Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Red Neuronal Profunda de Índices Residuales para la recuperación de Imágenes

Tesis que presenta:
Edwin Efraín Jiménez Lepe

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis
Dr. Andrés Méndez Vázquez

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Octubre de 2017.

Red Neuronal Profunda de Índices Residuales para la recuperación de Imágenes

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:
Edwin Efraín Jiménez Lepe
Ingeniero en Sistemas Computacionales
Instituto Tecnológico de Tepic 2010-2015

Becario de CONACyT, expediente no. 589637

Director de Tesis
Dr. Andrés Méndez Vázquez

CINVESTAV del IPN Unidad Guadalajara, Octubre de 2017.

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Deep Residual Hashing Neural Network for Image Retrieval

A thesis presented by:
Edwin Efraín Jiménez Lepe

to obtain the degree of:
Master in Science

in the subject of:
Electrical Engineering

Thesis Advisors:
Dr. Andrés Méndez Vázquez

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, October 2017.

Deep Residual Hashing Neural Network for Image Retrieval

Master of Science Thesis
In Electrical Engineering

By:
Edwin Efraín Jiménez Lepe
Engineer in Computer Systems
Instituto Tecnológico de Tepic 2010-2015

Scholarship granted by CONACyT, no. 589637

Thesis Advisor:
Dr. Andrés Méndez Vázquez

CINVESTAV del IPN Unidad Guadalajara, October 2017.

Declaration of Authorship

I, Edwin Efraín Jiménez Lepe, declare that this thesis titled, "Deep Residual Hashing Neural Network for Image Retrieval" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"A picture is worth a thousand words."

English idiom, attributed to Tess Flanders.

Resumen

Red Neuronal Profunda de Índices Residuales para la recuperación de Imágenes

por Edwin Efraín Jiménez Lepe

Los métodos convencionales en la Recuperación de Imágenes Basada en Contenido (CBIR, por sus siglas en inglés) utilizan como entrada características visuales definidas a mano pero en algunas ocasiones dichos vectores de características no conservan la similitud que existía originalmente entre las imágenes.

Tomando ventaja de las recientes mejoras en el área de Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) proponemos un método de aprendizaje profundo que genera códigos binarios basados en las características aprendidas. Aprovechamos una idea previamente propuesta: en un enfoque supervisado los códigos binarios pueden ser aprendidos utilizando una capa oculta extra en el modelo, para representar las características principales que identifican a las clases en una base de datos.

Los resultados de la experimentación superan los algoritmos de indexado mas recientes en el dataset CIFAR-10. Un aspecto notorio es que la red neuronal propuesta tiene 8 millones de parámetros, una cantidad menor a la de otros métodos basados en CNN, por ejemplo, AlexNet (un modelo usado ampliamente en el área de visión por computadora) tiene 60 millones de parámetros.

Abstract

Deep Residual Hashing Neural Network for Image Retrieval

by Edwin Efraín Jiménez Lepe

Conventional methods in Content-Based Image Retrieval use hand-crafted visual features as input but sometimes such feature vectors do not preserve the similarity between images. Taking advantage of the improvements in the Convolutional Neural Networks (CNN) area we propose a deep learning method that generates binary hash codes based on the learned features.

We exploit a previously proposed idea in the field: in a supervised manner the binary codes can be learned adding an extra hidden layer to represent the main features that identifies the classes in a database. The experimental results outperforms the state-of-the-art hashing algorithms on the CIFAR-10 dataset.

It is remarkable that the proposed neural network has 8 million parameters (DRHN-15) less than other CNN based methods e.g. AlexNet (a widely used model in CV tasks) has 60 million.

Acknowledgements

Firstly, I want to thank God for the miracle of life and the gift of an universe full of questions.

I am extremely grateful with my advisor Ph.D. Andrés Méndez Vázquez for his wise words in the guidance of this work and all the knowledge that I could obtain of the classes and the talks that we had. I know that his contributions will accompany me throughout my life and helped me to understand that if I work hard there are no limits.

I would like to extend my gratitude to the researchers and professors of the computer department: Félix Ramos, Raúl González, Ernesto López, Mario Siller and Pedro Mejía, I have learned valuable things from all of you and the discipline of the scientific work.

I own my deepest gratitude to my family, without their support, patience and love this work would have been impossible. Especially my parents Efraín Jiménez and María Lepe for encourage me to pursue my dreams, my sister Cristian Jiménez to remember me the important things in life and my girlfriend Paloma Altamirano to help me focus to achieve my goals. I know I can't recover the moments that I lost but I did my best and I'll keep doing that because family is my biggest treasure.

I really appreciate all my classmates and friends, we were together in the endless days of homework and the good times that came in these two years. I hope that we can continue to sharing the laughter.

At last but not least, my gratitude and appreciation to CONACYT and CINVESTAV, for all the facilities provided for the realization of this research project.

Any omission in this brief acknowledgement does not mean lack of gratitude.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ever-growing information generated and shared on the web leads to an amazing field of opportunity when we search for different kinds of data. In 1992 we produced an average of 100 GB of information per day, moreover, in 2013 we produced 28,875 GB per second[1]. In 2016 talking of images we have the next information by minute: Instagram users liked 2,430,555 posts, Giphy served 569,217 gifs, Facebook Messenger users shared 216,302 photos, Youtube users shared 400 hours of video and Snapchat users watched 6,944,444 videos[2].

The study of image retrieval text based began in 1970. Some problems emerged like the cost of manual annotation for database images [1] and limited capability of using a restricted number of words to describe the content of an image.

