su vez tiene la mayor varianza posible bajo la restricción que es ortogonal a los componentes anteriores. Los vectores resultantes (cada uno de los cuales es una combinación lineal de las variables y contiene n observaciones) son un conjunto de bases ortogonales no correladas.

El método usa la matriz de convarianza para estimar la varianza de los datos respecto a las dimensiones del subespacio vectorial en el que se encuentran. En la diagonal de la matriz se encuentra la covarianza de los datos en cada dimensión, y fuera de la diagonal se encuentra la correlación de los datos de la dimensión i-ésima con la j-ésima. Es decir, si un conjunto de datos se representa con dos variables, la matriz de covarianza será de 2x2, de la misma forma, tendremos una matriz N x N en dimensiones superiores, como es el ejemplo abordado en este trabajo. Con objeto de explicar gráfica y matemáticamente la matriz de covarianza y su importancia en el método PCA, se darán ejemplos en 2 dimensiones, ya que es imposible representar gráficamente las dimensiones superiores a 3. Sin embargo, la ampliación a dimensiones superiores consiste en la misma idea, pero en matrices mayores.



(a) (b) (c)

Figura 3.2: Distribuciones de datos y sus matrices de covarianza 2x2.

En la Figura 3.2 se puede apreciar que la matriz de covarianza es simétrica ya que es lo mismo la covarianza del elemento i-ésimo con el j-ésimo que viceversa. Además, se observa que la matriz de covarianza contiene la información sobre la dirección de la varianza máxima.

Por ejemplo, en la Figura 3.2a la varianza máxima se encuentra en el eje x_2 y se corrobora ya que el valor propio asociado al vector propio en esa dirección es cinco veces mayor (1 frente a 0.2). Por otra parte, en la Figura 3.2b, se puede observar una distribución de los datos oblicua en la que no se sabe a priori la dirección de máxima varianza. En este caso se aplica la técnica de diagonalización de la matriz para obtener la dirección de máxima varianza, la cual será la dirección del vector propio asociada al mayor valor propio. Esto se ilustra en la Figura 3.3. La proyección de los datos en las direcciones de los vectores propios se realiza de la siguiente manera:

Sea X el conjunto de todos los datos, que consideraremos $I-$dimensionales. Por conveniencia, los representamos en una matriz $I \times M$, con $M \leq I$:

$$X = (x_1, ..., x_M). \tag{3.1}$$

Su matriz de covarianza es la siguiente:

$$C_1 = XX^{\mathrm{T}}/M, \tag{3.2}$$

y su descomposición en valores propios se puede realizar de la siguiente manera:

$$C_1 U_1 = U_1 \Lambda_1 \Longrightarrow C_1 = U_1 \Lambda_1 {U^{\mathrm{T}}}_1 \tag{3.3}$$

Los datos contenidos en X, representados en el sistema de coordenadas de los vectores propios, son:

$$Y_1 = {U_1}^{\mathrm{T}} X \tag{3.4}$$

De esta manera se obtienen las direcciones de mayor varianza, y los datos pueden ser representados en direcciones que aportan información significativa sobre las variaciones más relevantes.

Principal component análisis (PCA) tiene varias aplicaciones. Sin embargo, la más relevante es reducir la dimensionalidad de un conjunto de datos cuyas dimensiones están muy correlacionadas entre sí, conservando la mayor cantidad posible de información. Una vez se han proyectado los datos en las direcciones de máxima varianza, la reducción de dimensiones propiciará la mínima perdida de información posible ya que la explicación de la varianza esta ordenada de manera creciente en las dimensiones en las que están representados los datos.

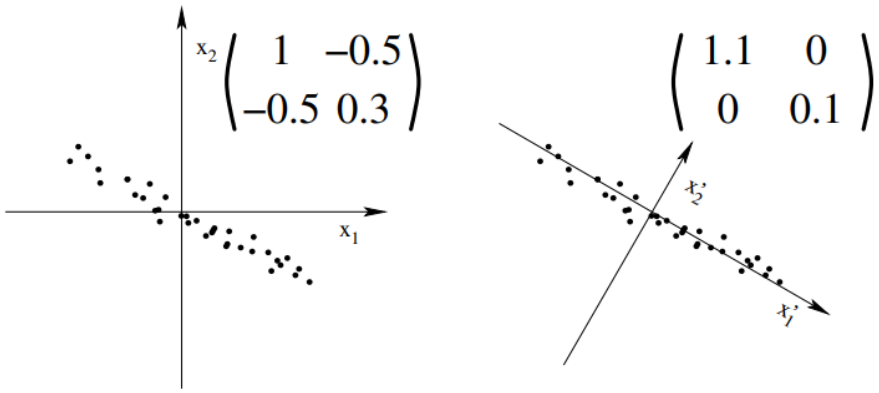


Figura 3.3: Diagonalización de la matriz de covarianza.

3.3. Binarización por comparación

La función de binarización o binarización por comparación, consiste en una función que selecciona pares aleatorios de las dimensiones del descriptor real de cada imagen para después compararlos entre sí, y de esta manera obtener un descriptor binario. Los pares no pueden ser repetidos ni tampoco ser el mismo par en orden diferente ya que se perdería una dimensión cada vez que esto ocurriera. La fórmula 3.5 detalla esta operación y en la figura 3.4 se ilustra para una dimensión del descriptor real de 4 y una dimensión del descriptor binario de 5. Nótese que la dimensión del descriptor binario puede ser igual al continuo, mayor o menor.

$$f(x,a,b) = \begin{cases} 0 & si \quad x(a) \leq x(b) \\ \\ 1 & si \quad x(a) > x(b) \end{cases} \tag{3.5}$$

En esta fórmula, x es el vector que se quiere binarizar y a y b son dos números enteros que representan las coordenadas del vector que se comparan.

Se ha comprobado que la binarización por comparación de pares aleatorios realizada directamente en los descriptor reales, sin realizar PCA antes, no ofrece buenos resultados. El motivo es que la varianza que explica los valores de cada dimensión esta repartida uniformemente entre todas las dimensiones, y por lo tanto comparar dimensiones individuales no es de gran utilidad, ya que, en este caso, el descriptor representa la imagen correctamente cuando se considera de manera completa. En la comparación de pares, la mayoría dan resultados equivocados, lo cual merma los resultados del método.

La varianza explicada por los valores de cada dimensión se refiere a la cantidad de cambio que cada dimensión del vector varía, al variar la imagen. Esto son valores concretos, por ejemplo de 1.8 a 2.5 en una dimensión. De esta manera, si tuviéramos solo una dimensión que

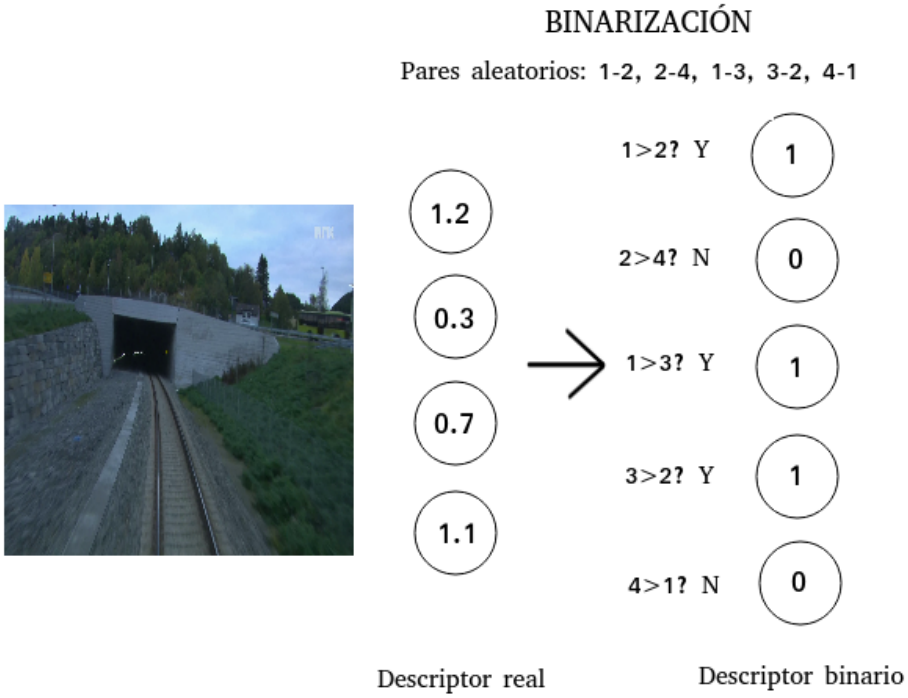


Figura 3.4: Ilustración sencilla de función de binarización

explicara el 100 % de la varianza bastaría con mirar a ese valor para clasificar correctamente la imagen, ya que, variaría exactamente de acuerdo a la categoría de cada imagen.

Mediante la aplicación del método PCA, explicado anteriormente, se consigue representar las imágenes con vectores en los que la descripción de varianza está ordenada, es decir, las primeras dimensiones del vector explican más varianza que las siguientes. Los nuevos vectores se consiguen después de una proyección lineal con los vectores propios de la matriz de covarianza de los vectores de las imágenes. Estos vectores propios al estar ordenados, es decir, al principio los que tienen valores propios mayores y por lo tanto explican más varianza de la imagen, resulta en que los nuevos vectores después de aplicar PCA tienen una explicación de la varianza ordenada de mayor a menor.

Aprovechando este ordenamiento, lo ideal sería que los pares se eligieran aleatoriamente pero dando mas probabilidad a pares hechos con las dimensiones del principio, ya que, según se ha argumentado anteriormente, estas variarán más representativamente dependiendo de la imagen y por lo tanto las comparaciones contendrán más información. Con la finalidad de lograr este efecto de más probabilidad a pares hechos con dimensiones al principio del descriptor, se propone la reducción del descriptor real con PCA a una dimensión pequeña k, proyectada por los k vectores propios que explican más varianza, para seguidamente tomar

pares aleatorios de estos vectores k dimensional y por lo tanto que todas las comparaciones sean con dimensiones que varían significativamente según la imagen. Se ha comprobado que esta técnica proporciona mejores resultados que coger pares aleatorios con probabilidad según una función de densidad exponencial con origen en el principio del descriptor.

