Universidad Zaragoza

Trabajo Fin de Master

Binarización de descriptores en redes neuronales de manera supervisada y no supervisada

Binarization of embeddings in neural networks with and without supervision

Autor/es

Pablo Peribáñez Sos

Director/es

Javier Civera Sancho

Escuela de ingenieria y arquitectura,Universidad de Zaragoza.

2021

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. ,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)

(Título del Trabajo)

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

Zaragoza,

Fdo:

Agradecimientos

Querría agradecer una vez más al Dr. Javier Civera por contar conmigo para la realización de un Trabajo de Fin de Estudios. Su dedicación tanto en el Trabajo de Fin de Grado como en este me ha permitido profundizar en temas de relativa importancia hoy en día y con un gran futuro.

Resumen

La binarización de descriptores en redes neuronales profundas es una técnica de reciente aparición, que ha surgido de la necesidad de reducir la huella computacional (en procesamiento y almacenamiento) de algoritmos de *image retrieval*. Entendemos por *image retrieval* la aplicación cuyo objetivo es encontrar imágenes, similares a una imagen *query* proporcionada por el usuario, dentro una base de datos de tamaño considerable. Esta reducción de la huella computacional permitiría trabajar en tiempo real a alta frecuencia y/o con bases de datos más grandes, ampliando así el rango de aplicaciones.

La binarización de descriptores consiste en transformar la salida de una red neuronal, consistente en un vector $y \in \mathbb{R}^p$ formado por p números reales en un descriptor binario $y \in \mathbb{B}^q$ formado por q elementos binarios. Nótese que un número real se representa en memoria con 32 o 64 BITS, mientras que un elemento binario solo necesita 1 BIT. La reducción de la huella computacional viene dada por este menor tamaño en memoria y una comparación más eficaz mediante la denominada distancia de Hamming.

La reducción en el tamaño al cambiar de un descriptor al otro es posible por el hecho de que el descriptor continuo presenta redundancia y no todas sus dimensiones son significativas. Este trabajo pretende aprovechar este supuesto y estudiar, implementar y evaluar una serie de técnicas previas a la binarización. Las técnicas se focalizarán en el pre-procesamiento de la imagen, antes del proceso conocido como *embedding*, como en el post-procesamiento del descriptor continuo antes de ser transformado en binario. También se introducirá una variante en el propio proceso de binarización.

Al tratarse de una técnica de reciente estudio, la literatura del tema es escasa. A este reto se le suma encontrar bases de datos adecuadas e implementar una arquitectura de red que permita alcanzar una alta precisión para poder centrar el estudio en el proceso de binarización. Se han utilizado dos bases de datos de uso común en este campo, *MNIST Handwritten Digits* y *CIFAR-10* y se han obtenido resultados que superan la precisión obtenida por algoritmos recientemente publicados.

Índice general

Índice de figuras

Índice de Tablas

1. Introducción

Este trabajo fin de máster se enmarca dentro de la visión por computador, disciplina que abarca una serie de técnicas de adquisición, procesamiento, análisis e interpretación de imágenes mediante sistemas computacionales y de forma automática. Las aplicaciones que hacen uso de la visión por computador han crecido rápidamente durante las últimas décadas. Podemos encontrar ejemplos en la industria tradicional, como el reconocimiento de defectos, y en entornos emergentes como el de los vehículos autónomos, con la localización y mapeo simultáneos (SLAM) así como en nuestros propios teléfonos móviles, con el reconocimiento facial entre otros.

Figura 1.1: Ejemplo de aplicación de *image retrieval* utilizando la base de datos *MNIST Handwritten Digits*. El usuario aporta una imagen y el objetivo es que el sistema devuelva las imágenes similares de la base de datos.

En este trabajo, concretamente, se va a trabajar con redes neuronales profundas para el reconocimiento de patrones visuales. Más específicamente, el estudio está centrado en el problema denominado *image retrieval*. El objetivo del *image retrieval* consiste en recuperar imágenes relacionadas o similares a una imagen *query* aportada por el usuario (Figura 1.1). Las redes neuronales profundas, que son las encargadas de reconocer los patrones visuales, se entrenan previamente con extensas bases de datos. A diferencia de otras aplicaciones, en *image retrieval* no se busca una clasificación final en un conjunto discreto de clases sino obtener una representación comprimida de cada imagen para poder compararlas y encontrar las más similares de acuerdo a su contenido semántico o de alto nivel.

Esta comparación, que se realiza mediante la computación de la distancia entre pares de imágenes, supone un gran coste computacional debido al espacio que ocupan estos descriptores en memoria. La reducción de su tamaño conlleva, en general, una pérdida de información que se traduce en un peor desempeño. Existen diversos estudios recientes que se centran en encontrar un compromiso entre coste y desempeño, y una de las técnicas más prometedoras consiste en binarizar la información extraída de cada imagen, almacenarla en descriptores binarios (por tanto más compactos) y realizar la comparación mediante la distancia de Hamming (más eficiente que la distancia euclídea en los procesadores actuales).

La transformación de cada dimensión del descriptor de un número real (representado por 32 ó 64 BITS) a un solo BIT (0 ó 1) supone una pérdida en la capacidad del descriptor de representar la información de la imagen. Como consecuencia, se debe lograr obtener un método que retenga la máxima información posible en el descriptor continuo y se logre transmitir al descriptor discreto.

El objetivo de este trabajo nace de esta necesidad y en concreto, el trabajo realizado parte de una implementación del estado del arte y propone el estudio de varias alternativas de procesamiento: el preprocesamiento de la imagen mediante las técnicas de reducción por análisis de componentes principales y de blanqueamiento de imágenes para reducir el número de variables en el entrenamiento de la red y consecuentemente reducir el tiempo de entrenamiento; la reducción y reordenamiento de las dimensiones del descriptor continuo mediante la aplicación de discriminante lineal y la implementación de una binarización exponencial que escoja pares de dimensiones del descriptor atendiendo a la relevancia de información que aporta cada dimensión.

1.1. Estructura del trabajo

Una vez introducido el contexto en el que se enmarca este trabajo y sus objetivos, en el capítulo 2 se procederá a explicar unos conceptos de redes neuronales necesarios para comprender el resto del estudio. En el capítulo 3, se detallarán los métodos y técnicas diseñadas y propuestas para el objetivo del trabajo. Las bases de datos empleadas para comprobar la precisión de las técnicas empleadas, así como la métrica de evaluación, se explicarán en el capítulo 4. Por último, el capítulo 5 resumirá los hallazgos encontrados y su interpretación.

2. Redes Neuronales

Este capítulo contiene conceptos básicos sobre redes neuronales, necesarios para comprender los algoritmos desarrollados y sus resultados.

El objetivo principal de una red neuronal es resolver problemas de una forma similar, a alto nivel, al cerebro humano. Las redes actuales suelen contener desde unos miles a unos pocos millones de unidades neuronales con cierta semejanza con las neuronas humanas. Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, por ello, se incluyen en la disciplina de *Machine Learning*. Sobresalen en áreas donde las soluciones a ciertos problemas son difíciles de representar con programas convencionales.

Las primeras publicaciones datan de la década de 1940 [1] pero no fue hasta finales de siglo cuando empezó a explotar su potencial. Esto fue debido, por un lado, a la introducción algoritmos que resolvían el problema de entrenamiento rápido en redes de múltiples capas como el algoritmo de propagación hacia atrás [2] y el max-pooling [3] y por otro lado, al incremento de potencia y memoria de ordenadores y el desarrollo de tarjetas gráficas (GPUs).

2.1. Estructura

Una red artificial consiste en un conjunto de unidades simples de procesamiento que se comunican enviando señales entre sí a través de una gran cantidad de conexiones ponderadas. La figura 2.1 representa el esquema básico de una red neuronal. Los componentes básicos son los siguientes:

- un conjunto de unidades de procesamiento ("neurona");
- un estado de activación a_j para cada unidad;
- una función de activación $h(\cdot)$ que transforma el estado de activación a_j en el output de cada unidad z_j;
- conexiones entre unidades. Generalmente cada conexión es definida por un peso w_{kj}, el cual determina el efecto que tiene la señal de la unidad j en la unidad activada z_k;
- una regla de propagación que determina el estado de activación a_j de una unidad respecto sus inputs externos;
- un input externo o *bias*, w_{j0}

En la mayoría de los casos se asume que cada unidad proporciona una contribución al input de la unidad a la que se conecta. El input total de la unidad j es simplemente el sumatorio ponderado de los outputs separados de cada unidad conectada junto con un *bias* w_{j0} (ecuación 2.1).

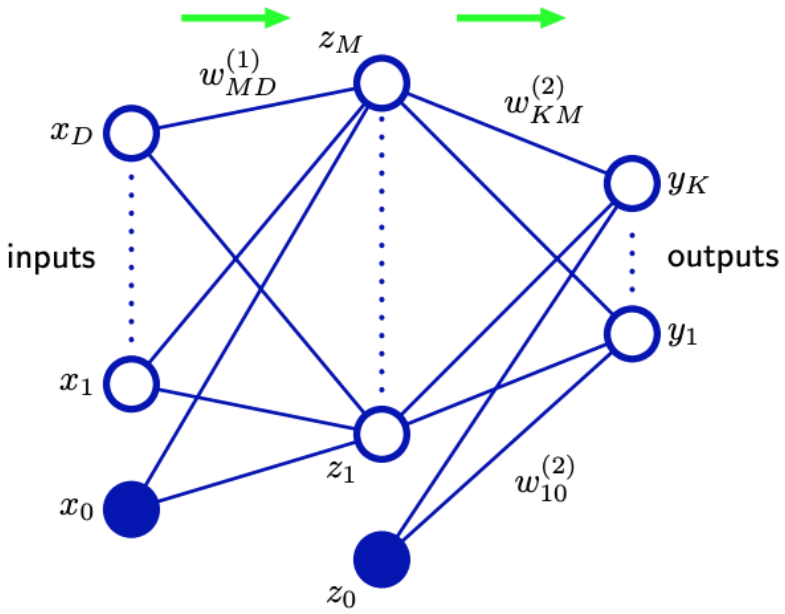


Figura 2.1: Componentes básicos de una red neuronal artificial. La regla de propagación empleada es el sumatorio estándar de pesos. Figura extraída de [4]

$$\mathrm{a_j} = \sum_{\mathrm{i=1}}^{\mathrm{D}} \mathrm{w_{ji}^{(1)} x_i} + \mathrm{w_{j0}^{(1)}} \tag{2.1}$$

donde d = i, ..., D es el número total de inputs, j = 1, ..., M la neurona dentro de cada capa y el superíndice (1) el número de capa.