The study of Content-Based Image Retrieval (CBIR) began in 1990, with images indexed by their visual features, such as texture and color [2]. In this context a strategy was to use global descriptors to measure similarity between query image and images in the database, nevertheless such descriptors are susceptible to changes as illumination, occlusion, intersection, and truncation [2].

In 2003 the BoW [3] model began to gain popularity in the area. However, in 2012 krizhevsky, *et al.* [4] outperforms previous results in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) using Convolutional Neural Networks (CNN), consequently there was an increased interest in deep learning methods. Therefore, this work presents a new approach to CBIR problem called Deep Residual Hashing Network (DRHN).

1.1 Motivation

According to [5] the advances in CBIR systems has been used in different areas such as:

- **Medicine** It can take advantage in teaching: finding important cases to expose to students. Research: searching for unusual patterns in patient records containing images. And diagnostics: similar cases can be found to help the clinical decision-making process [6].

[1] http://www.vcloudnews.com/every-day-big-data-statistics
[2] https://www.domo.com/blog/data-never-sleeps-4-0/

- **Biodiversity Information Systems** Using a combination of textual and image content based queries users can have results with images of similar living beings that exists in a specific country or those who share the environment and have similar shape, an example of this is given in the work of [7].
- **Digital Libraries** Some digital libraries support searching based on image content. One example is the digital museum of Taiwanese butterflies, which allow image queries based on color, texture and patterns [8]. Another interesting example is presented in [9] which supports geographic image retrieval.
- **Forensic** The retrieval based on image content has been used in criminal images, fingerprints and scene images [10].

Also according to [11] it has been used in detection of copyright violation on the Internet, Journalism and advertising, monitoring of satellite images, architectural and engineering design, art galleries, museums, cartography, radar engineering (detection and identification of targets, guidance of aircraft and missiles) and robotics for motion control through visual feedback.

So if we can improve the results in CBIR it benefits not only a specific application but provides better tools for a wide range of areas.

1.2 Problem Statement

The study of Content-Based Image Retrieval (CBIR) began more than twenty years ago, even so, there are open issues that keep people working on it. Some of this problems according to [12] are the next: the correct definition of a metric for image similarity, the semantic gap i.e. the lack of coincidence between the information that can be extracted from visual data and the interpretation that same data have for an user in a specific situation. Also there are other features that can affect the retrieval of images that fulfill user's query like: illumination, scale and size, background clutter, viewpoint and pose, rotation, occlusion and truncation, high intra-class variability and low inter-class variability.

In this thesis we propose the use of a CNN model to overcome some difficulties mentioned before, like illumination [13] and rotation [14], also we expect to reduce the semantic gap thanks to the filters learned by the model, as [15] explained: low level (lines, contrast, color), medium level (corners or other edge/color conjunctions, textures) and high level (more complex filters, class specific). Besides the model should learn a hash function that generate codes in a Hamming space such that class-separability is preserved and local neighborhoods are semantically relevant as [16] defined.

1.3 Objectives

1.3.1 General

Propose a CNN model to generate a feature based code for image indexing.

1.3.2 Specific

- Improve the current precision of retrieval in CBIR systems using a deep learning approach to generate codes for image indexing in a supervised manner.
- Analyze how model's depth affects the mean average precision (MAP) of retrieval task.
- Evaluate generated code length influence in the MAP of retrieval task.

1.4 Contributions

The contribution of this thesis is a Deep Model of Convolutional Neural Network that generate binary codes for image retrieval task, outperforming state-of-the-art results.

1.5 Hypothesis

The main idea of this work is based on the recent progress in computer vision using deep learning techniques, such as CNN. We think that a model that can classify images with high accuracy can lead to a good generation of feature based codes to retrieve similar images in a CBIR system.

1.6 Document Structure

The rest of this document is organized as follows. Information to understand the context of this thesis is presented in chapter 2. Related work is presented in chapter 3. The proposed model (DRHN) is explained in chapter 4. The chapter 5 presents the experimentation and results of the comparison to supervised and unsupervised methods in CIFAR-10 dataset. Finally, chapter 6 exposes the conclusions of the research and the ideas we have for future work.

Chapter 2

Theoretical Framework

In this chapter we analize two main concepts to understand the proposed idea. First, image retrieval based on annotated images (see subsection 2.1.1) and based on the content of the image (subsection 2.1.2). Second, Neural Networks (section 2.2) and Convolutional Neural Networks (section 2.3).

2.1 Image Retrieval

Image retrieval consist in the similarity of features of a given query image with those in the image database. The first approach was text-based it applies text retrieval techniques to image annotations. On the other hand, content-based apply image processing techniques to extract representative features and compare this features to retrieve images.

2.1.1 Text Based

Text-based image retrieval [17] can be based on annotations inserted manually or automatically to the images like keywords or descriptions. This keywords allows to retrieve, index and organize large collections of images.

To retrieve relevant images the user provide the keyword (or keywords) to search, but this have some drawbacks: for instance a little number of words cannot describe accurately the content of the image and some keywords may have multiple meanings, also the textual descriptions that an annotator provides could differ from another description on the same image i.e. different users tend to use diverse words to describe a same scene, since a picture have a personal meaning for every person and also can mean something different for the same person at different time, that is why content-based retrieval were born.

2.1.2 Content Based

Definition of CBIR based on [18].

Let $X = \{x_1, \dots x_N\}$ be a database with N images and $F = \{f_1 \dots f_N\}$ where f_i is a feature vector associated with x_i and contains the relevant information required for measuring the similarity between images.

Let T represent a mapping from the image space onto the h-dimensional feature space, f_i, i.e.,

$$T : x \to f, \tag{2.1}$$

where $x \in X$ and $f \in F$.