3.4. ¿Por qué funciona?

La robustez del método radica intuitivamente en dos motivos:

Primero en que, las comparaciones aunque son aleatorias, son las mismas para todas las imágenes. Por eso, aunque el descriptor pierda capacidad de explicar la información de la imagen en primera instancia debido al paso de continuo a binario, la precisión se mantiene cuando se compara la distancia del descriptor de cada imagen con las demás. Es decir, se consiguen distancias incluso más grandes entre los descriptores que representan imágenes diferentes y distancias incluso más pequeñas para los descriptores que representan imágenes con la misma etiqueta. En la figura 3.5 se explica esto visualmente.

Segundo en que, como se ha explicado anteriormente, mediante PCA se consigue ordenar los vectores de acuerdo a la varianza de que cada dimensión representa, así, ante diferentes imágenes la variación de las dimensiones del descriptor real en las primeras dimensiones variara significativamente, y por tanto, las comparaciones para pasar el descriptor a binario contendrán información relevante de cada imagen.

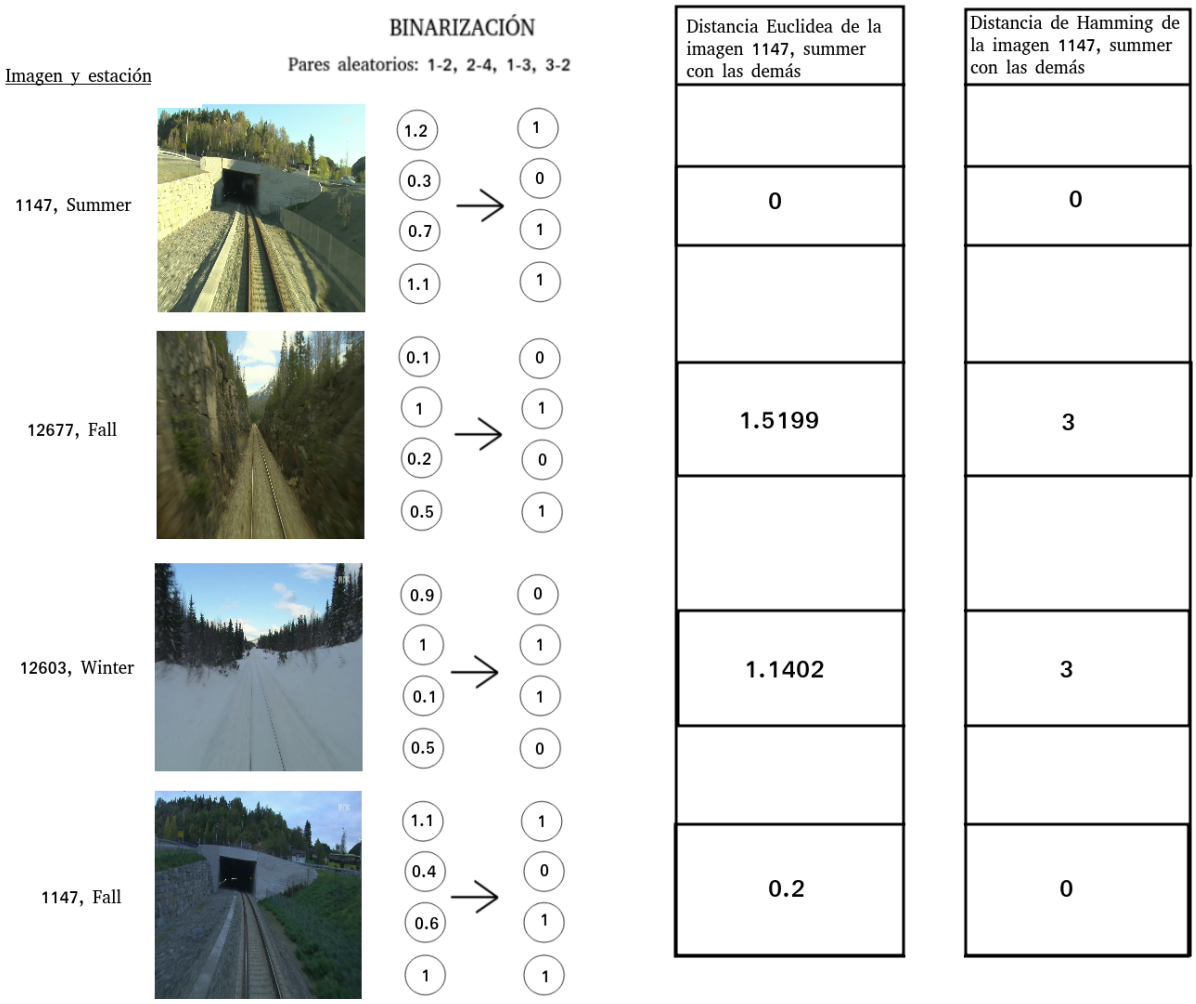


Figura 3.5: Explicación de la robustez de la función de binarización. Se puede apreciar como las imágenes que son iguales pero en diferente estación tienen descriptor reales parecidos y por lo tanto una distancia euclídea pequeña. Cuando se binariza el descriptor, la distancia es incluso menor. De manera contraria la distancia entre imágenes diferentes es también mayor usando descriptores binarios que continuos. Por lo tanto la perdida de información es mínima a pesar de usar descriptores que ocupan mucho menos espacio

Capítulo 4

BASES DE DATOS UTILIZADAS

4.1. MNIST handwritten digits

MNIST handwritten digits es una base de datos de dígitos manuscritos de diez clases, los números del 0 y al 9, frecuentemente utilizada como banco de pruebas en redes neuronales. La base de datos consta de 60.000 imágenes para el entrenamiento y 10.000 para evaluación de los modelos, cada imagen con una resolución de 28×28 en escala de grises.

Figura 4.1: Ejemplo de dígitos del *MNIST handwritten digits. [22]*

Para entrenar la red siamesa es necesario formar pares de imágenes positivos y negativos, es decir, pares de imágenes diferentes, pero del mismo dígito e imágenes diferentes y de diferente dígito, respectivamente. Se han creado 108.400 pares de imágenes con sus respectivas etiquetas, y 17.820 pares de imágenes de testeo.

La evaluación del método de binarización se realiza con la métrica *mean average precision* que se explica en el siguiente capítulo. Para esto, se necesita una base de datos de

Figura 4.2: Par negativo a la izquierda y par positivo a la derecha

consulta y otra base de datos de extracción. Siguiendo los artículos cientificos con mejores resultados hasta la fecha se adoptan las mismas particiones para una justa comparación de los resultados. La base de datos de consulta consta de 1000 imágenes con 100 imágenes de cada categoría. La base de datos de extracción consta de las 69.000 imágenes sobrantes.

4.2. CIFAR-10

CIFAR-10 es una base de datos que consta de 60.000 imágenes en color de 32×32 en 10 clases, con 6.000 imágenes por clase. Hay 50.000 imágenes de entrenamiento y 10.000 imágenes de test.

Las clases son las siguientes: avión, coche, pájaro, gato, ciervo, perro, rana, caballo, barco y camión. En la figura 4.3 se puede visualizar ejemplos de las imágenes de esta base de datos.

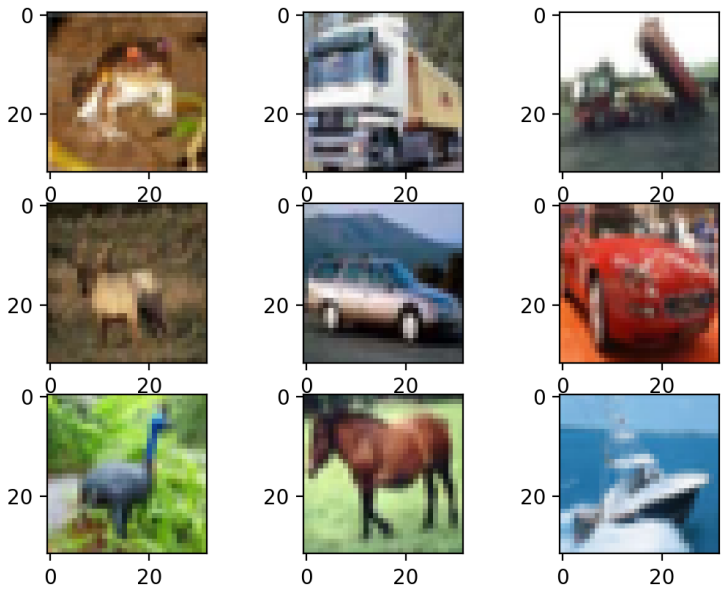


Figura 4.3: Ejemplo de imágenes de la base de datos Cifar-10

De la misma manera que para el conjunto de datos MNIST, la evaluación del método de

binarización se realiza con la métrica *mean average precision* que se explica en el siguiente capítulo. Para esto, se necesita una base de datos de consulta y otra base de datos de extracción. Siguiendo los artículos cientificos con mejores resultados hasta la fecha se adoptan las mismas particiones para una justa comparación de los resultados. La base de datos de consulta consta de 1.000 imágenes con 100 clases de cada categoría. La base de datos de extracción consta de las 59.000 imágenes sobrantes.

4.3. Partitioned Nordland Dataset

Para construir esta base de datos se usan los vídeos del ferrocarril de Nordland. En 2012, la empresa de radiodifusión de Noruega (NRK) realizó un documental sobre el Nordland Railway, una línea ferroviaria entre las ciudades de Trondheim y Bodø. Filmaron el viaje de 729 km con una cámara en la parte delantera del tren en invierno, primavera, otoño y verano. La duración de cada vídeo es de aproximadamente 10 horas y cada fotograma tiene una marca de tiempo con las coordenadas GPS.