Cada neurona de una capa se puede conectar a todas las neuronas de la capa anterior, caso denominado como capa totalmente conectada, o bien conectarse solo a un sub-conjunto de neuronas de la capa anterior, en cuyo caso la capa se refiere como localmente conectada. El primer caso presenta un entrenamiento más costoso debido a un mayor número de parámetros pero permite establecer el máximo número de relaciones posibles entre la entrada y la salida.

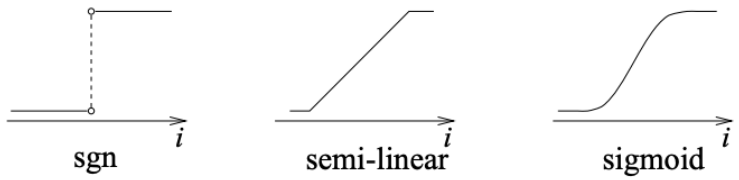


Figura 2.2: Diferentes tipos de función de activación para una unidad. Figura extraída de [4]

Es necesario, también, una función de activación h(·) que produzca el nuevo valor de activación de la unidad j a partir del input total $\mathrm{a_j}$:

$$\mathrm{z_j} = \mathrm{h(a_j)} \tag{2.2}$$

La función de activación puede ser una función no decreciente del input total de la unidad, aunque, generalmente, se utiliza una de las siguientes funciones límite definidas: una

función límite estricta (función "signo"), una función lineal o semi-lineal, o una función límite mas suavizada como la función sigmoide (figura 2.2). En algunas aplicaciones es común también el uso de la tangente hiperbólica, estando los valores de salida en el rango $[-1, +1]$.

Los distintos valores de z_j de cada unidad se combinan de nuevo linealmente para obtener el output activado de cada unidad:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.3}$$

donde $k = 1, ..., K$ es el número total de outputs.

Combinando las tres ecuaciones 2.1, 2.2 y 2.3 y agrupando los pesos y *bias* en el vector de parámetros ajustables $\mathbf{w}$, se obtiene la ecuación que calcula los outputs y_k:

$$y_k(\mathbf{x}, \mathbf{w}) = h\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{2.4}$$

2.2. Paradigmas de aprendizaje

Hay tres grandes paradigmas de dentro del aprendizaje automático:

- **Aprendizaje supervisado**, en el que se emplean ejemplos de entrenamiento consistentes en pares $(x, y), x \in X, y \in Y$ con el objetivo de encontrar una función $\mathcal{F} : X \to Y$ capaz de predecir el valor de salida y correspondiente a cualquier objeto de entrada x. Para ello, la función $\mathcal{F}$ tiene que generalizar a partir de los datos de entrenamiento a otros conjuntos de datos no presentados previamente. Normalmente se utiliza una función de coste que representa la discrepancia de nuestro modelo con los datos de entrenamiento, la cual se minimiza con algoritmos como el descenso por el gradiente. Tareas que incluye el paradigma de aprendizaje supervisado son el reconocimiento de patrones clasificación, regresión y tratamiento de datos secuenciales (por ejemplo, reconocimiento del habla, del lenguaje manuscrito o de gestos).
- **Aprendizaje no supervisado**, en el que se emplean ejemplos de entrenamiento consistentes en datos no anotados $x \in X$. Así, el aprendizaje no supervisado típicamente trata los objetos de entrada como un conjunto de variables aleatorias, siendo el objetivo estimar la distribución o algunas de sus propiedades para el conjunto de datos. Entre las aplicaciones de aprendizaje no supervisado se encuentran el agrupamiento, la estimación de distribuciones estadísticas o la compresión de datos.
- **Aprendizaje por refuerzo**, en el que los datos $x \in X$ no se tienen sino que se generan a partir de la interacción de un agente con el entorno. En cada instante temporal t, el agente realiza una acción y_t y el medio ambiente genera una observación x_t y un costo instantáneo c_t, de acuerdo con las dinámicas (por lo general, desconocidas) del agente y el entorno y sus parámetros. El objetivo es descubrir una política para la selección de las acciones que minimiza una cierta medida de un costo a largo plazo, por ejemplo, el coste acumulativo esperado. La dinámica del medio ambiente y el coste a largo plazo para cada política general son

desconocidos, pero pueden ser estimados. Tareas que caen dentro del paradigma de aprendizaje por refuerzo son problemas de control, juegos y otras tareas secuenciales.

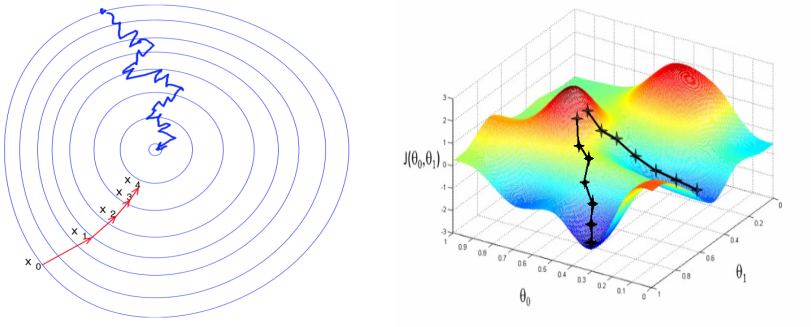
2.3. Entrenamiento

El entrenamiento de una red neuronal es un problema de optimización de la función de coste J(w) elegida:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} \text{Coste}(h_w(x_{(i)}), y_{(i)}) \tag{2.5}$$

donde $i = 1, ..., m$ es cada ejemplo de entrenamiento.

El método estándar para entrenar redes neuronales es el descenso de gradiente estocástico (*Stochastic Gradient Descent* o SGD). El descenso de gradiente clásico obtiene una nueva aproximación del vector que contiene los pesos mediante el gradiente completo para los datos de entrenamiento, lo que ralentiza enormemente el aprendizaje. La idea del SDG (Figura 2.3a) es usar solo un sub-conjunto, para calcular la nueva aproximación del vector de pesos, acelerando notablemente el cálculo ya que los conjuntos de datos utilizados para el entrenamiento suelen ser demasiado grandes para almacenarlos completamente en la RAM y/o realizar cálculos de manera eficiente. Además, la función optimizada generalmente no es convexa, por lo que el uso de diferentes partes de los datos en cada iteración puede causar que el modelo converja en un mínimo local.



(a) *GD* y *SGD* (b) Función de pérdida

Figura 2.3: Stochastic Gradient Descent: **a:** Ilustración de la convergencia de los algoritmos de *Gradient Descent* (rojo) y *Stochastic Gradient Descent* (azul). **b:** Función de coste y SGD con dos parámetros variables. Figuras extraidas de [5].

Otros métodos de resolución del problema de optimización, que dependiendo del caso pueden presentar mejores resultados que SGD son *Mini-Batch Gradient Descent* [6], *Momentum* [7] o el método de Adam [8], de gran uso y que consiste en una estimación de momento adaptativo.

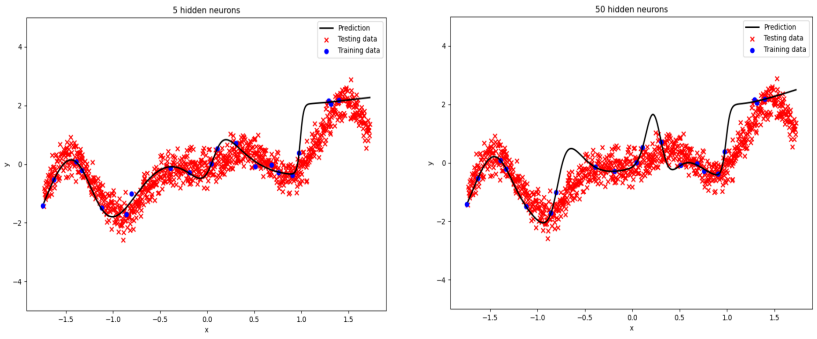
La forma de trabajo más habitual consiste en dividir el conjunto de muestras disponibles para entrenar en tres sets: set de entrenamiento, set de validación y set de testeo. También es necesario definir los hiperparámetros que definen el entrenamiento:

- **Tasa de aprendizaje**, que determina el tamaño del paso en cada iteración mientras se mueve hacia un mínimo de la función de coste;

- **Número de épocas**, que define el número de veces que el recorre el conjunto de muestras completo con el algoritmo;
- **Tamaño del batch**, corresponde con el número de muestras que recorrer la red;
- **Número de capas ocultas**;
- **Número de neuronas por capa**;
- **Función de activación**;

Cada subconjunto o batch de muestras elegidas del set de entrenamiento recorre la red hacia adelante, *forward propagation*, como se ha descrito con la ecuación 2.4. Una vez finalizado el lote, se observa la diferencia entre el input y el valor deseado y se introduce en la función de coste (Ecuación 2.5). Para saber cómo cambiar el valor de los pesos y los *bias*, se aplica el algoritmo de propagación hacia atrás, *backward propagation* que calcula las derivadas parciales de la función de coste respecto a cada uno de los parámetros modificables. Por lo tanto, en lugar de utilizar la salida deseada para entrenar las neuronas ocultas, se utiliza la derivada del error con respecto a sus actividades. La actividad de cada neurona oculta puede tener efectos en muchas neuronas de salida, por lo que se deben combinar. Una vez se tienen las derivadas del error para todas las unidades ocultas, se puede calcular las derivadas del error para sus pesos de entrada.

Las contribuciones de cada muestra del *batch* se suman y se actualizan los parámetros variables antes de empezar el nuevo *batch*. El proceso se repite para todos los *batch* definidos y se comprueba la precisión del modelo empleando las muestras del set de validación. El modelo puede quedar poco ajustado a los valores de entrenamiento, *underfitting* (figura 2.4a), y siendo incapaz de acertar nuevas predicciones; o por el contrario, demasiado ajustado, *overfitting* (figura 2.4b). En este caso el error en el set de entrenamiento será mínimo, pero el exceso de ajuste hará difícil que elabore predicciones acertadas.



(a) Underfitting (b) Overfitting

Figura 2.4: Ejemplo de *Underfitting* y *Overfitting* en una red entrenada con sólo 5 neuronas por capa y con 50 neuronas por capa.