The similarity between two images x_i and x_j can be measured using similarity function $d(f_i, f_j)$.

The problem of retrieval can be posed as follows: Given a query image q, retrieve a subset of images M from X such that:

$$d(T(q), T(m)) \leq t, \qquad m \in M, \tag{2.2}$$

where t is a user-defined threshold. Instead of this, a user can ask the system to output, say, the top-k images which are most similar to the query image.

The main problem related to CBIR is how to define the correct features to extract in order to generate a feature vector that preserves the original similarity between images and which distance metric to use ensuring a good result in the retrieval.

According to [5] retrieval is an application-and-context-dependent task. It requires the translation of high-level user perceptions into low-level image features (semantic gap). Also we need an efficient way to store the indexed images to make a fast retrieval, but this is beyond the scope of the current work.

Using the idea of [19], the operation of a CBIR system is shown in Figure 2.1. The process of retrieval is sustented by an offline phase when feature extraction is performed to save the index related to each image in the database. When a query is performed is the online phase, we use the same feature extractor to identify the query image and perform a matching to retrieve the id of relevant images and fetch those related images from database for user visualization.

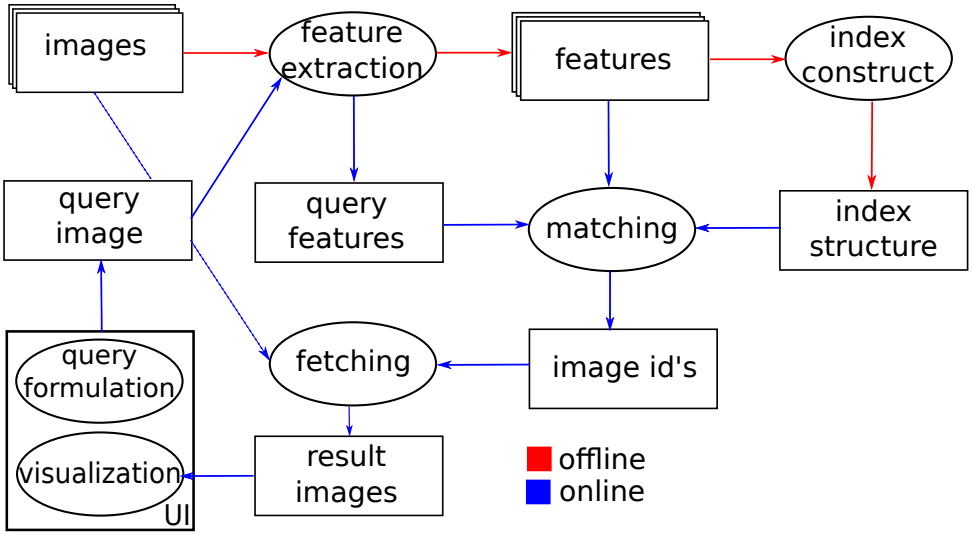

FIGURE 2.1: Operation of CBIR system. Adapted from [19].

2.1.3 How to represent the image?

The human eye is an expert to extract information from raw images but to perform computer vision tasks the algorithms usually needs features extracted therefrom. Those features can be *global* or *local*.

In a global feature representation the image is represented by one multi-dimensional vector, which contain information from whole image, measuring some aspects like color, texture or shape. On the other hand, local feature representation try to represent the image based on some *interest regions*, [20] defined a local feature as: "*it is an*

image pattern which differs from its immediate neighborhood".

The global representation is not invariant to some transformations and sensitive to clutter and occlusion [21], by comparison, the use of local features for large scale image search have higher performance [22].

The local features are conformed by a *feature detector* and *feature descriptor*. The first one detect points (x, y) or regions (x, y, σ) where σ denotes the scale of the region [23], while the second is a vector of values that describes and image patch around some interest point or region.

According to [21] a feature detector should (in theory) fulfill the next properties:

- Robustness, the algorithm should detect the same interest points regardless scaling, rotation, shifting and noise.
- Repeatability, the algorithm should be able to detect the same features in the same scene or object even if there exists some variations in viewing condition.
- Accuracy, the feature detection algorithm should localize the same image feature for images that have a correspondence.
- Generality, features detected by the algorithm can be used in a variety of applications.
- Efficiency, the algorithm should be fast, i.e., can work in real-time applications.
- Quantity, the algorithm should be able to detect most of the features in the image.

2.1.4 Feature Detectors

An overview of the history of local features detectors showing their invariance to some transformations is shown in Figure 2.2. The green surrounded methods are based in feature detection for regions with similar characteristics (blobs), and, the red surrounded ones are based in corner detection.

Harris Detector

[24] proposed a corner detector based in the idea of move a pixel centered window along all directions looking for a significant change.

Given an image I as input, $I(u, v)$ denote the intensity of the pixel in position (u, v). $E(x, y)$ denote the difference caused by a shift (x, y):

$$E(x, y) = \sum_{u,v} w(u, v)(I(u + x, v + y) - I(u, v))^2 \tag{2.3}$$

where $w(u, v)$ is a window function. according to [24] the term $I(u+x, v+y)$ can be approximated by a Taylor expansion as

$$I(u + x, v + y) \approx I(u, v) + (I_x(u, v)x + I_y(u, v)y)^2 \tag{2.4}$$

which can be re-written in matrix form:

$$E(x, y) \approx (x, y)M(x, y)^\top \tag{2.5}$$

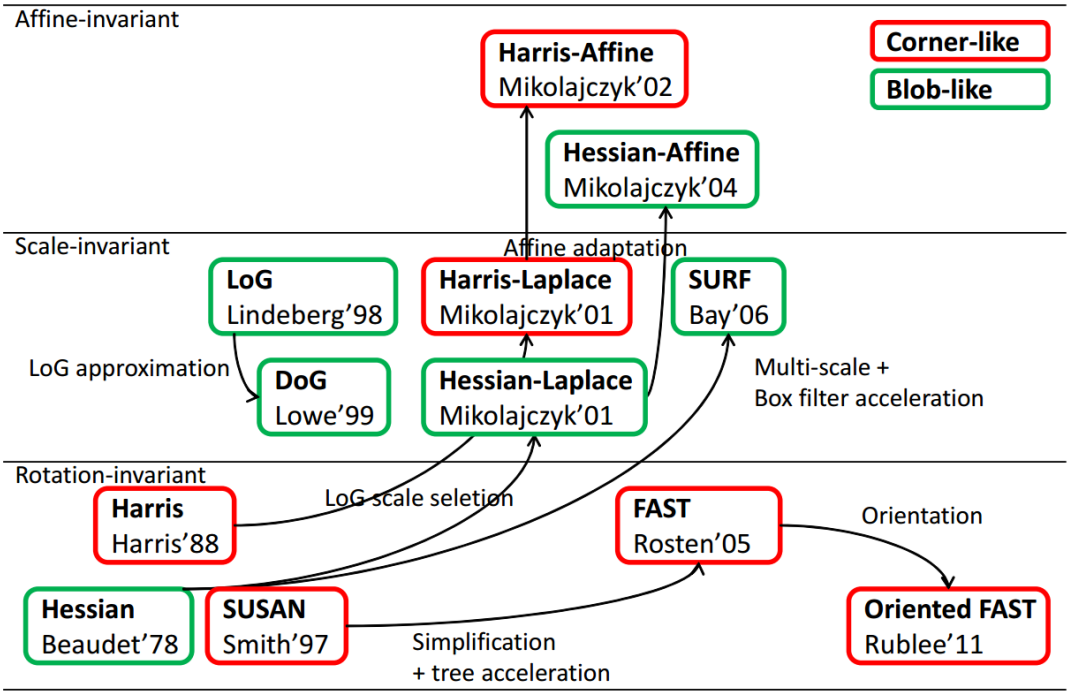

FIGURE 2.2: History of local feature detectors. Extracted from [23].

where

$$M = \sum_{u,v} w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.6}$$

Given that, if $E(x,y)$ becomes large with a shift (x,y) a corner have been detected.

Two modifications of this detector were proposed in 2001 [25] and 2002 [26]. The first one (Harris-Laplace) detects feature points using the Harris detector on multiple scales, after that such points are verified using the Laplacian of Gaussian to check if the scale is maxima or not in the scale direction. The second one (Harris-Affine) uses Harris-Laplace to detect feature points and refine them iteratively using the second moment matrix as described in their paper.

Hessian Detector

[27] detects a pixel (x,y) as feature point if the determinant of the Hessian matrix H (see Equation 2.7) is local-maxima compared with the neighboring pixels as shown in Equation 2.8.

$$H(x,y,\sigma) = \begin{bmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{xy}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{bmatrix} \tag{2.7}$$

$$det(H) = L_{xx}L_{yy} - L_{xy}^2 \tag{2.8}$$

In Equation 2.7 $L(x,y,\sigma)$ is an image smoothed by a Gaussian Kernel $G(x,y,\sigma)$ as shown in Equation 2.9.

Two variations of this detector were proposed in 2001 [25] and 2005 [28]. The first one is invariant to scale and the second one is invariant to affine transformations.

Laplacian-of-Gaussian (LoG) Detector

[29] proposed a scale-invariant regions detector. Which searches "for stable regions across all possible scales, using a continuous function of scale known as scale space" as described by [23], see Equation 2.9.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.9}$$

where $G(x, y, \sigma)$ is a Gaussian Kernel:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp(-(x^2 + y^2)/2\sigma^2) \tag{2.10}$$

In [29], they search for scale space extrema of a scale-normalized LoG denoted by $\sigma^2\nabla^2 L$, where σ^2 is a normalization term and

$$\nabla^2 L = L_{xx} + L_{yy} \tag{2.11}$$

Difference-of-Gaussian (DoG) Detector

The DoG $D(x, y, \sigma)$ is computed from the difference of two nearby scales separated by a constant k, as defined in [23]:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \\ &\approx (k - 1)\sigma^2\nabla^2 L \end{aligned} \tag{2.12}$$

DoG detects (x, y, σ) as a feature region if the response of $D(x, y, \sigma)$ is a local maxima or minima.

Speeded Up Robust Features (SURF) Detector

[30] is an approximation of the Hessian-Laplace detector. Approximates with box filters the partial derivatives L_{xx}, L_{xy} and L_{yy}. SURF detects scale-invariant blob-like features.

Features from Accelerated Segment Test (FAST) Detector

[31] detects pixels that are brighter or darker than neighboring pixels based on the next steps: For every pixel p, the intensities of 16 pixels on a Bresenham circle of radius 3 are compared with p, and classified as: brighter, similar and darker. If exist at least S connected pixels on the circle classified as brighter or darker, p is detected as a corner. $S = 9$ is usually used.

2.1.5 Feature Descriptors

SIFT Descriptor

Presented in [32], previous to generate the description the orientation of local region (x, y, σ) is estimated. Gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are computed using pixel differences.

$$m(x, y) = \left((L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2\right)^{\frac{1}{2}} \tag{2.13}$$

$$\theta(x,y) = \tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \tag{2.14}$$

where the intensity at $I(x,y)$ is denoted by $L(x,y)$, and I is a smoothed image by the Gaussian kernel.