La figura 4.4 ilustra la partición de todo el conjunto de imágenes en el conjunto de datos de Nordland. Hay un set de prueba con tres secuencias diferentes de 1.150 imágenes (un total de 3.450, en amarillo en la figura). El resto de imágenes son de entrenamiento (24.569, en rojo en la figura). Al usar varias secciones, aumenta la variedad de lugares y cambios de apariencia contenidos en el conjunto de prueba. También se deja una separación de algunos kilómetros entre cada prueba y tramo de tren descartando algunas imágenes para garantizar la diferencia entre los datos de prueba y de tren.

Dada la similitud entre imágenes consecutivas, en este trabajo se considera que dos imágenes son del mismo lugar si temporalmente están separadas por 3 imágenes o menos. Aplicamos una ventana deslizante de 5 imágenes sobre todo el conjunto de datos para agrupar las imágenes tomadas de cinco segundos consecutivos. Este proceso se puede ver en la siguiente imagen 4.5.

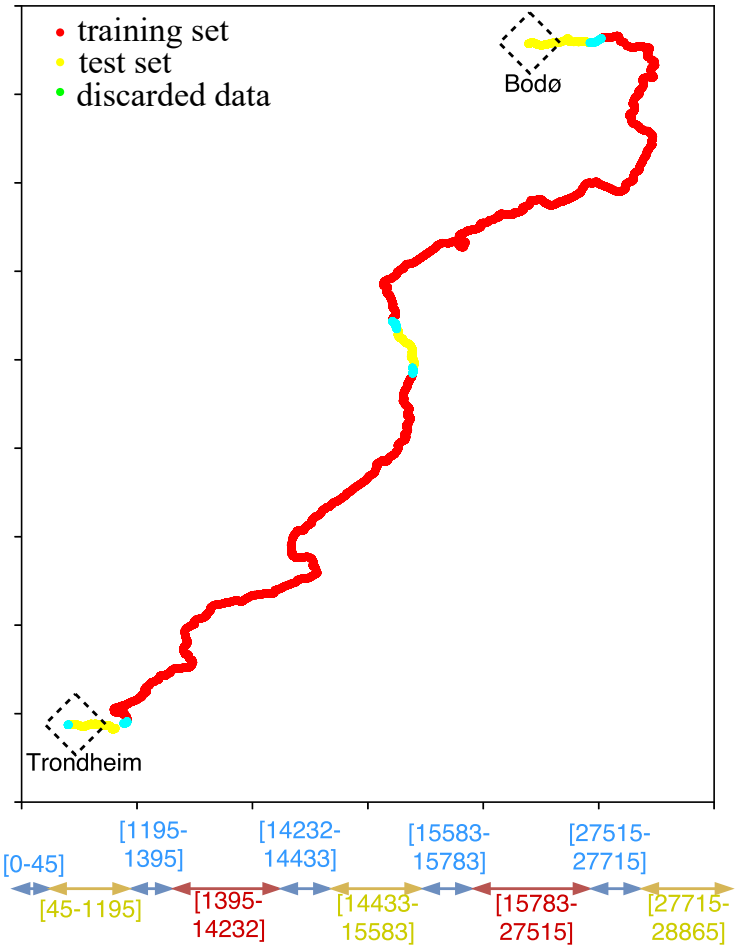


Figura 4.4: Separación de los conjuntos de entrenamiento y test.[24]

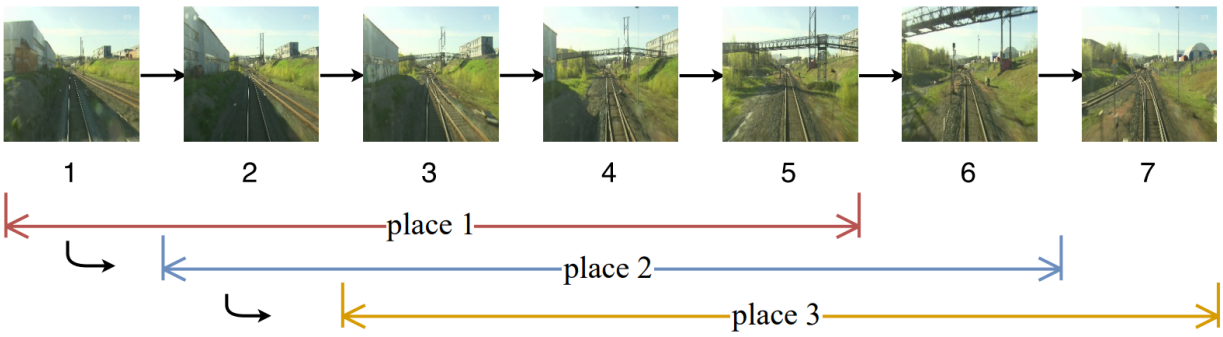


Figura 4.5: Ventana en la que dos imágenes se consideran la misma.[24]

Capítulo 5

EVALUACIÓN Y MÉTRICAS

5.1. Information Retrieval

En ciencias de la computación, *information retrieval* (recuperación de información)[1] o simplemente *retrieval* es la actividad de obtener recursos del sistema de información que son relevantes para una necesidad de información de una colección de esos recursos. Las búsquedas se pueden basar en texto completo u otro índice basado en contenido. *Information retrieval* es la ciencia dedicada a la búsqueda de información en un documento, en los propios documentos y también la búsqueda de los metadatos que describen los datos y las bases de datos de texto, imágenes o sonidos.

Un proceso de *information retrieval* comienza cuando un usuario realiza una consulta al sistema. Las consultas son declaraciones formales de necesidades de información, por ejemplo, cadenas de búsqueda en motores de búsqueda web. En la recuperación de información, una consulta no identifica de forma única un solo objeto en la colección. En cambio, varios objetos pueden coincidir con la consulta, quizás con diferentes grados de relevancia. En nuestra aplicación todos los objetos (imágenes) tienen el mismo grado de relevancia.

Para medir como de satisfactorio es este proceso, la métrica mas común es *mean average precision*, mAP. La *mean average precision* para un conjunto de consultas es la media de las puntuaciones de la *average precision* de cada consulta (query, q). Se puede ver en la fórmula 5.1.

$$\text{mAP} = \frac{\sum_{q=1}^{Q} \text{AP(q)}}{Q} \tag{5.1}$$

Para entender como funciona la métrica *average precision* es imprescindible conocer las

[1]Se mantendrá en lo sucesivo el nombre en inglés por su amplio uso

métricas de *precision* y *recall*.

Precisión: Fracción de las imágenes extraídas que pertenecen a la misma clase que la de la consulta. Formula 5.2

$$\text{precisión} = \frac{|\{\text{imágenes relevantes}\} \cap \{\text{imágenes extraídas}\}|}{|\{\text{imágenes extraídas}\}|} \tag{5.2}$$

Recall: Fracción de las imágenes relevantes para la consulta que se extraen correctamente. Formula 5.3

$$\text{recall} = \frac{|\{\text{imágenes relevantes}\} \cap \{\text{imágenes extraídas}\}|}{|\{\text{imágenes relevantes}\}|} \tag{5.3}$$

Average precision: Precisión y recall son métricas de valor único basadas en la lista completa de imágenes devueltas por el sistema. Para los sistemas que devuelven una secuencia clasificada de imágenes, es conveniente considerar también el orden en el que se presentan las imágenes devueltas. Al calcular una precisión y recuperar en cada posición en la secuencia clasificada de imágenes, se puede trazar una curva de precision-recall, trazando la precisión *p(r)* como función del recall, r. Average precision calcula la media de *p(r)* sobre el intervalo desde r=0 hasta r=1. Esto es el área bajo la curva de precsion-recall. Fórmula 5.4

$$\text{AP} = \int_0^1 p(r)dr \tag{5.4}$$

Esto es equivalente a la fórmula 5.5.

$$\text{AP} = \frac{\sum_{k=1}^{n}(P(k) \times \text{rel}(k))}{\text{número de imágenes relevantes}} \tag{5.5}$$

donde k es el rango de la secuencia de imágenes extraídas, n es el número de imágenes extraídas, *P(k)* es la precisión en el corte k en la lista y *rel(k)* es una función indicadora igual a 1 si la imagen en el rango k es una imagen relevante, y cero en caso contrario.

A continuación se explica el proceso con un ejemplo en detalle: Imaginemos que tenemos una base de datos con imágenes y se hace una consulta de una imagen a la base de datos tal y como la figura 5.1 ilustra.

Se procede calculando las imágenes más similares a la consulta, en este trabajo, la similitud se calcula mediante la distancia de Hamming de la imagen de consulta con cada una de las de la base de datos. De esta manera se obtiene una secuencia de imágenes ordenadas de menor a mayor distancia, G'. Esta secuencia contendrá imágenes que correspondan a la misma categoría que la de la consulta y otras que no, figura 5.2. En el trabajo esta secuencia contiene las 1.000 imágenes con menores distancias, es decir, se calcula la average precision

Figura 5.1: Consulta en una base de datos de imágenes. Q, consulta (query), G, base de datos con las imágenes etiquetadas

en secuencias de 1.000, AP@1000.

Figura 5.2: Secuencia ordenada en la cual hay imágenes que corresponden a la misma categoría que la de la consulta y otras que no

Con esta secuencia de aciertos y fallos lo siguiente es calcular la Average precision para la consulta en concreto. Tal y como se ilustra en la figura 5.3. Se calcula la precisión para cada imagen, teniendo en cuenta si la imagen es relevante o no, de acuerdo a la fórmula 5.5. Nótese que si las 3 imágenes que son relevantes estuvieran al principio la extracción habría sido perfecta y la AP sería 1, en este caso hay fallos en la segunda y tercera posición y finalmente en este ejemplo la AP es 0,7.

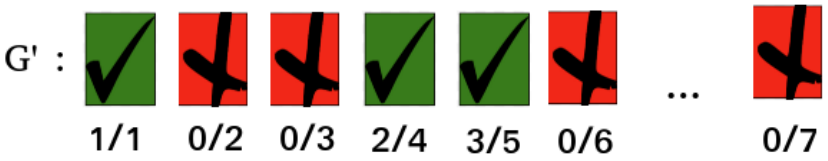


Figura 5.3: Cálculo de Average precision para una secuencia de aciertos y fallos. Se suman los resultados de todas las imagenes: 1+0+0+2/4+3/5+0 ...