El ajuste de los hiperparámetros permite encontrar un modelo que se ajuste tanto al set de entrenamiento como a nuevas muestras introducidas por el set de validación. Finalmente, para confirmar la precisión del modelo y elegir entre varios posibles conjuntos de hiperparámetros válidos, se utiliza el set de test y se obtiene la precisión final.

2.4. Redes neuronales convolucionales

La red neuronal convolucional, o CNN, es un tipo especializado de modelo de red neuronal diseñado para trabajar con imágenes bidimensionales, aunque se pueden usar con datos unidimensionales y tridimensionales.

El elemento clave de este tipo de redes es la capa convolucional. Esta capa realiza una operación llamada convolución, una operación lineal que consiste en la multiplicación de un conjunto de pesos con la entrada, al igual que una red neuronal tradicional. Dado que la técnica fue diseñada para entradas bidimensionales, la multiplicación se realiza entre una matriz de datos de entrada y una matriz bidimensional de pesos, denominada filtro o kernel.

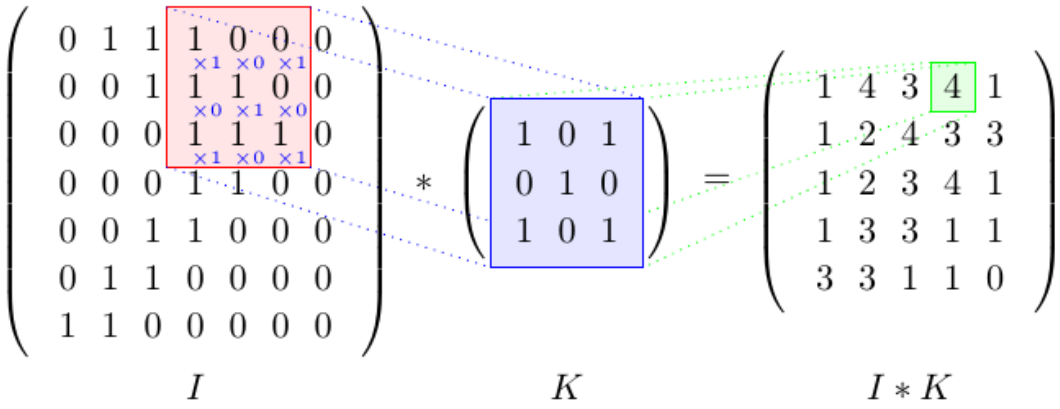


Figura 2.5: Ejemplo de filtro o kernel y de operación de convolución. El filtro es desplazado por toda la imagen para multiplicar todos los píxeles y obtener una nueva matriz que contiene las activaciones de dicho filtro.

El filtro es de menor tamaño que los datos de entrada y el tipo de multiplicación aplicada entre un parche del tamaño de filtro de la entrada y el filtro es un producto escalar. El uso de un filtro más pequeño que la entrada es intencional, ya que permite que el mismo filtro (conjunto de pesos) se multiplique por la matriz de entrada varias veces en diferentes puntos de la entrada. Específicamente, el filtro se aplica sistemáticamente a cada parte superpuesta o parche del tamaño del filtro de los datos de entrada, de izquierda a derecha y de arriba a abajo.

Si el filtro está diseñado para detectar un tipo específico de característica en la entrada, entonces la aplicación de ese filtro sistemáticamente en toda la imagen de entrada le da al filtro la oportunidad de descubrir esa característica en cualquier lugar de la imagen. Esta capacidad se conoce comúnmente como invariancia de traslación.

La innovación de usar la operación de convolución en una red neuronal es que los valores del filtro son pesos que deben aprenderse durante el entrenamiento de la red. La red aprenderá a partir de los datos qué patrones o características de la entrada son relevantes. La aplicación sucesiva de filtros (ver figura 2.6) reducirá el tamaño de las imágenes mediante su transformación a un espacio más compacto, pero sin perder las características o patrones que son críticos para obtener una buena predicción.

Junto a las capas convolucionales, se suelen utilizar otros dos tipos de capas: *pooling* y *flatten* (Figura 2.7). La primera, generalmente, se coloca después de una capa convolucional con la finalidad de reducir el número de píxeles para favorecer el tiempo de cálculo aunque

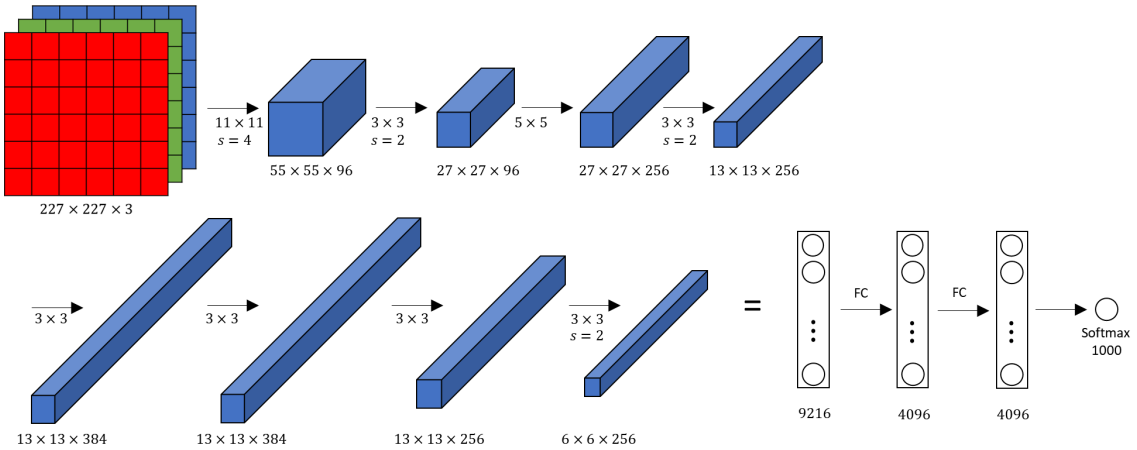


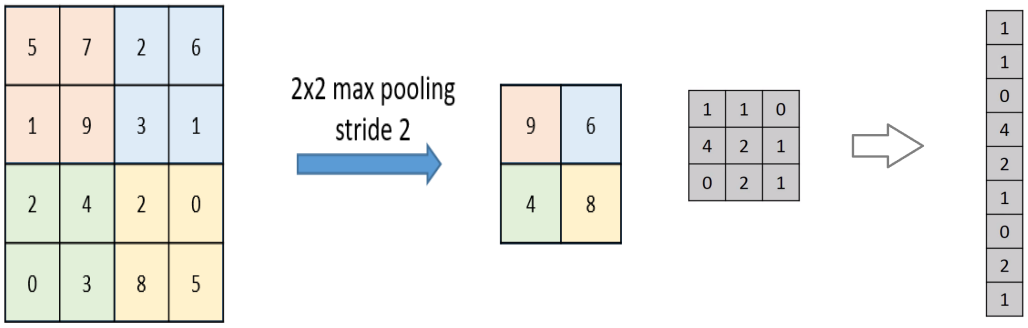
Figura 2.6: Ejemplo de red neuronal convolucional, AlexNet, en la que se aplican capas convolucionales y de max-pooling de manera secuencial, acabando con una *fully-connected layer*. Al estar orientada esta arquitectura a clasificación, la función de activación es la denominada *softmax*. Figura extraída de [9]

.

suponga una pérdida de información.

La operación que se suele utilizar en esta capa es *max-pooling* que divide a la imagen de entrada en un conjunto de rectángulos y, respecto de cada subregión, se queda con el máximo valor.

La segunda operación, *flatten*, tiene como objetivo transformar la matriz de salida de una capa convolucional en un vector unidimensional al final de la red. Esta última capa de salida tendrá tantas dimensiones como el número de clases que se debe predecir, o como el tamaño del descriptor que se desee aprender. La figura 2.6 muestra una de las arquitecturas más conocidas de red neuronal, en la que se puede observar la concatenación y distribución de estos tipos de capa.



(a) *Pooling* (b) *Flatten*

Figura 2.7: Ejemplo de **a**: *Pooling* y **b**: *Flatten*.

Por último, el *padding* consiste en añadir píxeles alrededor de la imagen para conseguir un tamaño deseado a la salida de la capa convolucional, y el *stride*, se refiere a un hiperparámetro que indica el número de píxeles que se desplaza el filtro al recorrer la imagen para aplicar la convolución. Ambos deben considerarse también en el diseño de la arquitectura de red.

2.5. Redes neuronales siamesas

Las redes siamesas [10] se componen de dos redes neuronales paralelas e idénticas y que comparten los valores de sus pesos. La utilidad en el contexto del problema de *image retrieval* es el envío paralelo de dos imágenes, una por cada red, para poder comparar los descriptores obtenidos en cada salida. En el caso de ser similares, se deduce que la distancia entre sus respectivos descriptores a la salida de la red debería ser pequeña. Y viceversa, en el caso de ser pares muy distintos, la distancia entre sus respectivos descriptores debería ser grande.

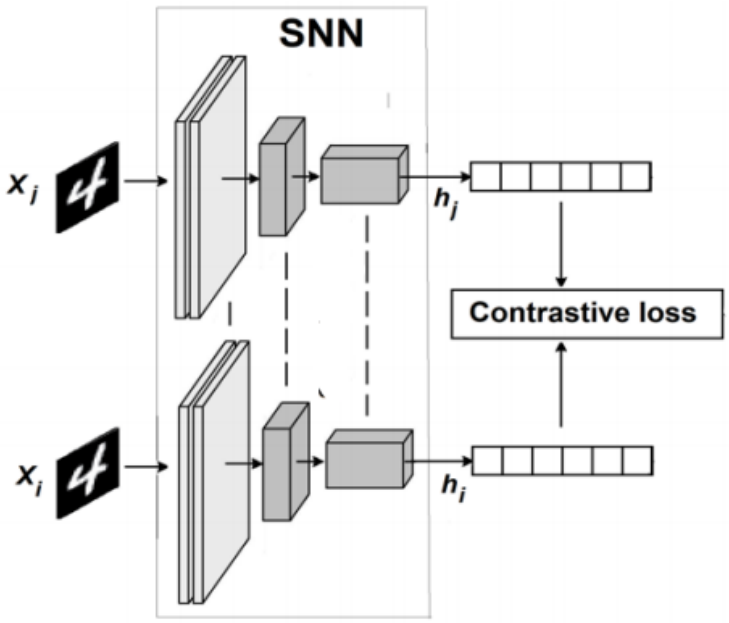


Figura 2.8: Ejemplo de redes neuronales siamesas: Ambas imágenes son introducidas en las redes idénticas y empleando la función *contrastive loss* se fuerza a la red a aprender representaciones cuya distribución está relacionada con el contenido de la imagen. Figura extraída de [10]

.