After that, an orientation histogram is formed using sample pixels in the feature region to calculate the gradient orientations, such histogram contains 36 bins to cover the 360 degree range of orientations. Each pixel adds a score of $m(x,y)$ weighted by a Gaussian window to the bin corresponding to $\theta(x,y)$. The highest peak in the histogram corresponds to the dominant direction of local gradients.

Once that orientation has been asigned the SIFT descriptors are computed. The descriptor is represented by a 3D histogram of gradient location and orientation. Location is a 4×4 grid, and orientation is represented by eight bins, resulting in a 128-dimensional descriptor.

SURF Descriptor

[30] uses the Haar-wavelet responses in x and y directions. The characteristic scale of the SURF feature is denoted by s, the size of the Haar-wavelet is set to $4s$. Using a sliding window with size of $\pi/3$ the Haar-wavelet responses of pixels in a circle with radius of $6s$ are accumulated.

The feature region is rotated, after that is divided into subregions of size 4×4. For each subregion, d_x, d_y, $|d_x|$, and $|d_y|$ are computed at 5×5 spaced sample points, where d_x and d_y are the Haar-wavelet responses in x and y directions of size $2s$. These values are accumulated with the Gaussian weights, given a subvector $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. The subvectors of 4×4 are concatenated to form the 64-dimensional SURF descriptor.

Binary Robust Independent Elementary Features (BRIEF) Descriptor

[33] proposed a binary descriptor, i.e., is a bit string description of an image patch constructed from a set of binary intensity tests. Given a t-th smoothed image patch p_t, a binary test τ for d-th bit is defined by:

$$x_{td} = \tau(p_t; a_d, b_d) = \begin{cases} 1 & \text{if } p_t(a_d) \geq p_t(b_d) \\ 0 & \text{else} \end{cases} \tag{2.15}$$

where a_d and b_d denote relative positions in the patch p_t, and $p_t(\cdot)$ is the intensity of the point. The D-bit binary string $x_t = (x_{t1}, \ldots, x_{td}, \ldots, x_{tD})$ for p_t is obtained using D independent tests.

2.1.6 Image Representation

Bag-of-Visual Words (BoVW or BoW)

As described in [34] the BoW representation for images is based on the following steps: i) detect regions/points of interest, ii) compute local descriptors for such points/regions, iii) quantize the descriptors into words to form a visual vocabulary, and iv) find the occurrences of each word in the image to generate the BoW feature

(or a histogram of word frequencies), This four steps are represented in Figure 2.3.

Formally, BoW model can be defined as follows: Given a dataset D with n images, represented by the visual features extracted d, such that $D = d_1, d_2, \ldots, d_n$, an unsupervised algorithm is used to group D based on a number of visual words $W = w_1, w_2, \ldots, w_v$ where V is the cluster number. Then, data can be summarized in a $V \times N$ table of counts $N_{ij} = n(w_i, d_j)$, where $n(w_i, d_j)$ denotes how often the word w_i appears in image d_i.

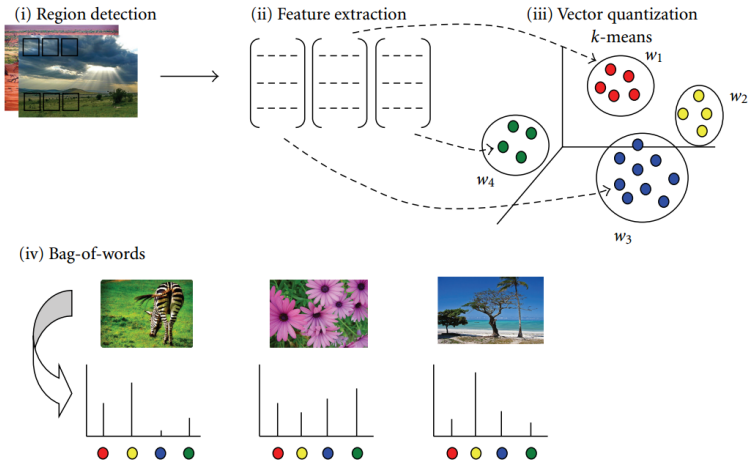

FIGURE 2.3: Steps to generate the BoW representation. Image extracted from [34]

Fisher Kernel and Fisher Vector

Let X be a data item (e.g. a feature vector), the generation of X is modeled by a probability density function $p(X|\lambda)$ where λ represent the parameters. [35] proposed to describe X by the gradient G_λ^X of the log-likelihood function, also known as Fisher score:

$$G_\lambda^X = \nabla_\lambda \mathcal{L}(X|\lambda) \tag{2.16}$$

where $\mathcal{L}(X|\lambda)$ denotes the log-likelihood function $\log p(X|\lambda)$.

The gradient vector as specified in [36] is a description of the direction that makes the parameters best fit the data. Fisher kernel is used on these gradients, and is based on the idea of natural gradient [37]:

$$K(X, Y) = G_\lambda^X F_\lambda^{-1} G_\lambda^Y \tag{2.17}$$

F_λ is the Fisher information matrix of $p(X|\lambda)$, defined as:

$$F_\lambda = E_x[\nabla_\lambda \mathcal{L}(X|\lambda) \nabla_\lambda \mathcal{L}(X|\lambda)^\top] \tag{2.18}$$

F_λ^{-1} is positive semidefinite and symmetric, so it can be decomposed as $L_\lambda^\top L_\lambda$. Therefore the Fisher kernel can be rewritten as a dot-product between gradient vectors $\mathcal{G}_\lambda$ with:

$$\mathcal{G}_\lambda^X = L_\lambda G_\lambda^X \tag{2.19}$$

where $\mathcal{G}_\lambda^X$ is refered as the Fisher vector.