Finalmente, se ejecutaría el proceso anterior para todas las imágenes de consulta y se obtendría la media acorde con la ecuación 5.1 para así recibir la mean average precision.

5.2. Reconocimiento de lugares

En la base de datos del Nordland, para comprobar el funcionamiento del reconocedor se necesita una base de datos de referencia y una base de datos de entrada o búsqueda. Por cada imagen de entrada se deberá realizar el procesamiento y la búsqueda del lugar más parecido en la base de referencia. Se puede ver el proceso de comprobación del funcionamiento en la figura 5.4.

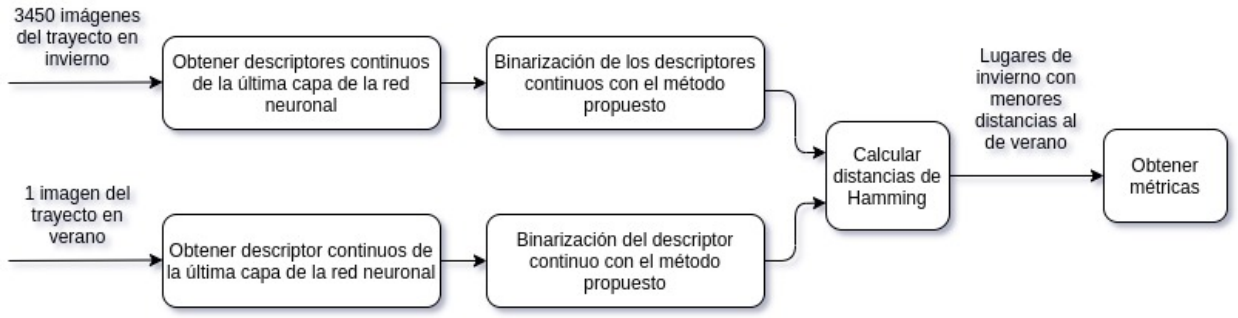


Figura 5.4: Ejemplo del proceso de evaluación del reconocedor. La referencia son las imágenes de los lugares en invierno. La entrada de búsqueda son las imágenes de los mismos lugares en verano.

Para la comprobación se dispone del conjunto de test, el cual cuenta con imágenes de 3.450 lugares distintos, con cada lugar formado por las imágenes extraídas de cada estación durante 5 segundos distintos.

Para evaluar el reconocedor se utilizan las imágenes de los lugares en una estación del año como referencia y en otra estación como entrada. Se anotan los lugares más cercanos para cada imagen de entrada y se obtiene la siguiente métrica:

Fracción de lugares correctos (fc): Es el número de veces que el lugar predicho por el sistema coincide con el lugar de entrada, esto es:

$$fc = \frac{\text{Número de lugares acertados}}{\text{Número de lugares evaluados}} \tag{5.6}$$

Se cuenta como lugar acertado si el más cercano en la base de referencia coincide con el de entrada. Como se explica anteriormente, en este trabajo se considera una ventana de cinco. Por ello se cuenta como acierto si el lugar predicho está entre dos imágenes anteriores y dos posteriores con respecto a la imagen tomada en la misma ubicación exacta.

Capítulo 6

DISEÑO DEL MÉTODO DE BINARIZACIÓN

En las siguentes secciones se explican los experimentos que se han realizado con la finalidad de ajustar el método de binarización para obtener el mejor desempeño posible.

6.1. PCA antes de la binarización por comparación

En este apartado se compara la mean average precision aplicando PCA para reducir la dimensión del descriptor real, y sin aplicar PCA, para el caso de la base de datos de CIFAR-10 (debido a su mayor complejidad respecto al MNIST, en CIFAR-10 se ven las diferencias de una manera más precisa).

	16 bit	32 bit	64 bit
Sin PCA	0,11	0,13	0,11
Con PCA	0,8575	0,9093	0,8769

Tabla 6.1: Comparación de rendimiento (mAP@1000, %) de los algoritmos de binarización en el conjunto de datos CIFAR-10. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes números de bits y según si se aplica PCA o no antes de la función de binarización.

Se puede observar que los resultados sin PCA toman valores cercanos a $0,1$, que es el resultado que obtendría un algoritmo aleatorio en este problema de 10 categorías. Es decir, los descriptores sin aplicar PCA no aportan información para la tarea que se lleva a cabo.

Al aplicar PCA las dimensiones están ordenadas de manera que las primeras explican más varianza que las últimas, de esta manera cada dimensión contiene información relevante de la imagen y la comparación de dimensiones ofrece resultados discriminantes para diferenciar si dos imágenes pertenecen a la misma clase o no.

6.2. Dimensión después de PCA y método de binarización por comparación

Una vez se ha demostrado que el método funciona mejor realizando PCA antes de hacer la binarización por comparación, la siguiente pregunta es, ¿a cuanto debemos reducir la dimensión del descriptor real antes de hacer la binarización por comparación? Se debe tener en cuenta que hay una dimensión mínima a la que se puede reducir según la dimensión del descriptor binario que queramos. Así, si queremos por ejemplo un descriptor binario de 32 bits, la mínima dimensión a la que podemos reducir el descriptor real es 9, con el cual podemos hacer hasta 36 comparaciones sin repetir ninguna.

La binarización por comparación, puede ser aleatoria, o asignando probabilidades de coger unas dimensiones o otras según la importancia que le asignemos.

El primer método que se propone para hacer la binarización por comparación es una distribución probabilística exponencial, asignando de esta manera más probabilidad de coger las primeras dimensiones para la comparación. Esto tiene sentido debido a que PCA ordena las dimensiones de manera que las primeras explican más varianza. La función se ilustra en la figura 6.1

El segundo método propuesto en realizar las comparaciones de manera aleatoria.

En lo siguiente se mostrarán los resultados la binarizacion por comparación aleatoria y con función exponencial con diferentes dimensiones del descriptor real .

Dimensión	Aleatorio	Exponencial
16	0,81028	0,82113
14	0,84013	0,83986
12	0,87691	0,87144

Tabla 6.2: Comparación de rendimiento (mAP@1000, %) de los algoritmos de binarización en el conjunto de datos CIFAR-10 para un descriptor binario de 64 bits. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes dimensiones del descriptor real y método de comparación.

Los mejores resultados se obtienen para la dimensión más baja posible (12 en el caso

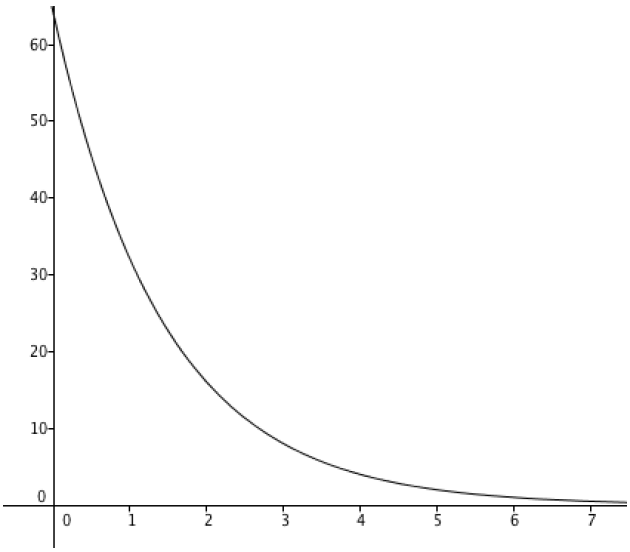


Figura 6.1: Función exponencial probada. Se puede observar que los primeros elementos tendrán más posibilidad de ser cogidos para la comparación que los últimos.

de 64 bits del descriptor binario) y para una comparación de las dimensiones aleatoria. Intuitivamente esto se explica ya que cuanto más pequeña es la dimensión del descriptor real más varianza va a explicar cada dimensión individualmente y por otro lado la aleatoriedad hace tener una diversidad mayor en los pares comparados.

Capítulo 7

RESULTADOS

7.1. MNIST handwritten digits

7.1.1. Arquitectura de red y entrenamiento

Para la evaluación del conjunto de datos MNIST se ha utilizado la red neuronal de la figura 7.1. Esta red consta de dos capas convolucionales con max-pooling y una capa totalmente conectada después de la operación *flatten*, que como se ha explicado en la capitulo 1, asigna un vector unidimensional.

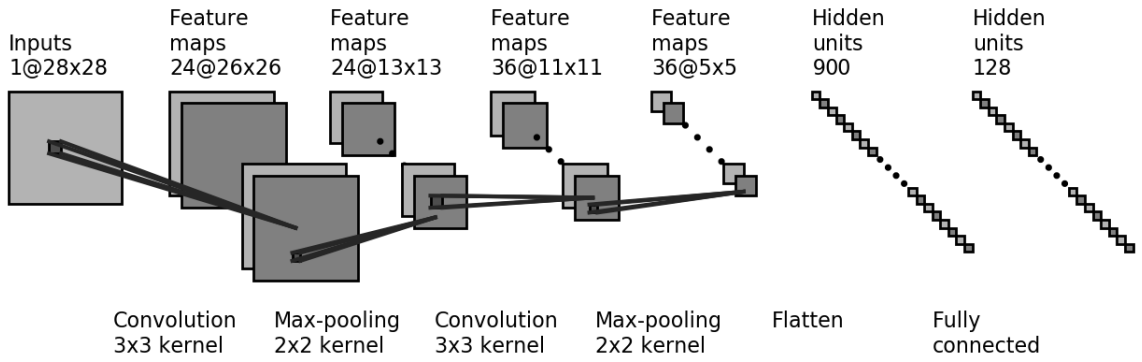


Figura 7.1: Red neuronal usada para el entrenamiento del conjunto de datos MNIST. [16]

El entrenamiento se lleva a cabo mediante una estructura siamesa, cuya estructura se puede observar en la figura 7.2 , y cuyo funcionamiento se explica con detalle en el capitulo 2 sección 2.4.