Para entrenar esta red se usa la función de coste conocida como *constrastive loss function* que está basada en la distancia entre valores, a diferencia de las funciones de coste basadas en errores de predicción, como la función de coste logística utilizadas en la clasificación. Como cualquier función de coste basada en la distancia, intenta asegurarse de que los ejemplos semánticamente similares estén cercanos en el espacio de los descriptores.

$$\mathcal{L}(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}, \mathrm{p}_\mathrm{ij}) = \mathrm{p}_\mathrm{ij} \left\|\mathbf{d}_\mathrm{i} - \mathbf{d}_\mathrm{j}\right\|_2^2 + (1 - \mathrm{p}_\mathrm{ij})\max(0, \tau - \left\|\mathbf{d}_\mathrm{i} - \mathbf{d}_\mathrm{j}\right\|_2^2) \tag{2.6}$$

Durante el entrenamiento se introduce un conjunto de muestras $\mathrm{S} = (\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}, \mathrm{p}_\mathrm{ij})$ compuesto por pares $\mathbf{x}_\mathrm{i}$ y $\mathbf{x}_\mathrm{j}$ con etiquetas binarias $\mathrm{p}_\mathrm{ij} \in \{0, 1\}$ asignadas a ellos de forma que si ambos vectores son semánticamente similares p_ij es 1 y si no son similares p_ij es 0. El conjunto se divide en un subconjunto de pares positivos con $\mathrm{p}_\mathrm{ij} = 1$ y pares negativos $\mathrm{p}_\mathrm{ij} = 0$.

La función de coste incrementa cuando la distancia entre descriptores $\left\|\mathbf{d}_\mathrm{i} - \mathbf{d}_\mathrm{j}\right\|_2^2$ es grande pero se trata de la misma pareja, $\mathrm{p}_\mathrm{ij} = 1$, o cuando para distinto par, $\mathrm{p}_\mathrm{ij} = 0$, la distancia es muy pequeña, gracias a la función *max* que compara con la tolerancia τ fijada. De esta forma, la red, al intentar minimizar la función de coste durante el entrenamiento, busca acercar la distancia entre descriptores.

3. Propuesta de métodos de binarización

El objetivo es encontrar imágenes, similares a la proporcionada por el usuario, dentro una base de datos de tamaño considerable. Una de las técnicas más recientes consiste en binarizar la información extraída de cada imagen para mejorar la eficiencia en la recuperación de imágenes. Con este estudio se pretende estudiar e implementar una serie de técnicas previas a la binarización para mejorar la precisión final del algoritmo.

Antes de entrar en detalle en cada técnica estudiada, es necesario explicar como se realiza el proceso de binarización. En concreto en este trabajo, se utiliza la binarización por comparación.

Para realizar la comparación se necesita tener la información de cada imagen almacenada en descriptores reales. En el caso de la red siamesa se extrae de la última capa pero en redes que terminan con un clasificador se necesita extraer de la penúltima capa. En otras palabras, se debe extraer de la red neuronal un vector de números reales con tantas dimensiones como neuronas tengan las capas de donde se extrae.

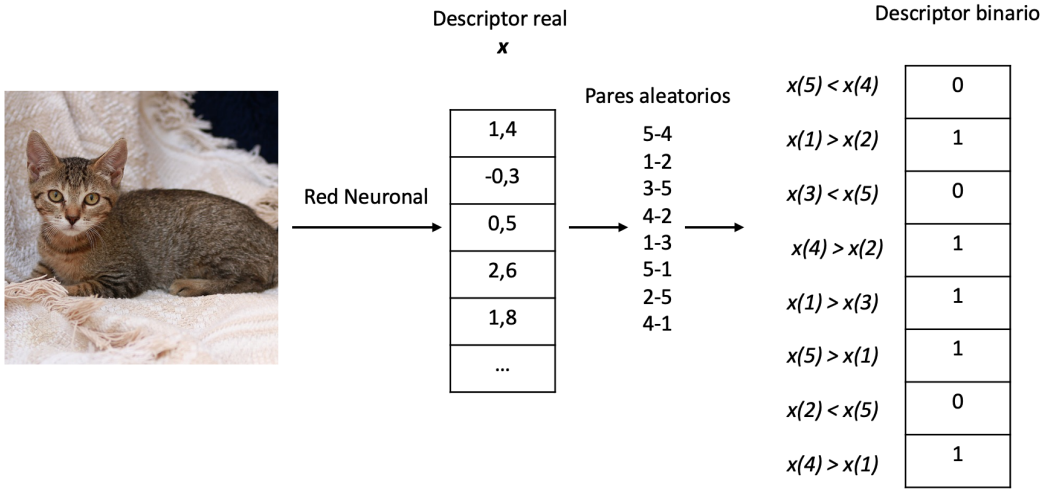


Figura 3.1: Binarización por comparación: ejemplo de transformación de un descriptor continuo en uno discreto de 8 BITS.

La función de binarización selecciona pares aleatorios de las dimensiones del vector con la condición de no repetir pares ni elegir el mismo par, pero en orden inverso. Dentro de cada pareja se realiza una comparación de forma que si el primer componente es menor que el segundo, se obtiene un cero y viceversa, si el primer componente es mayor que el segundo, se obtiene un cero; tal y como se muestra en la ecuación 3.1 donde $\mathbf{x}$ es el descriptor real.

$$\mathrm{f}(\mathbf{d}, \mathrm{a}, \mathrm{b}) = \begin{cases} 0 & \text{if } \mathbf{d}(\mathrm{a}) \leq \mathbf{d}(\mathrm{b}) \\ 1 & \text{if } \mathbf{d}(\mathrm{a}) \geq \mathbf{d}(\mathrm{b}) \end{cases} \tag{3.1}$$

La figura 3.1 representa de forma gráfica el proceso completo. Dos de las técnicas elegidas para mejorar la precisión final del algoritmo están orientadas a la reducción de dimensiones del descriptor real manteniendo aquellas más representativas. A continuación se enumeran todos los métodos estudiados:

- Preprocesamiento por análisis de componentes principales (PCA) para reducirla a las dimensiones más características.
- Preprocesamiento por blanqueamiento de imágenes.
- Preprocesamiento por blanqueamiento de imágenes por análisis de componentes de fase cero (ZCA).
- Análisis discriminante lineal (LDA) para la reducción y reordenación de las dimensiones del descriptor real en comparación con el análisis de componentes principales (PCA) [11].
- Cambio de una selección aleatoria de componentes en la binarización, a una selección exponencial con mayor probabilidad a las primeras dimensiones obtenidas al aplicar PCA.

3.1. Preprocesamiento por PCA

Una de las utilidades del análisis de componentes principales (PCA) es reducir la dimensionalidad de un conjunto de datos que consta de muchas variables correlacionadas entre sí [12], ya sea en gran medida o levemente, mientras se retiene un determinado porcentaje de variabilidad en el conjunto de datos. El proceso consiste en transformar las variables a un nuevo sistema de coordenadas, las cuales se conocen como componentes principales y son ortogonales, ordenadas de mayor a menor varianza. De esta manera, el primer componente principal retiene la variación máxima que estaba presente en los componentes originales. Los componentes principales son los vectores propios de la matriz de covarianza y, por tanto, son ortogonales. PCA reduce eficazmente la dimensión de los datos de la imagen mientras mantiene las propiedades principales de la imagen original [13].

Para ilustrar el análisis de componentes principales se va a utilizar un conjunto de datos $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ con $n = 2$ dimensiones. Supongamos que queremos reducir las dimensiones de 2 a 1 (en la aplicación concreta de este trabajo fin de máster se reducirá de 784 a 154, por ejemplo). Primero se debe preprocesar el conjunto para que las x_1 y x_2 tengan media cero y una varianza similar. PCA encontrará un sub-espacio, cuya dimensión sea inferior a la de los datos de entrada, en el que proyectar el conjunto de datos tal y como se muestra en la figura 3.2.

Formalmente, la matriz de covarianza se calcula de la siguiente forma:

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^{T} \tag{3.2}$$

u_1, dirección principal de variación de los datos, es el mayor y principal vector propio de la matriz de covarianza Σ, y u_2, el segundo vector propio. El conjunto de vectores propios se almacenan en U, matriz ortogonal, con sus correspondientes valores propios, $\lambda_1, \lambda_2, ..., \lambda_n$. $u_1^T x$ es la longitud de la proyección de x en el vector u_1 y similarmente, $u_2^T x$ es la longitud de la proyección de x en el vector u_2. A esta rotación se le denomina x_{rot}.

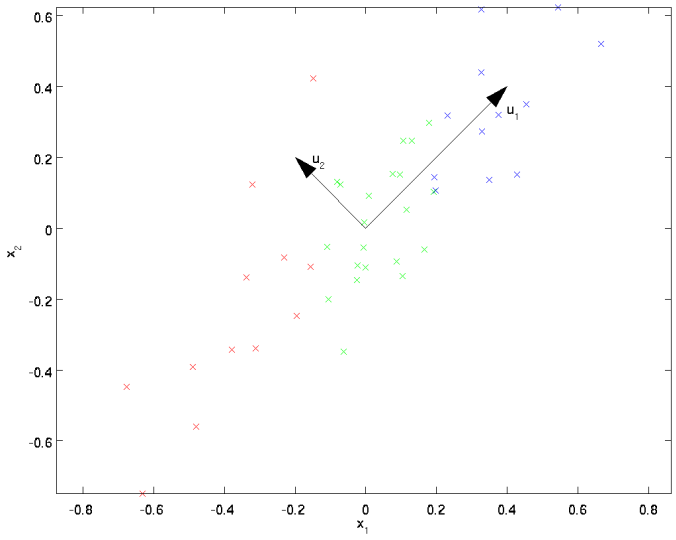


Figura 3.2: Componentes principales de un conjunto de características bidimensionales. u_1 es la dirección principal de variación de las los datos y u_2, la segunda dirección de variación.