Vector of Locally Aggregated Descriptors (VLAD)

The vector of locally aggregated descriptors [38] is an encoding technique that produces a fixed-length vector representation v from a set $X = \{x_1, \ldots, x_n\}$ with n local d-dimensional descriptors of a given image. A visual dictionary $C = \{\mu_1, \ldots, \mu_k\}$ is learned off-line, which is used as a quantization function assigning any input local descriptors to its closest centroid (visual word) as:

$$q : \mathbb{R} \rightarrow C \subset \mathbb{R}^d \tag{2.20}$$

$$x \mapsto q(x) = arg \min_{\mu \in C} \|x - \mu\|^2 \tag{2.21}$$

where $\|\cdot\|$ is the L_2 norm.

For each quantization index $i \in [1, \ldots, k]$ a d-dimensional sub-vector v^i is obtained by accumulating the difference between the descriptor and the centroid it is assigned to:

$$v^i = \sum_{x:q(x)=\mu_i} x - \mu_i \tag{2.22}$$

where $v = [v^1 \ldots v^k]$ is a D-dimensional vector, and $D = k \times d$.

To obtain the VLAD, v is normalized in two stages. First, each v_j, $j = 1$ to D, is modified as $v_j := |v_j|^\alpha \times sign(v_j)$, where $\alpha \leq 1$. Second, the VLAD vector is L_2-normalized as $v := \frac{v}{\|v\|}$

2.1.7 Similarity Measurement

According to [11] given two feature vectors f_1 and f_2 of N positions a widely used distance is the Minkowski-Form.

$$D(f_1, f_2) = \left(\sum_{1}^{N} \mid f_1(i) - f_2(i) \mid^p \right)^{\frac{1}{p}} \tag{2.23}$$

Each position of the vector have equal importance. When $p = 1$, the Equation 2.23 corresponds to the Manhattan Distance. If $p = 2$ it refers to Euclidean Distance, and $p = \infty$ it is called Chebyshev Distance.

In a CBIR system, when a search is performed the images with minor distance are retrieved. The choose of a distance measure can affect the performance of the system.

2.1.8 CBIR systems

In this subsection a short review of some CBIR systems is shown

- MIFile. Image similarity search engine based on Metric Inverted File [39] which allows to perform similarity search on huge datasets. It allows search using an url or uploading an image. It has 97 millions of images indexed, such images belong to YFCC100M (Yahoo Flickr Creative Commons 100M) dataset. It is available to use at `http://mifile.deepfeatures.org/`, developed in *Istituto di Scienza e Tecnologie dell'Informazione*.

- Lucignolo. Image similarity search engine which allow to search an uploaded image with labels added by the user. It has almost 106 millions of images. Available at http://lucignolo.isti.cnr.it/
- Akiwi. Developed by HTW Berlin and pixolution GmbH. It has 15 millions of images. Available at http://www.akiwi.eu/
- Anaktisi. CBIR that uses a combination of image descriptors as histogram color and texture information. It has 225,000 images from 13 databases. Available at http://orpheus.ee.duth.gr/anaktisi/
- Google Image Search. CBIR system that allows to search similar images given an user image query.

In Figure 2.4 the top-5 results of using an airplane image as query are shown for the CBIR systems described above.

FIGURE 2.4: Query of a plane image in 5 CBIR systems

2.2 Neural Networks

Neural Networks are a widely used model in machine learning we can define it based in the book of Haykin [40]. The basic unit of this model is a *neuron* which is defined in Equation 2.24 where w_{kj} represent the synapses or connecting links, k denotes the neuron and j represent the input signal x_j

Another element is the *bias* which help us to increase or decrease the output of a neuron before we apply the activation function, it is represented by b_k. The representation of a neuron is given in Figure 2.5 and Equation 2.25 where y_k represents the output signal of the neuron.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.24}$$

$$y_k = \varphi(u_k + b_k) \tag{2.25}$$

We can have a lot of neurons sharing the same inputs but with different synapses and we call it a *layer*. When we have one layer or more we call it *multilayer perceptron*

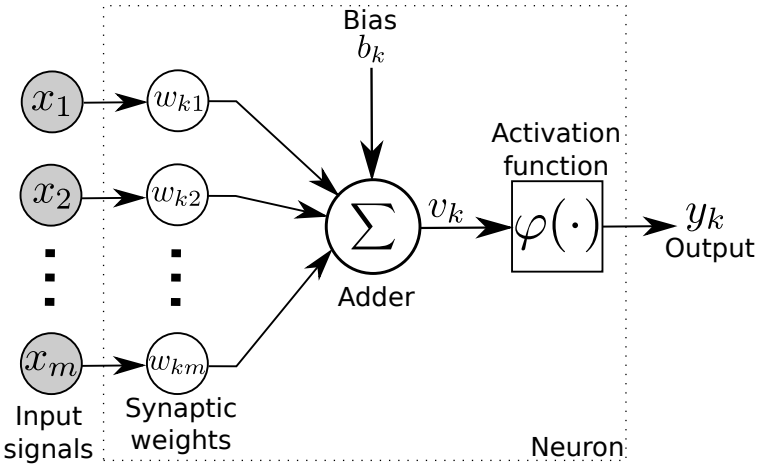

FIGURE 2.5: Example of a neuron. Adapted from [40].

which consist of an *input layer* where every unit is an input signal, one or more *hidden layers* where every unit is a neuron, and an *output layer* where the network give us the result of processing the input signal, which traverses the network layer-by-layer in a forward direction.