Una vez entrenada, la red usada individualmente nos permitirá extraer descriptores continuos de dimensión 128 para su posterior binarización y evaluación. Nótese que el entrena-

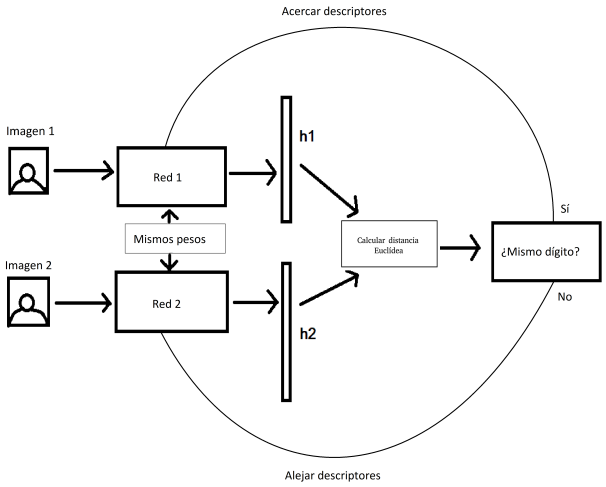


Figura 7.2: Ejemplo de red siamesa y su funcionamiento.

miento de la red es muy importante: aunque se tenga un buen método de binarización, si el descriptor continuo no es capaz de representar correctamente las imágenes, tampoco lo hará el descriptor binario.

Todo esto se ha implementado en Python usando las librerías Keras y TensorFlow.

Figura 7.3: Logotipo de Tensorflow y Keras

7.1.2. Resultados

Con el propósito de comparar el método desarrollado con los trabajos de binarización hechos hasta la fecha, se realiza la tarea de retrieval, explicada anteriormente, y con la

misma evaluación, *mean average precision* para los 1000 primeros elementos devueltos.

Método	Referencia	16 bit	32 bit	64 bit
DeepBit [17]	CVPR16	28.18	32.02	44.53
LSH [6]	VLDB00	42.10	50.45	66.23
UTH [12]	ACM17	43.15	46.58	49.88
SphH [11]	CVPR12	52.97	65.45	65.45
SpeH [31]	CVPR12	59.72	64.37	67.60
ITQ [7]	PAMI13	70.06	76.86	80.23
HashGAN [5]	CVPR18	94.31	95.48	96.37
Nuestro	Propuesto	**99.16**	**99.20**	**99.08**

Tabla 7.1: Comparación de rendimiento (mAP@1000, %) de diferentes algoritmos de binarización en el conjunto de datos MNIST. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes números de bits.

7.2. CIFAR-10

7.2.1. Arquitectura de red y entrenamiento

Para la evaluación del conjunto de datos CIFAR-10 se ha utilizado la red neuronal *RESNETV2 50*, la cual puede visualizarse en la figura 7.4.

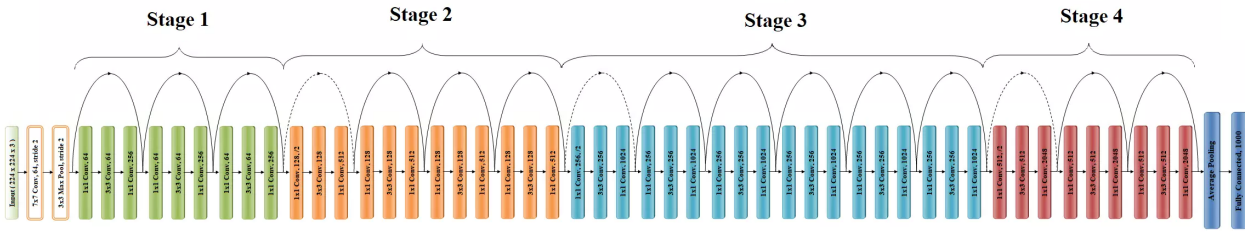


Figura 7.4: Red neuronal Resnetv2 50

La arquitectura es la siguiente: Primero se realiza la convolución inicial y la agrupación máxima utilizando tamaños de kernel de 7 7 y 3 3, respectivamente. Luego, comienza la Etapa 1 de la red la cual contiene tiene 3 bloques residuales que contienen 3 capas cada uno. El tamaño del kernel utilizado para realizar la operación de convolución en las 3 capas del bloque de la etapa 1 es 64, 64 y 256 respectivamente. Las flechas curvas se refieren a la conexión de identidad, la cual es caracteristica de la red ResNet y ayuda a que no desaparezca el gradiente cuando se entrena la red. La Etapa 2 de la red contiene 4 bloques residuales con también 3 capas cada uno. El tamaño del kernel utilizado es 128,128 y 512 respectivamente.

La Etapa 3 contiene 6 bloques residuales con 3 capas cada uno. El tamaño del kernel utilizado esta vez es 256, 256 y 1024 respectivamente. Por último tenemos la etapa 4 que consta de 3 bloques residuales con 3 capas cada uno. El tamaño del kernel utilizado es 512,512 y 2048. Finalmente una capa adaptativa de maxpool convierte la salida en un vector unidimensional y se termina con una capa convolucional que pasa de 2048 a un capa de 10 neuronas donde se hace la clasificación.

Los descriptores de cada imagen para llevar a cabo el proceso de binarización se extraen de la ultima capa de la ultima etapa, estos tienen dimensión 32768.

El entrenamiento se ha llevado a cabo para la tarea de clasificación, tal y como se explica con detalle en el capitulo 2 sección 2.3. Los parámetros entrenables han sido adquiridos de [23].

Todo esto se ha implementado con la librería Pytorch.

Figura 7.5: Logotipo de Pytorch

7.2.2. Resultados

Al igual que con el conjunto de datos MNIST y con el propósito de comparar el método desarrollado con los trabajos de binarización hechos hasta la fecha, se realiza la tarea de retrieval, explicada anteriormente, con la misma evaluación, *mean average precision* para los 1000 primeros elementos devueltos.

Se puede observar que en nuestro trabajo el mejor resultado se obtiene para 32 bits. Esto se debe a que el descriptor de 32 bits binario es lo suficientemente descriptivo como para contener toda la información de las imágenes, por lo tanto, añadir más dimensiones no aporta información adicional sino que añade ruido, ya que las nuevas comparaciones son con información no relevante y esto lleva a que imágenes de la misma categoría puedan tener distancias de Hamming más grandes y viceversa, por lo tanto la precisión es menor.

Método	Referencia	16 bit	32 bit	64 bit
DGH [19]	NIPS14	19.9	20.0	21.2
ITQ [7]	PAMI13	20.1	20.7	23.5
AGH [20]	ICML11	21.7	20.5	18.2
KSH [18]	CVPR12	45.1	47.3	50.7
ITQ-CCA [8]	TPAMI12	46.3	49.8	50.5
CNNH [32]	AAAI14	45.3	50.9	53.7
SDH [27]	CVPR15	49.9	52.5	54.6
DNNH [15]	CVPR15	55.6	55.8	59.9
DHN [34]	AAAI16	56.4	60.3	62.6
HashNet [4]	TPAMI17	64.3	67.5	68.7
HashGAN [3]	CVPR18	66.8	73.1	74.9
PGDH [33]	ECCV18	74.1	74.7	76.2
MIHash [1]	ICCV17	76.0	77.6	76.1
GreedyHash [29]	NIPS18	78.6	81.0	83.3
JMLH [28]	IEEE19	80.5	84.1	83.7
Nuestro	Propuesto	**84.55**	**90.97**	**89.92**

Tabla 7.2: Comparación de rendimiento (mAP@1000, %) de diferentes algoritmos de binarización en el conjunto de datos CIFAR-10. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes números de bits.

7.3. Partitioned Nordland Dataset

7.3.1. Arquitectura de red y entrenamiento

Para la evaluación del conjunto de datos Nordland se ha utilizado la red neuronal VGG-16, la cual puede visualizarse en la figura 7.6.

Los descriptores de cada imagen para llevar a cabo el proceso de binarización se extraen de la ultima capa ya que al tratarse de una estructura triplet, al igual que en el caso del MNIST la red no tiene una ultima capa de clasificación. La dimensión de los descriptores que se extraein es de 128.

El entrenamiento se realiza mediante una estructura triplet con ajuste fino de la red VGG-16 Places. Añadiendo una capa totalmente conectada con 128 dimensiones y entrenando mediante la función de error de Wohlhart-Lepetit. Los parametros entrenables han sido adquiridos de [24].

Todo esto se ha ejecutado con la librería de Caffe.

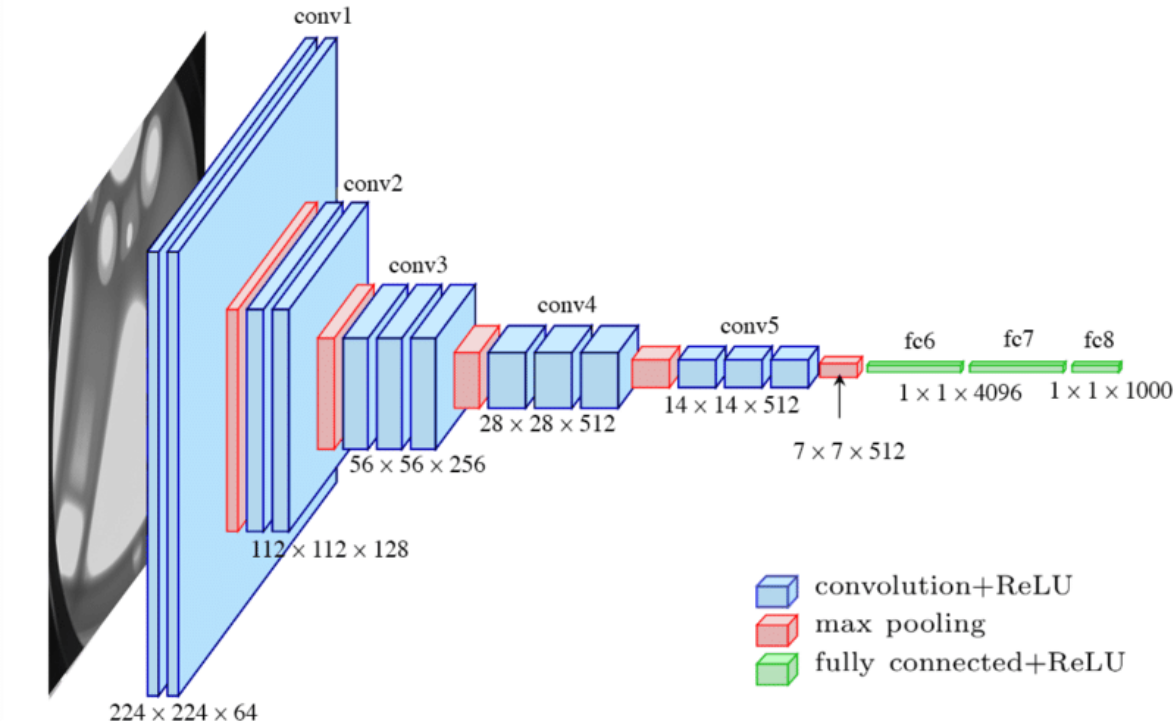


Figura 7.6: Red neuronal VGG-16.[24]