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & ... & u_n \\ | & | & & | \end{bmatrix} \tag{3.3}$$

$$x_{rot} = U^T x = \begin{bmatrix} u_1^T x \\ u_2^T x \end{bmatrix} \tag{3.4}$$

Para reducir dimensiones, se seleccionan los k primeros componentes de x_{rot} que corresponden a las k direcciones principales, en este caso $k = 1$. A este conjunto se le suele denominar la forma compacta o comprimida. A partir de aquí se plantean dos alternativas. La primera, recuperar una aproximación de los datos iniciales manteniendo solo los componentes principales y entrenar la red con las reconstrucciones y la segunda, entrenar la red directamente con la imagen reducida por PCA.

3.1.1. Entrenamiento de la red con imágenes reconstruidas de la reducción por PCA

La idea intuitiva tras esta propuesta consiste en la reducción de las dimensiones de los datos de entrada (pero preservando la máxima cantidad de variabilidad), con el objetivo de entrenar modelos más pequeños y eficientes (el tamaño de las redes neuronales dependen de los datos de entrada). La aproximación se calcularía del siguiente modo:

$$x' = U \begin{bmatrix} x_1' \\ \vdots \\ x_k' \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sum_{i=1}^{k} u_i x_i' \tag{3.5}$$

Para decidir el número de componentes seleccionados, k, normalmente se observa el porcentaje de varianza retenido para diferentes valores de k

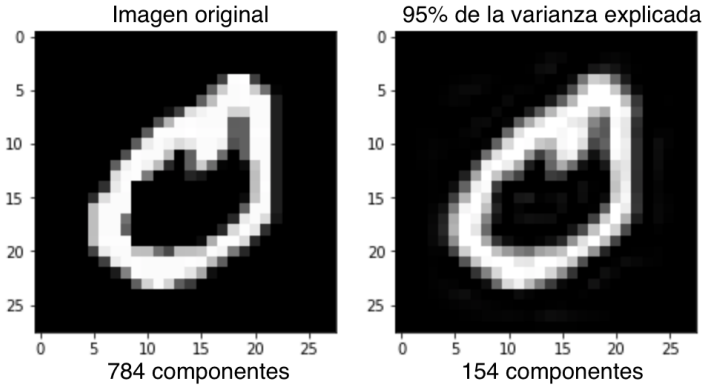


Figura 3.3: Ejemplo de imagen reconstruida en sus dimensiones principales: con 154 componentes de 784 se logra explicar un 95 % de la varianza.

Si k = n, entonces tenemos una aproximación exacta a los datos y decimos que se retiene el 100 % de la varianza. Es decir, se conserva toda la variación de los datos originales. Por el contrario, si k = 0, entonces estamos aproximando todos los datos con el vector cero y, por lo tanto, se retiene el 0 % de la varianza.

Alimentar la red con una imagen x' en vez de x reduce el tiempo de procesamiento y recientes publicaciones han comprobado que se obtiene una aproximación muy buena de la imagen original.

Para ilustrar la eficacia de esta reconstrucción se ha calculado y representado gráficamente en la figura 3.4 la cantidad de varianza explicada media según el número de dimensiones de la imagen reconstruida para la base de datos *MNIST handwritten digits*. La figura 3.3 es un ejemplo de una imagen de esta base de datos reconstruida en sus dimensiones principales.

3.1.2. Entrenamiento de la red con imágenes reducidas por PCA

La segunda opción consiste en no reconstruir la imagen y realizar el entrenamiento con los componentes principales tal y como se han obtenido al aplicar PCA. Al estar ordenados de mayor a menor por la varianza retenida, no se tiene una imagen clara sino un conjunto de pixeles ordenados según PCA. Por este motivo, aplicar una red convolucional no tendría sentido. En este caso se necesita una red con capas completamente conectadas y que vayan reduciendo el número de neuronas hasta quedar tantas como dimensiones queramos que tenga el descriptor continuo.

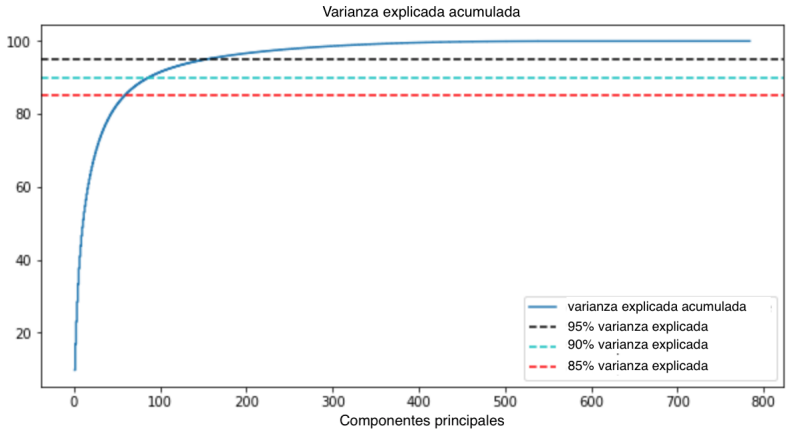


Figura 3.4: Varianza explicada acumulada en función del número de dimensiones para la base de datos *MNIST handwritten digits*: Con apenas 150 dimensiones de las 784 totales es posible explicar un 95 % de la varianza.

3.2. Preprocesamiento por blanqueamiento

En el apartado anterior se ha utilizado PCA para reducir la dimensión de los datos de entrada. Existe un paso de preprocesamiento estrechamente relacionado llamado blanqueamiento [14] o, en otras publicaciones, formación de esferas, que es necesario para algunos algoritmos. Entrenando imágenes, la entrada sin procesar es redundante, ya que los valores de los píxeles adyacentes están altamente correlacionados. El objetivo del blanqueamiento es conseguir que la entrada sea menos redundante. Más formalmente, se pretende que los algoritmos de aprendizaje vean una entrada de entrenamiento donde las características estén menos correlacionadas entre sí, y todas las dimensiones tengan la misma varianza.

Para hacer que las características no estén correlacionadas tenemos que seguir el razonamiento anterior y obtener $x_{rot}^{(i)}$. Siguiendo el ejemplo anterior, esta matriz quedaría como se muestra en la figura 3.5.

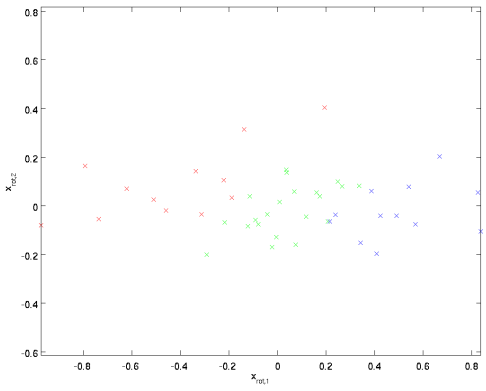


Figura 3.5: Conjunto de datos rotados y proyectados en las dos direcciones principales, $x_{rot}^{(i)}$.

Calculando la matriz de covarianza, se puede observar que los valores de la diagonal son λ_1 y λ_2, y que los valores de fuera de la diagonal son cero, por lo tanto, se puede confirmar que $x_{rot,1}$ y $x_{rot,2}$ están no-correlacionados.

$$\Sigma = \begin{bmatrix} 7,29 & 0 \\ 0 & 0,69 \end{bmatrix} \tag{3.6}$$

El último paso sería hacer que los distintos componentes de $x_{rot}^{(i)}$ tengan varianza unitaria. Esto se consigue reescalando por $1/\sqrt{\lambda_i}$:

$$x_{white,i} = \frac{x_{rot,i}}{\sqrt{\lambda_i}} \tag{3.7}$$

Si además se desea combinar este blanqueamiento con una reducción de dimensiones, tan solo es necesario incluir los k componentes deseados y operar finalmente.

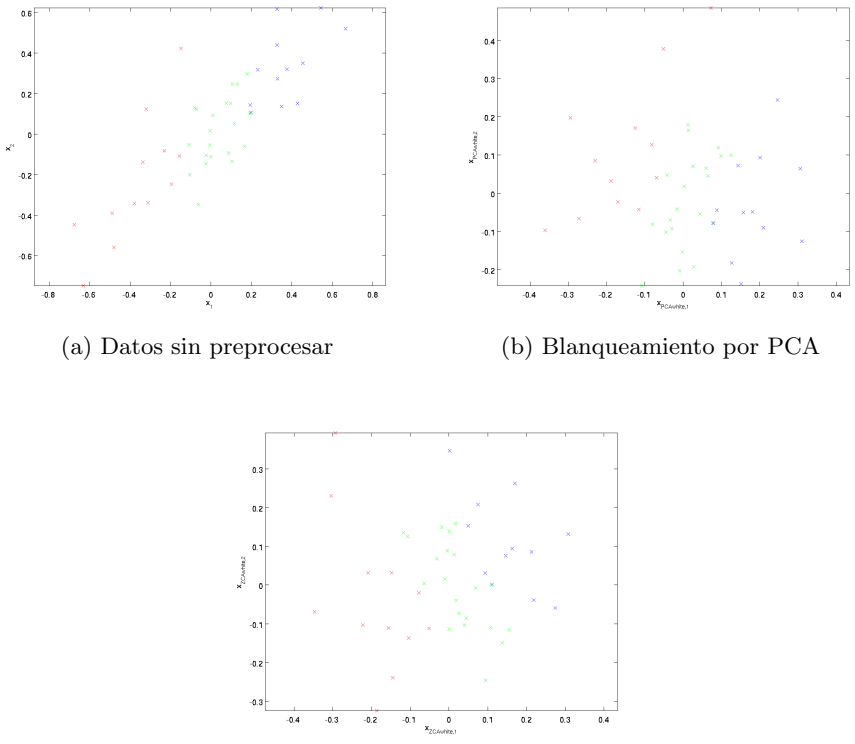
3.3. Preprocesamiento por blanqueamiento ZCA

La transformación explicada en el apartado anterior no es la única posibilidad. De hecho, los datos blanqueados se mantendrán blanqueados después de cualquier rotación R, es decir, $x_{white}^{*} = Rx_{white}$. La técnica denominada blanqueamiento por ZCA[15] (análisis de componentes de fase cero), escoge como matriz de rotación a la propia matriz de vectores propios de la matriz de covarianza U, obteniendo:

$$x_{ZCAwhite} = Ux_{white} \tag{3.8}$$

Una de las propiedades de la transformación ZCA, denominada también transformación de Mahalanobis es que los datos blanqueados se aproximan lo máximo posible a los datos originales. Esto se debe precisamente a que ZCA intenta transformar los datos lo menos posible, por lo que es mejor que cada fila esté cerca de una de las funciones base originales (que serían imágenes con un solo píxel activo). Esto es posible porque las correlaciones en imágenes naturales son en su mayoría muy locales (por lo que los filtros de decorrelación también pueden ser locales).