The *error back-propagation algorithm* is widely used to train this kind of network. It consists of two phases: a *forward pass* when we receive the input signal and it traverses the network until the output layer, and *backward pass* where the synaptic weights are adjusted according to an error-correction rule.

That is why in section 2.3 we define *feed forward (forward pass)* and *back propagation (backward pass)* for any kind of layer in the CNN.

2.2.1 FeedForward and Backpropagation

Figure 2.6 shows a multilayer perceptron with an input layer, two hidden layers and an output layer. This network is fully connected, it means that all outputs of a layer are received as input by all units in the next layer. The input signal is propagate through the network from left to right, layer-by-layer until it reach the output of the network, we call it *function signal*. This signal is calculated as a function of the inputs and associated weights applied to that neuron.

On the other hand, an error signal originates at an output neuron, and it's recalculated for every neuron to correct the synapses weights according to the expected output, Figure 2.7 shows the error signal flow during backpropagation on the same model.

2.2.2 Summary of the back-propagation algorithm

Haykin [40] defines five steps to apply the training algorithm:

1. Initialization: Usually bias is initialized in 0 and weights are picked from a uniform distribution with zero mean, sometimes in the range [-1,1].

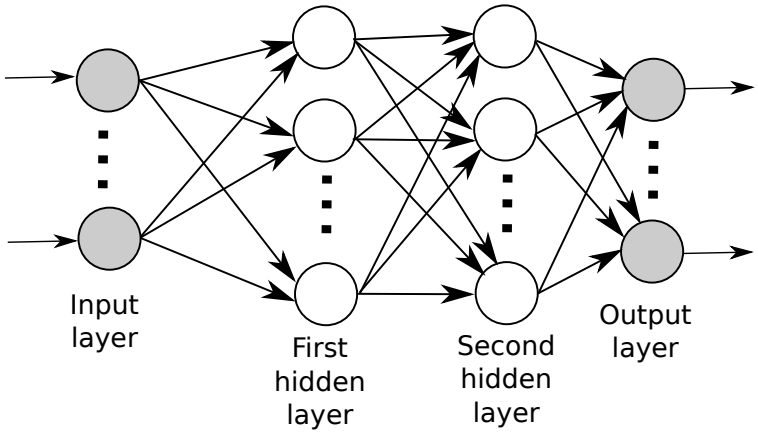

FIGURE 2.6: Example of feed forward in MLP. Adapted from [40].

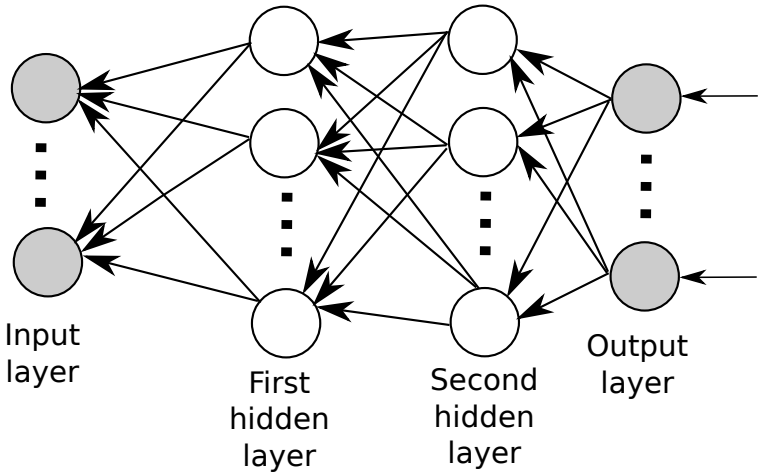

FIGURE 2.7: Example of backpropagation in MLP. Adapted from [40].

2. Presentations of Training Examples: When an iteration over all training examples is completed it is called *epoch*. For each element in the training set of examples steps 3 and 4 are applied.[1]

3. Forward Computation: The current training sample can be denoted as $(x(n), d(n))$, where $x(n)$ is a vector of features i.e. input layer and $d(n)$ is the desired output of the network for that given sample. We compute the value $v_j^l(n)$ for neuron j in current layer l proceeding forward through the network, layer by layer, calculating:

$$v_j^l(n) = \sum_{i=0}^{m_l} w_{ji}^{(l)}(n) y_i^{l-1}(n) \tag{2.26}$$

where $y_i^{l-1}(n)$ is the output signal of neuron i in previous layer $l-1$ at iteration n, and $w_{ji}^{(l)}(n)$ is the synaptic weight of neuron j in layer l that is fed from neuron i in layer $l-1$. For $i = 0$, we have $y_0^{l-1}(n) = +1$ and $w_{j0}^l(n) = b_j^{(l)}(n)$ is the bias applied to neuron j in layer l. Assuming the use of an activation

[1]The examples of the epoch can be divided in *mini-batches* of the same size and compute steps 3 and 4 with it.

function, the output signal of neuron j in layer l is

$$y_j^{(l)} = \varphi_j(v_j(n)) \tag{2.27}$$

If neuron j is in the first hidden layer (i.e., $l = 1$), set

$$y_j^{(0)} = x_j(n) \tag{2.28}$$

if neuron j is in the output layer (i.e., $l = L$, where L is referred to as the depth of the network), set

$$y_j^{(L)} = o_j(n) \tag{2.29}$$

Compute the error signal

$$e_j(n) = d_j(n) - o_j(n) \tag{2.30}$$

4. Backward Computation: Compute the local gradients of the network δ, defined by

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'\left(v_j^{(L)}(n)\right), & \text{for neuron } j \text{ in output layer L} \\ \varphi_j'\left(v_j^{(L)}(n)\right)\sum_k \delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n), & \text{for neuron } j \text{ in hidden layer l} \end{cases} \tag{2.31}$$

where $\varphi_j'(\cdot)$ denotes differentiation with respect to the argument.