Caffe

Figura 7.7: Logotipo de Caffe

7.3.2. Resultados

A continuación se pueden comprobar en las imágenes 7.8 7.9 7.10 7.11 los resultados para diferente diferentes números de bits binarios: 64, 128, 256, 512, 1024, 2048, 4096. En la última columna se representan los resultados del estado del arte los cuales son con descriptores reales de dimensión 128 (4096 bits ya que son de tipo floating point). Es por eso que se ha decidido no comparar valores mayores que este número de bits. Se puede apreciar que nuestro algoritmo de binarización aproxima valores State-Of-the-Art, llegando incluso a superarlos

en algunos casos. A partir de valores de descriptor de 1024 los resultados son prácticamente iguales a los resultados State-Of-the-Art.

Los mejores resultados se corresponden con el descriptor de 4096 bits, cuyos resultados se pueden ver en la tabla 7.3 .

Resultados para 4096 bits				
input/referencia	summer	fall	winter	spring
summer	-	97,47	83,08	93,68
fall	97,44	-	85,53	94,4
winter	84,35	83,74	-	86,75
spring	94,36	93,65	86,23	-

Tabla 7.3: Comparación de la fracción de lugares correctos (fc, %) para descriptor de 4096 bits binario.

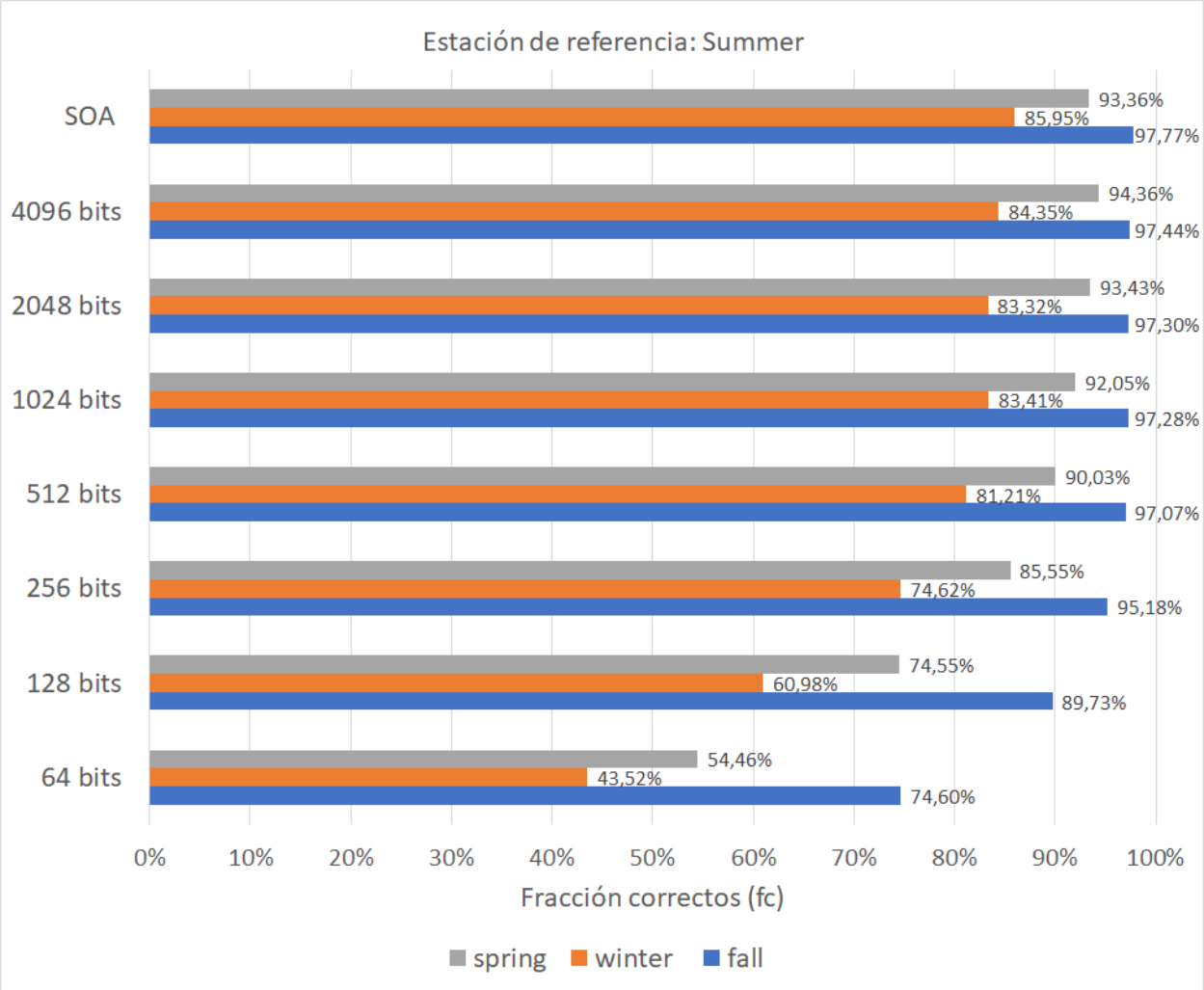


Figura 7.8: Resultados con el verano como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua.

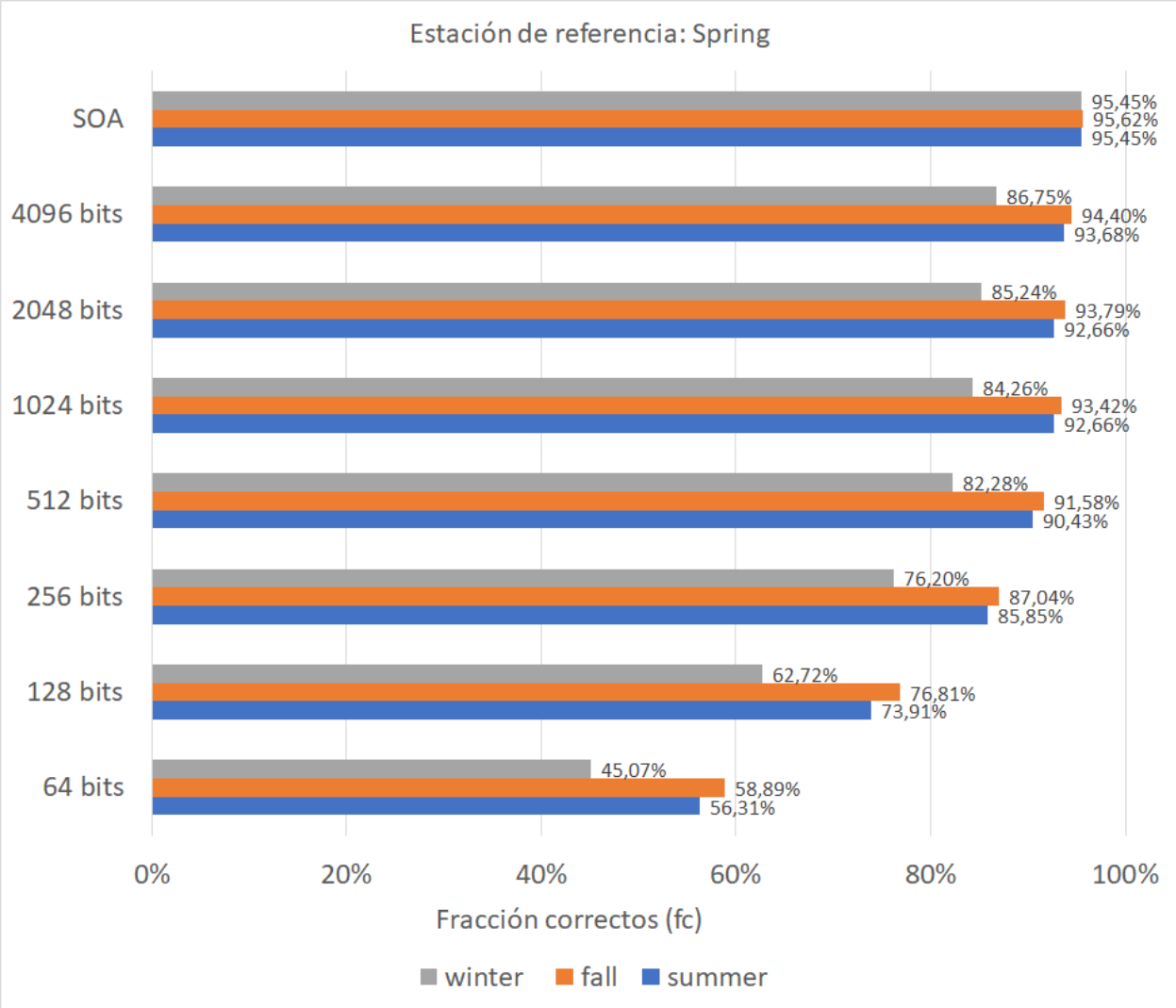


Figura 7.9: Resultados con la primavera como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua.