Al estar transformadas muy ligeramente, las imágenes blanqueadas con ZCA todavía se parecen a las imágenes normales. Esto es importante para algoritmos como las redes neuronales convolucionales, cuyos filtros explotan los patrones entre píxeles contiguos y dependen en gran medida de las propiedades locales de las imágenes naturales. Para la mayoría de los otros algoritmos de aprendizaje automático, es de menor importancia si los datos se blanquean con ZCA o se incluyen otros métodos, como el propio PCA [16].



(a) Datos sin preprocesar

(b) Blanqueamiento por PCA

(c) Blanqueamiento por ZCA

Figura 3.6: Comparación entre PCA y ZCA: **a:** Datos sin preprocesar. **b:** Blanqueamiento por PCA. **c:** Blanqueamiento por ZCA.

Una última consideración a tener en cuenta durante el reescalado es que algunos valores propios λ_i pueden ser cercanos a cero de modo que al dividir entre $\sqrt{\lambda_i}$, los valores desbordarían. En la práctica se emplea un factor de regularización, quedando la ecuación finalmente:

$$x_{white,i} = \frac{x_{rot,i}}{\sqrt{\lambda_i + \epsilon}} \tag{3.9}$$

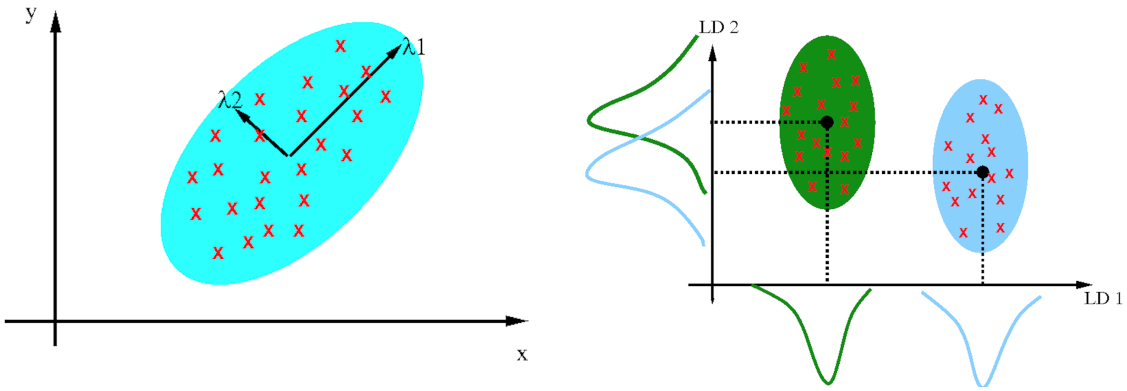
Cuando x tiene valores comprendidos entre $[-1, 1]$, el valor típico empleado es $\epsilon \approx 10^{-5}$. Para el caso de imágenes, la regularización tiene también el efecto de suavizar ligeramente (o filtrado de paso bajo) la imagen de entrada. Esto también tiene el efecto deseable de eliminar los artefactos de alias causados por la forma en que se colocan los píxeles en una imagen y puede mejorar las funciones aprendidas.

3.4. Análisis discriminante lineal del descriptor real

El análisis discriminante lineal (LDA)[17] es una generalización del discriminante lineal de Fisher, un método utilizado en estadística, reconocimiento de patrones y aprendizaje automático para encontrar una combinación lineal de características que caracterizan o separan dos o más clases de objetos. Este método proyecta un conjunto de datos en un espacio de menor dimensión con buena separabilidad de clases para evitar el sobreajuste ("maldición de la dimensionalidad") y reducir los costos computacionales. La combinación

resultante puede usarse como clasificador lineal o, más comúnmente, para la reducción de dimensionalidad antes de la clasificación posterior.

Tanto el análisis discriminante lineal (LDA) como el análisis de componentes principales (PCA) son técnicas de transformación lineal que se utilizan comúnmente para la reducción de dimensionalidad (ambas son técnicas para la factorización matricial de datos). La diferencia más importante entre ambas técnicas es que PCA se puede describir como un algoritmo no supervisado, ya que "ignora" las etiquetas de clase y su objetivo es encontrar las direcciones (los denominados componentes principales) que maximizan la varianza en un conjunto de datos. LDA, sin embargo, es un algoritmo supervisado que calcula las direcciones ("discriminantes lineales") que representan los ejes que maximizan la separación entre múltiples clases.



(a) PCA (b) LDA

Figura 3.7: Comparación entre PCA y LDA. Como se muestra en el eje x (componente nuevo LD 1 en la dimensionalidad reducida) y el eje y (componente nuevo LD 2 en la dimensionalidad reducida), LDA separaría bien las dos clases distribuidas normalmente. Figuras extraídas de [18]

La siguiente pregunta es: ¿Qué es un subespacio de características adecuadas que maximiza los ejes de los componentes para la separación de clases?

Para responder a esta pregunta, supongamos que nuestro objetivo es reducir las dimensiones de un conjunto de datos de dimensión d proyectándolo en un subespacio de k dimensiones, donde $k < d$.

Se calculan los vectores propios de nuestro conjunto de datos y se recopilan en las llamadas matrices de dispersión, es decir, la matriz de dispersión entre clases y la matriz de dispersión dentro de la clase. Cada uno de estos autovectores está asociado con un autovalor que nos informa acerca de la "magnitud" de los autovectores.

Si observamos que todos los valores propios tienen una magnitud similar, entonces esto puede ser un buen indicador de que nuestros datos ya están proyectados en un espacio de características adecuadas.

Y en el otro escenario, si algunos de los valores propios son mucho más grandes que otros, podríamos estar interesados en mantener solo los vectores propios con los valores propios más altos, ya que contienen más información sobre nuestra distribución de datos. Por otro lado, si los valores propios cercanos a 0 son menos informativos, podríamos considerar descartarlos para construir el nuevo subespacio de características, de manera similar a como hacíamos con PCA.

LDA se aplica en cinco pasos básicos:

- Calcular la media de los vectores correspondientes a cada clase $\mathbf{m_i} = \mu_{\omega i}$;
- Calcular la matriz de dispersión de cada clase, S_W, y entre clases, S_B, para obtener los discriminantes $S_W^{-1}S_B$:

$$S_w = \sum_{i=1}^{c} S_i, \tag{3.10}$$

$$S_B = \sum_{i=1}^{c} N_i(\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{3.11}$$

donde

$$S_i = \sum_{\mathbf{x}\in D_i}^{n} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T \tag{3.12}$$

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x}\in D_i}^{n} \mathbf{x}_k \tag{3.13}$$

- Calcular los vectores propios $(e_1, e_2, e_3...e_d)$ y los valores propios correspondientes $(\lambda_1, \lambda_2, \lambda_3...\lambda_d)$ de las matrices de dispersión;
- Ordenar los vectores propios en orden decreciente según el valor de su valor propio correspondiente y elegir los k valores propios más grandes para formar la matriz W_{dxk} donde cada columna representa un vector propio;
- Usar esta matriz de vectores propios W para transformar las muestras a un nuevo subespacio aplicando la ecuación 3.14, donde X es una matriz n x (d − dimensiones) representando las n muestras e Y son las muestras en el nuevo subespacio:

$$\mathbf{Y} = \mathbf{X}x\mathbf{W} \tag{3.14}$$

Este método presenta una limitación y es que al resolver el problema de generalizado de valores propios $S_W^{-1}S_B$, el máximo de discriminantes que vamos a obtener es igual al número de clases menos 1 ya que S_B está formado por la matriz de dispersión entre clases. Sin embargo, en la mayoría de aplicaciones esta reducción es suficiente.

3.5. Emparejamiento exponencial

Como se ha explicado al principio de este capítulo, la función de binarización escoge pares aleatorios del descriptor continuo para realizar la comparación. Sin embargo, PCA ordena los pares de mayor a menor varianza por lo que es posible que si la selección aleatoria escoge un gran número de pares que contengan las últimas dimensiones del descriptor, no se esté realizando la comparación con la información más significativa de la imagen.

La última modificación ha consistido en cambiar esa selección aleatoria a una selección que siga una distribución exponencial, dando mayor probabilidad de ser escogidas las primeras dimensiones, y se ha hecho una comparación entre ambas posibilidades.

4. Bases de datos y Métricas

4.1. Métrica de evaluación: *mean average precisión*

La precisión media *mAP* es una métrica popular que se utiliza para medir el desempeño de los modelos que realizan tareas de recuperación de documentos, información y detección de objetos. Se define como:

$$\text{mAP} = \frac{\sum_{\text{q=1}}^{\text{Q}} \text{AveP(q)}}{\text{Q}} \tag{4.1}$$

donde Q es el número de consultas en el set de datos y AveP es la media de precisión para una consulta dada q. Para entender esta métrica es necesario explicar primero los conceptos de precisión y recall, de uso común para juzgar el desempeño de un modelo de clasificación y que se van a explicar con un ejemplo sencillo de recuperación de documentos. La precisión se define como la proporción de los documentos extraídos que son relevantes para la consulta del usuario sobre los documentos extraídos.

$$\text{Precisión} = \frac{|\{\text{documentos relevantes}\} \cap \{\text{documentos extraídos}\}|}{\{\text{documentos extraídos}\}} \tag{4.2}$$

Por otro lado, recall es definido como la proporción de documentos extraídos que son relevantes para la consulta del usuario sobre los documentos relevantes.

$$\text{Recall} = \frac{|\{\text{documentos relevantes}\} \cap \{\text{documentos extraídos}\}|}{\{\text{documentos relevantes}\}} \tag{4.3}$$

En el contexto de *image retrieval*, el usuario introduce una imagen, *query image*, y obtiene un conjunto de imágenes similares de la base de datos.