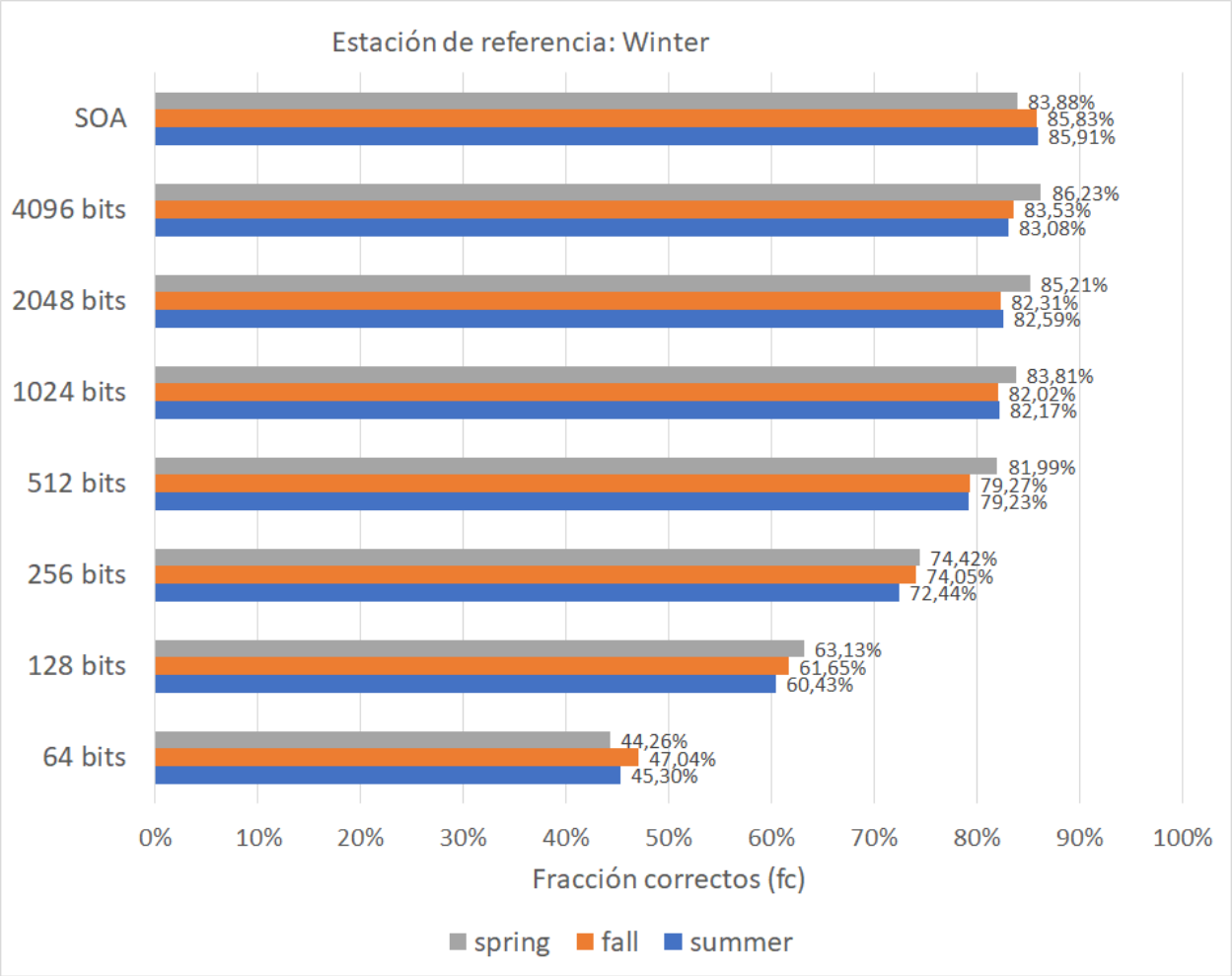


Figura 7.10: Resultados con el invierno como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua.

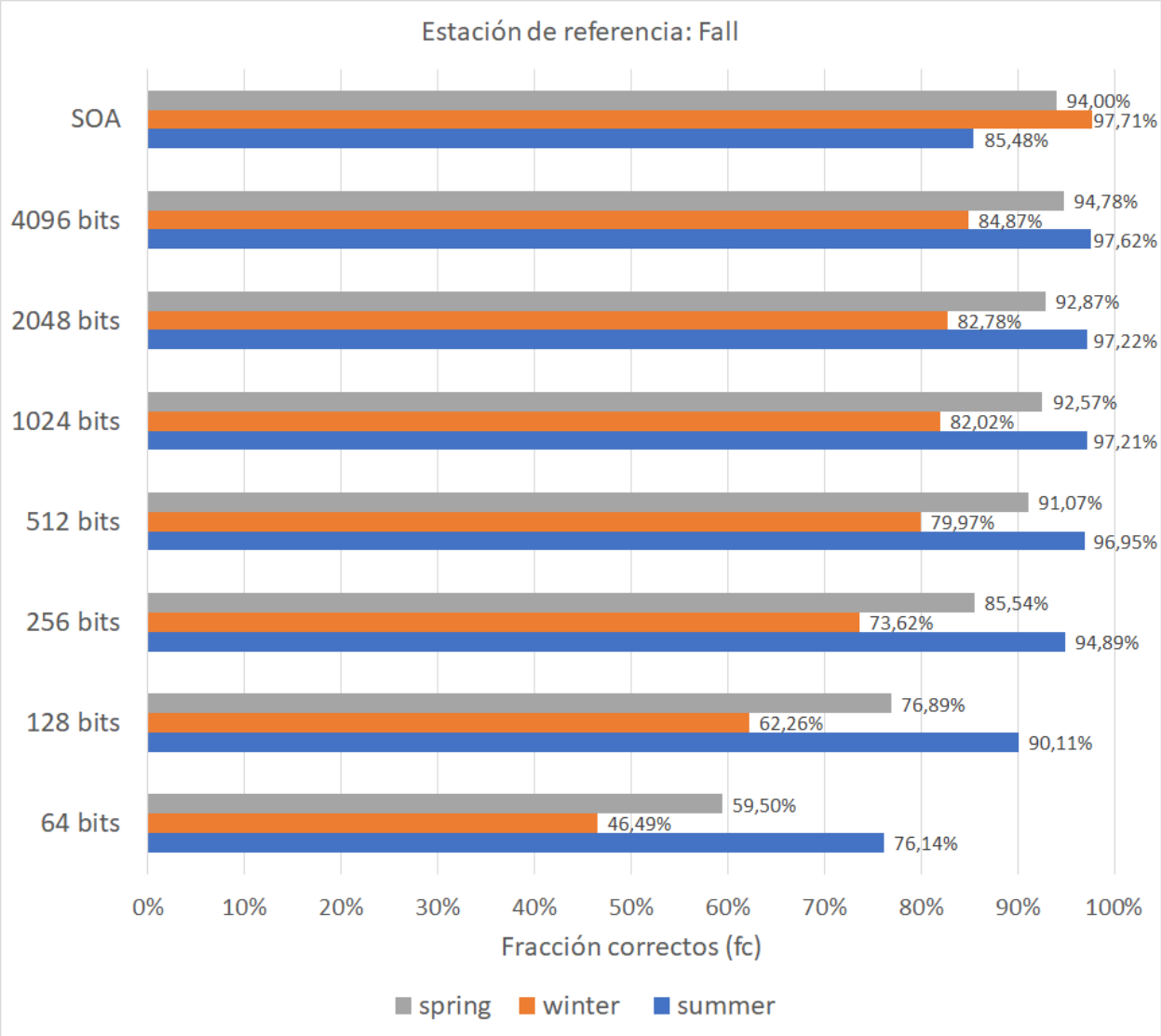


Figura 7.11: Resultados con el otoño como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua.

Capítulo 8

CONCLUSIÓN

En este trabajo se ha propuesto la binarización de descriptores continuos generados a partir de redes neuronales. El método propuesto muestra que se pueden obtener descriptores binarios con una precisión prácticamente similar al caso continuo mediante comparación de las dimensiones del descriptor continuo. Se ha demostrado que el método aumenta drásticamente su precisión si se realiza PCA antes de realizar las comparaciones.

La evaluación se ha llevado a cabo en tres bases de datos con diferentes características, para así, explorar las limitaciones del método a medida que la complejidad de la base de datos aumenta. La base de datos *MNIST* testea el método con imágenes en escala de grises, *CIFAR-10*, con imágenes en color de diferentes objetos, y finalmente, *Partitioned Nordland* con imágenes de reconocimiento de lugares reales. Los resultados obtenidos han sido muy prometedores, superando nuestro método al estado del arte en las bases de datos del *MNIST* y *CIFAR-10*. En el caso de la base de datos del *Partitioned Nordland*, al no haber ningún método de binarización con resultados, se ha comparado con el estado del arte en caso continuo. Los resultados en esta base de datos también han sido excelentes, ya que, se supera la precisión en algunas estaciones y se obtienen resultados similares en el resto de casos.

Se ha utilizado una arquitectura de red y entrenamiento diferente en cada base de datos para ajustar la complejidad del modelo a los datos disponibles en cada caso. Para la base de datos *MNIST* se ha entrenado una estructura siamesa, para el caso de *CIFAR-10* se ha usado una red entrenada para clasificación y finalmente para *Partitioned Nordland* se han usado los descriptores resultantes del entrenamiento de una estructura triplet.

Como líneas futuras de trabajo se propone estudiar la eficacia del método en reconocimiento de objetos no sólo en bases de datos de imágenes más complejas sino también en vídeo, donde la rapidez de cálculo de los descriptores binarios puede jugar un papel determinante para la aplicación a sistemas de funcionamiento en tiempo real.

Capítulo 9

HERRAMIENTAS

Para trabajar con las redes se han utilizado las librerías de , Tensorflow y Keras [21], Pytorch [26] y Caffe [14],por su versatilidad a la hora de diseñar y probar distintas estructuras de redes neuronales. Para el resto de procedimientos como la comprobación del funcionamiento o la elaboración de gráficas se ha utilizado Python, especialmente las librerías Numpy [25], SciPy [30] y MatPlotLib [13].