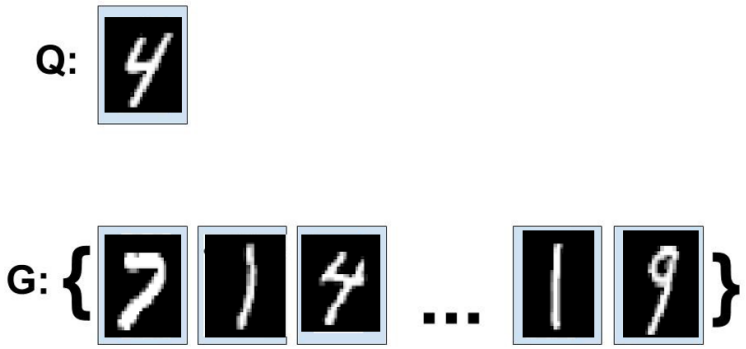


Figura 4.1: *Query image* y conjunto extraído. Q es la imagen consultada y G un conjunto etiquetado de la base de datos.

Para cada elemento de Q se calcula la distancia de Hamming, que mide la similitud de la imagen respecto a la imagen de consulta, y se ordena toda la secuencia de menor a mayor distancia, G′. Sin embargo, la secuencia puede contener imágenes que no se correspondan con la consulta deseada.

Figura 4.2: Similitud entre *Query image* y conjunto extraído: El conjunto G′ representa si las imágenes son similares a la consulta o no.

Esta combinación de precisión y ordenamiento, según la distancia de cada imagen con la consultada, permite trazar la curva que precision-recall, p(r). El área bajo esta curva es el *average precision*:

$$\mathrm{AP} = \int_0^1 \mathrm{p(r)dr} \tag{4.4}$$

Siguiendo con la nomenclatura anterior, otra fórmula equivalente sería:

$$\mathrm{AP} = \frac{\sum_{\mathrm{k=1}}^{\mathrm{n}} (\mathrm{Precision(k)\ x\ Relevance(k)})}{\mathrm{N\ imágenes\ relevantes}} \tag{4.5}$$

donde la función *Relevance* es un indicador que equivale a 1 si la imagen en el puesto k es relevante y 0, en el caso contrario.

Figura 4.3: Cálculo de *average precision*

Continuando con el ejemplo, una vez se tiene la secuencia ordenada de aciertos y fallos, es posible calcular el *average precision* con la ecuación 4.5. En este caso quedaría:

$$\mathrm{AP} = \frac{1}{3}\left(\frac{1}{1} + \frac{0}{2} + \frac{0}{3} + \frac{2}{4} + \frac{3}{5} + \frac{0}{6}\right) = 0,7 \tag{4.6}$$

Average Precision penaliza modelos que no son capaces de ordenar G′ con las imágenes correctas en las primeras posiciones. Teniendo en cuenta que son 3 las imágenes correctas, si en el momento de ordenarlas, hubieran quedado en las tres primeras posiciones, el AP sería de 1:

$$\mathrm{AP} = \frac{1}{3}\left(\frac{1}{1} + \frac{2}{2} + \frac{3}{3} + \frac{0}{4} + \frac{0}{5} + \frac{0}{6}\right) = 1 \tag{4.7}$$

En concreto en este trabajo, siguiendo otras publicaciones similares, se calculan secuencias con las 1.000 imágenes con menores distancias, AP@1000. Para un conjunto de imágenes consultadas, se calcula cada AP particular y se realiza la media, obteniendo finalmente la *mean average precision*:

$$\mathrm{mAP} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{AP_i} \tag{4.8}$$

4.2. MNIST handwritten digits

La base de datos *MNIST handwritten digits* está compuesta por 60.000 imágenes para entrenamiento y 10.000 para la evaluación. Cada una tiene una resolución de 28 x 28 píxeles en escala de grises y contiene un dígito manuscrito correspondiente a una de las diez clases, los dígitos del 0 al 9. Su frecuente uso en redes neuronales ha permitido alcanzar modelos de enorme precisión en el entrenamiento de la red.

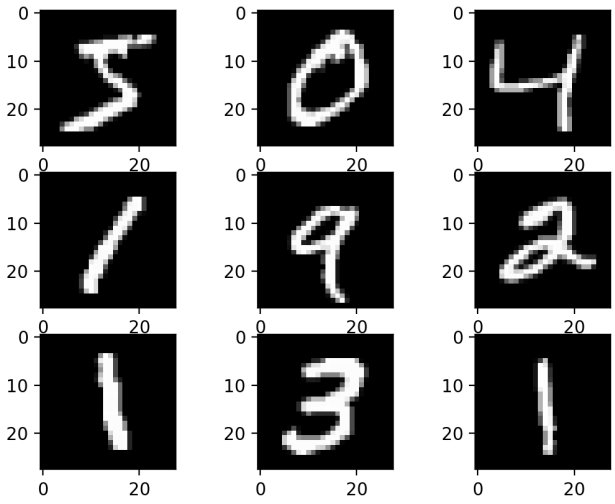


Figura 4.4: Ejemplos de dígitos de la base de datos *MNIST handwritten digits*.

En este caso, el entrenamiento se ha realizado con una red neuronal convolucional siamesa como la explicada en el capítulo 2, en la que se generan pares de imágenes tanto positivos, imágenes distintas pero del mismo dígito, como negativos, distinta imagen y dígito. De la red ya entrenada se obtiene un descriptor por cada imagen, cuyas dimensiones serán reducidas empleando una de las técnicas de reducción, para posteriormente ser binarizado. La evaluación del método en esta base de datos se ha realizado, de acuerdo a artículos con rigurosidad científica, aplicando la métrica *mean average precision* explicada en la sección anterior. Para esta base de datos, el conjunto de consulta está formado por 1000 imágenes con 100 de cada clase y el conjunto de extracción lo forman las 69.000 restantes.

4.3. CIFAR-10

La base de datos *CIFAR-10* está compuesta por 50.000 imágenes para entrenamiento y 10.000 para la evaluación. Cada una tiene una resolución de 32 x 32 píxeles a color y contiene una de las 10 posibles clases: camión, barco, rana, caballo, coche, pájaro, avión, gato, perro o ciervo.

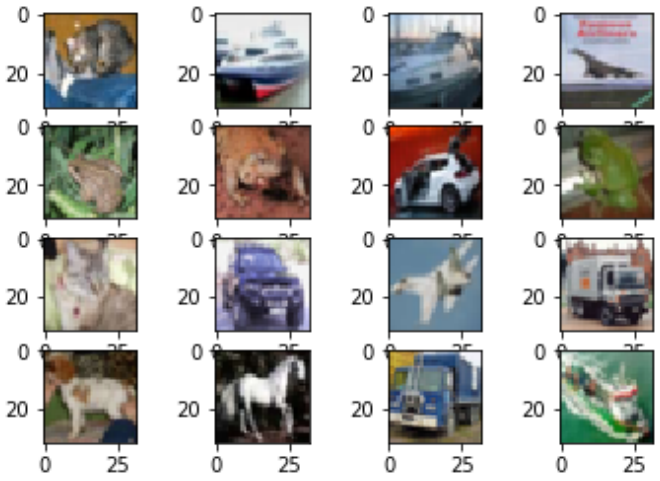


Figura 4.5: Ejemplos de imágenes de la base de datos *CIFAR-10*.

Para el entrenamiento de la red se ha empleado una red neuronal *Resnet 50*. Una tipología de red diseñada para aliviar el entrenamiento de redes con un elevado número de capas mediante lo que se conoce como conexiones de salto entre bloques residuales. Estas resuelven el problema de la desaparición del gradiente en las redes neuronales profundas al permitir esta ruta de acceso directo alternativo para que avance el gradiente. Las características de la distribución de las capas y los bloques residuales se detallan en la tabla 4.1 y la figura 4.6 muestra un detalle de las dos primeras etapas.

Etapa	Bloques residuales	Capas por bloque	Kernel en cada capa
Convolución inicial	-	2	7x7, 3x3
Etapa 1	3	3	64x64, 64x64, 256x256
Etapa 2	4	3	128x128, 128x128, 512x512
Etapa 3	6	3	256x256, 256x256, 1024x1024
Etapa 4	3	3	512x512, 512x512, 2048x2048
Maxpool	Transforma la salida en vector unidireccional		
Clasificación	Softmax para las 10 clases		

Tabla 4.1: Etapas, bloques residuales, capas y kernels empleados en la *Resnet 50*.

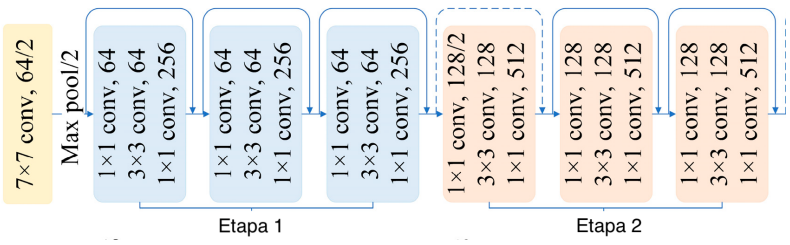


Figura 4.6: Detalle de la capa inicial de convolución y las dos primeras etapas de bloques residuales de la *Resnet 50*.

El descriptor se ha extraído de la penúltima capa con una dimensión de 32.768 y, de nuevo, se ha aplicado la métrica *mean average precision* con las 1.000 primeras imágenes obtenidas.

5. Resultados

5.1. MNIST handwritten digits

Las técnicas se han implementado en Python utilizando las librerías Keras y Tensorflow y la estructura explicada en el capítulo anterior.

	Año	Conferencia	16 bits	32 bits	64 bits
[19]	2016	CVPR	28,18	32,02	44,53
[20]	2017	-	43,15	46,58	49,88
[21]	2018	CVPR	94,31	95,48	96,36
[11]	2020	-	99,16	99,20	99,08
PCA (entrada) + Reconstrucción*	2021	-	74,48	78,35	78,00
PCA (entrada) + Sin reconstrucción*	2021	-	78,65	80,25	79,12
Blanqueamiento	2021	-	86,34	83,23	88,38
Blanqueamiento por ZCA	2021	-	86,12	89,33	89,28
Reducción LDA	2021	-	**99,31**	**99,56**	-
Emparejamiento exponencial	2021	-	99,28	99,61	99,72

Tabla 5.1: mAP para distintos algoritmos de binarización en la base de datos *MNIST handwritten digits*. Se ha utilizado mAP@1000 y se han obtenido resultados para 3 tamaños distintos de descriptor. *Para el preprocesamiento PCA se ha elegido una varianza representada del 95 % como valor respresentativo, el resto de valores constan en la figura 5.1.

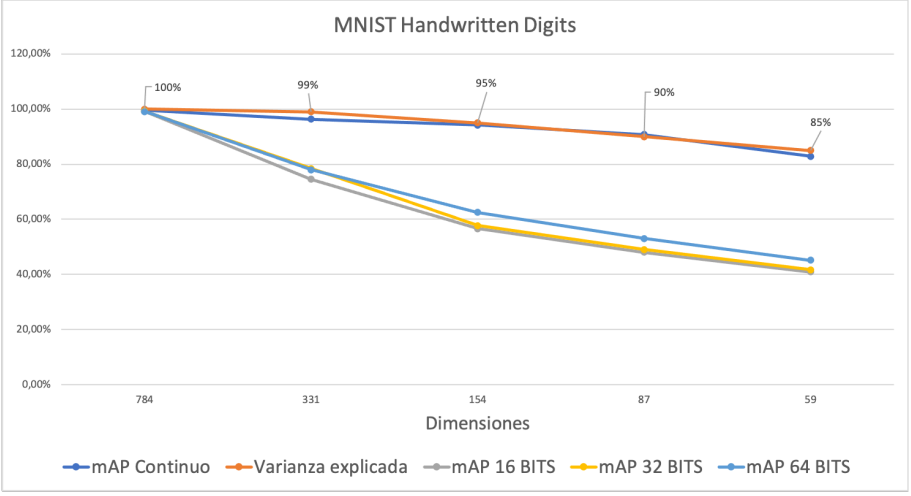


Figura 5.1: Varianza explicada acumulada, mAP del descriptor continuo y del discreto para distintos tamaños de descriptores en función del número de dimensiones para la base de datos *MNIST handwritten digits*: Con apenas 150 dimensiones de las 784 totales es posible explicar un 95 % de la varianza.

5.2. CIFAR-10

Las técnicas se han implementado en Python utilizando la librería Pytorch y la estructura explicada en el capítulo anterior.

	Año	Conferencia	16 bits	32 bits	64 bits
[22]	2015	CVPR	19,43	24,86	27,73
[23]	2017	-	21,53	26,50	31,85
[24]	2019	CVPR	28,44	28,53	28,67
[21]	2018	CVPR	29,94	31,47	32,53
[25]	2020	CVPR	53,2	57,3	57,8
[11]	2020	-	85,75	**90,93**	87,69
PCA (entrada) + Reconstrucción*	2021	-	53,74	55,95	58,69
PCA (entrada) + Sin reconstrucción*	2021	-	69,93	67,38	72,39
Blanqueamiento	2021	-	82,47	81,29	84,10
Blanqueamiento por ZCA	2021	-	83,57	89,28	89,73
Reducción LDA	2021	-	85,17	92,14	-
Emparejamiento exponencial	2021	-	**88,36**	89,92	**90,13**

Tabla 5.2: mAP para distintos algoritmos de binarización en la base de datos *CIFAR-10*: Se ha utilizado mAP@1000 y se han obtenido resultados para 3 tamaños distintos de descriptor. *Para el preprocesamiento PCA se ha elegido una varianza representada del 95 % como valor respresentativo, el resto de valores constan en la figura 5.2.

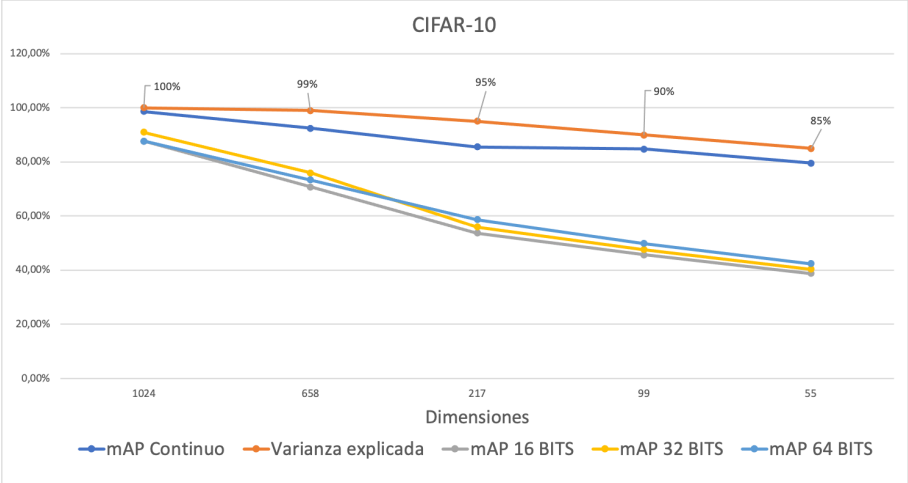


Figura 5.2: Varianza explicada acumulada, mAP del descriptor continuo y del discreto para distintos tamaños de descriptores en función del número de dimensiones para la base de datos *CIFAR-10*: Con apenas 217 dimensiones de las 784 totales es posible explicar un 95 % de la varianza.

5.3. Conclusiones

Las tablas 5.1 y 5.2 muestran la precisión mAP obtenida aplicando las técnicas explicadas en el capítulo 3 para las bases de datos *MNIST Handwritten Digits* y *CIFAR-10*, respectivamente. Se han incluido las precisiones, medidas con la misma métrica, alcanzadas por algoritmos desarrollados en los últimos años, algunos de ellos representados en la conferencia CVPR, *Computer Vision and Pattern Recognition*, una conferencia de gran prestigio dentro del campo de la visión por computador.

Una de las primeras conclusiones es que, para está aplicación, los métodos de preprocesamiento utilizados en otros ámbitos (en concreto, la aplicación de PCA a los datos de entrada) suponen una pérdida notable de precisión. Las figuras 5.1 y 5.2 representan la evolución de la precisión mAP del preprocesamiento por PCA al disminuir el número de dimensiones. Se puede apreciar que una disminución porcentual de un punto en la varianza retenida al reducir dimensiones conlleva una gran disminución en la precisión. Aunque el ahorro en coste sea considerable por la disminución de variables, el salto en precisión es considerable. Entre la posibilidad de entrenar una red convolucional con imágenes reconstruidas tras aplicar PCA o entrenar una red de capas completamente conectadas también hay cierta diferencia. La diferencia entre entrenar la imagen original y la reconstruida es considerable por lo que, en el caso de preprocesar mediante PCA, es mejor continuar con una red no convolucional cuya última capa tenga el tantas neuronas como dimensiones del descriptor continuo se tenga.

En ambas gráficas también se puede apreciar una pérdida en precisión mAP al convertir el descriptor continuo en discreto de entre el 10 y el 15 %.

Los dos métodos de blanqueamiento son una mejor alternativa que aplicar directamente PCA a la imagen sin procesar y se ha comprobado que aplicando ZCA, al obtener una aproximación a los datos originales pero decorrelacionados, la precisión es mayor que aplicando el blanqueamiento sin variantes.

Los dos métodos con los que mayor precisión se ha alcanzado han sido la reducción de dimensiones por LDA y la modificación introducida al método de [11] que consistía en cambiar a un emparejamiento exponencial en vez de aleatorio en la binarización. En concreto, con estas dos contribuciones se han obtenidos resultados que superan el estado del arte. Como única limitación del método propuesto, se debe mencionar que es un método que necesita supervisión y, por lo tanto, únicamente es aplicable en aquellos problemas en los que se disponga de anotaciones de clasificación.

Bibliografía

[1] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[2] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[3] J. Weng, N. Ahuja, and T. Huang. Cresceptron: a self-organizing neural network which grows adaptively. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, 1:576–581 vol.1, 1992.

[4] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] Léon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012.

[6] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012.

[7] Miguel Moreira and Emile Fiesler. Neural networks with adaptive learning rate and momentum terms. Technical report, Idiap, 1995.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Lev V Utkin, Maxim S Kovalev, and Ernest M Kasimov. An explanation method for siamese neural networks. *arXiv preprint arXiv:1911.07702*, 2019.

[11] Guillermo García Otin. Binarización de descriptores en redes neuronales. 2020.

[12] SC Ng. Principal component analysis to reduce dimension on digital image. *Procedia computer science*, 111:113–119, 2017.

[13] Mohammed Amin Belarbi, Saïd Mahmoudi, and Ghalem Belalem. Pca as dimensionality reduction for large-scale image retrieval systems. *International Journal of Ambient Computing and Intelligence (IJACI)*, 8(4):45–58, 2017.

[14] Hervé Jégou and Ondřej Chum. Negative evidences and co-occurences in image retrieval: The benefit of pca and whitening. In *European conference on computer vision*, pages 774–787. Springer, 2012.

[15] Hui Li, Xiao-Jun Wu, and Tariq S Durrani. Infrared and visible image fusion with resnet and zero-phase component analysis. *Infrared Physics & Technology*, 102:103039, 2019.

[16] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314, 2018.

[17] A. M. Martinez and A. C. Kak. Pca versus lda. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):228–233, 2001.

[18] Aleix M Martínez and Avinash C Kak. Pca versus lda. *IEEE transactions on pattern analysis and machine intelligence*, 23(2):228–233, 2001.

[19] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1183–1192, 2016.

[20] Shanshan Huang, Yichao Xiong, Ya Zhang, and Jia Wang. Unsupervised triplet hashing for fast image retrieval. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, pages 84–92, 2017.

[21] Kamran Ghasedi Dizaji, Feng Zheng, Najmeh Sadoughi, Yanhua Yang, Cheng Deng, and Heng Huang. Unsupervised deep generative adversarial hashing network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3664–3673, 2018.

[22] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. Deep learning of binary hash codes for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 27–35, 2015.

[23] Yueqi Duan, Jiwen Lu, Ziwei Wang, Jianjiang Feng, and Jie Zhou. Learning deep binary descriptor with multi-quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1183–1192, 2017.

[24] Erkun Yang, Tongliang Liu, Cheng Deng, Wei Liu, and Dacheng Tao. Distillhash: Unsupervised deep hashing by distilling data pairs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2946–2955, 2019.

[25] Yuming Shen, Jie Qin, Jiaxin Chen, Mengyang Yu, Li Liu, Fan Zhu, Fumin Shen, and Ling Shao. Auto-encoding twin-bottleneck hashing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2818–2827, 2020.