Bibliografía

[1] F. Cakir y col. "MIHash: Online Hashing with Mutual Information". En: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, págs. 437-445. DOI: 10.1109/ICCV.2017.55.

[2] Michael Calonder y col. "BRIEF: Computing a local binary descriptor very fast". En: *IEEE transactions on pattern analysis and machine intelligence* 34 (nov. de 2011). DOI: 10.1109/TPAMI.2011.222.

[3] Y. Cao y col. "HashGAN: Deep Learning to Hash with Pair Conditional Wasserstein GAN". En: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, págs. 1287-1296. DOI: 10.1109/CVPR.2018.00140.

[4] Z. Cao y col. "HashNet: Deep Learning to Hash by Continuation". En: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, págs. 5609-5618. DOI: 10.1109/ICCV.2017.598.

[5] Kamran Ghasedi Dizaji y col. "Unsupervised deep generative adversarial hashing network". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, págs. 3664-3673.

[6] Aristides Gionis, Piotr Indyk y Rajeev Motwani. "Similarity Search in High Dimensions via Hashing". En: *Proceeding VLDB '99 Proceedings of the 25th International Conference on Very Large Data Bases* 99 (mayo de 2000).

[7] Y. Gong y col. "Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), págs. 2916-2929. DOI: 10.1109/TPAMI.2012.193.

[8] Y. Gong y col. "Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), págs. 2916-2929. DOI: 10.1109/TPAMI.2012.193.

[9] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[10] Raia Hadsell, Sumit Chopra y Yann LeCun. "Dimensionality Reduction by Learning an Invariant Mapping". En: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*. CVPR '06. USA: IEEE Computer Society, 2006, págs. 1735-1742. ISBN: 0769525970. DOI: 10.1109/CVPR.2006.100. URL: https://doi.org/10.1109/CVPR.2006.100.

[11] J. Heo y col. "Spherical hashing". En: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, págs. 2957-2964. DOI: 10.1109/CVPR.2012.6248024.

[12] Shanshan Huang y col. "Unsupervised Triplet Hashing for Fast Image Retrieval". En: (feb. de 2017).

[13] J. D. Hunter. "Matplotlib: A 2D graphics environment". En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.

[14] Yangqing Jia y col. "Caffe: Convolutional Architecture for Fast Feature Embedding". En: *arXiv preprint arXiv:1408.5093* (2014).

[15] Hanjiang Lai y col. "Simultaneous Feature Learning and Hash Coding with Deep Neural Networks". En: (abr. de 2015).

[16] Yann Lecun y col. "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE*. 1998, págs. 2278-2324.

[17] Kevin Lin y col. "Learning compact binary descriptors with unsupervised deep neural networks". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, págs. 1183-1192.

[18] W. Liu y col. "Supervised hashing with kernels". En: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, págs. 2074-2081. DOI: 10.1109/CVPR.2012.6247912.

[19] Wei Liu y col. "Discrete Graph Hashing". En: *Advances in Neural Information Processing Systems 27*. Ed. por Z. Ghahramani y col. Curran Associates, Inc., 2014, págs. 3419-3427. URL: http://papers.nips.cc/paper/5332-discrete-graph-hashing.pdf.

[20] Wei Liu y col. "Hashing with Graphs." En: ene. de 2011, págs. 1-8.

[21] Martın Abadi y col. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[22] *MS Windows NT Kernel Description*. https://github.com/cazala/mnist. Accessed: 2020-09-14.

[23] *MS Windows NT Kernel Description*. https://storage.googleapis.com/bit_models/. Accessed: 2020-05-20.

[24] Daniel Olid, José M. Fácil y Javier Civera. "Single-View Place Recognition under Seasonal Changes". En: *PPNIV Workshop at IROS 2018*. 2018.

[25] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[26] Adam Paszke y col. "Automatic differentiation in PyTorch". En: (2017).

[27] F. Shen y col. "Supervised Discrete Hashing". En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, págs. 37-45. DOI: 10.1109/CVPR.2015.7298598.

[28] Yuming Shen y col. "Embarrassingly simple binary representation learning". En: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019, págs. 0–0.

[29] Shupeng Su y col. "Greedy Hash: Towards Fast Optimization for Accurate Hash Coding in CNN". En: *Advances in Neural Information Processing Systems 31*. Ed. por S. Bengio y col. Curran Associates, Inc., 2018, págs. 798-807. URL: http://papers.nips.cc/paper/7360-greedy-hash-towards-fast-optimization-for-accurate-hash-coding-in-cnn.pdf.

[30] Pauli Virtanen y col. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". En: *Nature Methods* 17 (2020), págs. 261-272. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[31] Yair Weiss, Antonio Torralba y Rob Fergus. "Spectral Hashing". En: *Advances in Neural Information Processing Systems 21*. Ed. por D. Koller y col. Curran Associates, Inc., 2009, págs. 1753-1760. URL: http://papers.nips.cc/paper/3383-spectral-hashing.pdf.

[32] R. Xia y col. "Supervised hashing for image retrieval via image representation learning". En: *Proceedings of the National Conference on Artificial Intelligence* 3 (ene. de 2014), págs. 2156-2162.

[33] Xin Yuan y col. "Relaxation-Free Deep Hashing via Policy Gradient". En: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sep. de 2018.

[34] H. Zhu y col. "Deep Hashing Network for Efficient Similarity Retrieval". En: *AAAI*. 2016.

Índice de figuras

1.1. Ejemplo de reconocimiento de imágenes 6

2.1. Neurona artificial. 10
2.2. Ejemplo de capa totalmente conectada 11
2.3. Ejemplo de capa localmente conectada 11
2.4. Ejemplo de funcionamiento de la operación de convolución llevada a cabo por la capa convolucional. El filtro es desplazado por toda la imagen multiplicando todos los píxeles del Input y resultando la nueva matriz ilustrada como I * K 12
2.5. Ejemplo de capa reductora. Se ilustra la operación de *max pooling* la cual consiste en escoger el máximo de una ventana. 12
2.6. Ejemplo de funcionamiento de la capa *flatten* 13
2.7. Ejemplo numérico de predicción de los dígitos del dataset MNIST. Tras atravesar la red neuronal sin entrenar se puede apreciar que las predicciones no son correctas (primera columnas de neuronas), la siguiente columna de neuronas ilustra lo que idealmente debería de resultar de la predicción. El error se calcula con una función de coste cuadrática. 15
2.8. Ejemplo de red siamesa y su funcionamiento. h1 y h2 representan los descriptores de cada imagen al final de la red. 16
2.9. Ejemplo de red triplet y su funcionamiento. $h1$, $h2$ y $h3$ representan los descriptores de cada imagen al final de la red. 17

3.1. Método de binarización 18
3.2. Distribuciones de datos y sus matrices de covarianza 2x2. 19
3.3. Diagonalización de la matriz de covarianza. 21
3.4. Ilustración sencilla de función de binarización 22

3.5. Explicación de la robustez de la función de binarización. Se puede apreciar como las imágenes que son iguales pero en diferente estación tienen descriptor reales parecidos y por lo tanto una distancia euclídea pequeña. Cuando se binariza el descriptor, la distancia es incluso menor. De manera contraria la distancia entre imágenes diferentes es también mayor usando descriptores binarios que continuos. Por lo tanto la perdida de información es mínima a pesar de usar descriptores que ocupan mucho menos espacio 24

4.1. Ejemplo de dígitos del *MNIST handwritten digits. [22]* 25
4.2. Par negativo a la izquierda y par positivo a la derecha 26
4.3. Ejemplo de imágenes de la base de datos Cifar-10 26
4.4. Separación de los conjuntos de entrenamiento y test.[24] 28
4.5. Ventana en la que dos imágenes se consideran la misma.[24] 28

5.1. Consulta en una base de datos de imágenes. Q, consulta (query), G, base de datos con las imágenes etiquetadas 31
5.2. Secuencia ordenada en la cual hay imágenes que corresponden a la misma categoría que la de la consulta y otras que no 31
5.3. Cálculo de Average precision para una secuencia de aciertos y fallos. Se suman los resultados de todas las imagenes: 1+0+0+2/4+3/5+0 31
5.4. Ejemplo del proceso de evaluación del reconocedor. La referencia son las imágenes de los lugares en invierno. La entrada de búsqueda son las imágenes de los mismos lugares en verano. 32

6.1. Función exponencial probada. Se puede observar que los primeros elementos tendrán más posibilidad de ser cogidos para la comparación que los últimos. 35

7.1. Red neuronal usada para el entrenamiento del conjunto de datos MNIST. [16] 36
7.2. Ejemplo de red siamesa y su funcionamiento. 37
7.3. Logotipo de Tensorflow y Keras 37
7.4. Red neuronal Resnetv2 50 38
7.5. Logotipo de Pytorch 39
7.6. Red neuronal VGG-16.[24] 41
7.7. Logotipo de Caffe 41
7.8. Resultados con el verano como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua. 43

7.9. Resultados con la primavera como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua. 44
7.10. Resultados con el invierno como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua. 45
7.11. Resultados con el otoño como estación de referencia para distintas dimensiones de descriptor binario y, en la primera fila los resultados State Of the Art (SOA) con dimensión 128 continua. 46

Índice de cuadros

6.1. Comparación de rendimiento (mAP@1000, %) de los algoritmos de binarización en el conjunto de datos CIFAR-10. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes números de bits y según si se aplica PCA o no antes de la función de binarización. 33
6.2. Comparación de rendimiento (mAP@1000, %) de los algoritmos de binarización en el conjunto de datos CIFAR-10 para un descriptor binario de 64 bits. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes dimensiones del descriptor real y método de comparación. 34

7.1. Comparación de rendimiento (mAP@1000, %) de diferentes algoritmos de binarización en el conjunto de datos MNIST. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes números de bits. 38
7.2. Comparación de rendimiento (mAP@1000, %) de diferentes algoritmos de binarización en el conjunto de datos CIFAR-10. Esta tabla muestra la precisión media media (mAP) de las 1.000 imágenes devueltas principales con respecto a diferentes números de bits. 40
7.3. Comparación de la fracción de lugares correctos (fc, %) para descriptor de 4096 bits binario. 42