After that we adjust the synaptic weights of the network in layer l according to the generalized delta rule:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \eta\delta_j^{(l)}(n)y_i^{l-1}(n) \tag{2.32}$$

where η is the learning-rate parameter.

5. Iteration: Repeat steps 3 and 4 until the error is within a valid range or when the rate of change in the error is under a valid threshold.

2.3 CNN

2.3.1 Convolution

2D convolution is equivalent to apply a filter in a image. It help us to learn the features that distinguished the images classes, instead of use hand-crafted features we let the CNN do the hard job. Convolution is the main layer of CNNs, the use of this kind of layer helped in the last years to obtain the best state-of-the-art results on image classification for small (mnist, cifar-10) and for really challenging datasets (ImageNet).

Feedforward

The feedforward of a convolutional layer it's pretty simple, but computationally expensive. We denote the convolution operation as $*$. It is the movement of a window over a subregion of the input image to make the addition of a Hadamard product, because in the subregion we do an element-wise multiplication and sum all the results of the window also (sometimes) we add a bias, the same process is repeated

until all the possible (and valid) subregions are covered.

The convolution in one dimension is expressed by [2]

$$g(t) = \int_{-\infty}^{\infty} f(\tau)h(t-\tau)d\tau \tag{2.33}$$

In 2-dimensional image processing the continuous convolution integral can be expressed:

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau_u,\tau_v)h(x-\tau_u, y-\tau_v)d\tau_u d\tau_v \tag{2.34}$$

Convolution of digital sampled images is similar but transforming the integrals into discrete summations over the image dimensions, m and n. In the equation, $h(0-\tau_u, 0-\tau_v)$ indicates a rotation of the image function $h(\tau_u,\tau_v)$ by 180 degrees about the origin (center of the function).

$$g(x,y) = \sum_{\tau_u}\sum_{\tau_v} f(\tau_u,\tau_v)h(x-\tau_u, y-\tau_v) \tag{2.35}$$

Before we define convolution for matrices there are two important concepts:

- **Stride**: It is the number of positions that we move the window on the input along the axes in each channel of the image.

TABLE 2.1: Subregions of the image with stride = 1

23	**-12**	16	90
12	**32**	12	45
-1	7	8	9
-2	-12	14	56

23	**-12**	**16**	90
12	**32**	**12**	45
-1	7	8	9
-2	-12	14	56

23	-12	**16**	**90**
12	32	**12**	**45**
-1	7	8	9
-2	-12	14	56

23	-12	16	90
12	**32**	12	45
-1	**7**	8	9
-2	-12	14	56

23	-12	16	90
12	**32**	**12**	45
-1	**7**	**8**	9
-2	-12	14	56

23	-12	16	90
12	32	**12**	**45**
-1	7	**8**	**9**
-2	-12	14	56

23	-12	16	90
12	32	12	45
-1	**7**	8	9
-2	**-12**	14	56

23	-12	16	90
12	32	12	45
-1	**7**	**8**	9
-2	**-12**	**14**	56

23	-12	16	90
12	32	12	45
-1	7	**8**	**9**
-2	-12	**14**	**56**

- **Padding**: Indicates how many columns and rows of zeros will be added in the border of the image. It allow us to apply different kinds of convolution: **full** we perform convolution using a $(k-1)$ padding, **valid** performs convolution without adding a zero padding, **same** performs a convolution using a $\frac{(k-1)}{2}$ padding (if stride $=1$), where k indicates the size of the filter. The use of padding helps to take advantage of existing information on the edges of the image.

[2] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/space/convol.htm

TABLE 2.2: Subregions of the image with stride = 2

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

TABLE 2.3: Padding to a 4x4 image

X

-12	16	90	34
32	12	45	-8
7	8	9	12
-12	14	56	18

X with 1-padding of zeros

0	0	0	0	0	0
0	-12	16	90	34	0
0	32	12	45	-8	0
0	7	8	9	12	0
0	-12	14	56	18	0
0	0	0	0	0	0

X with 2-padding of zeros

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-12	16	90	34	0	0
0	0	32	12	45	-8	0	0
0	0	7	8	9	12	0	0
0	0	-12	14	56	18	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

The definition of convolution for 2D matrix representation of images is shown in Equation 2.36 and it gives us an easy way to perform convolution, where w denotes the filter and x the input, m is the size of w and C indicates the channels of x and w, β indicates a bias, finally y is the output of the convolution operation. It can be observed that x needs to have a 180 degrees if we want y to be the output of convolution, otherwise it will be cross-correlation. The output of convolution is often called feature map.

$$y_{ij} = \beta + \sum_{c=0}^{C-1} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{abc} x_{(i+a)(j+b)c} \tag{2.36}$$

Examples of Convolution

- Input 3x3x1 image and 2x2x1x1 kernel, stride = 1, padding=0. The expected output dimension is 2x2x1. See Table 2.4

  But how is the step by step in convolution?, See Table 2.5

Backpropagation

To back propagate the error in a Convolutional Layer we need to calculate the gradient w.r.t three related parameters: weights, bias and the Input.

TABLE 2.4: Convolution to a 3x3 image

Input

16	24	32
47	18	26
68	12	9

Kernel

0	-1
1	0

Output of Convolution

23	-14
50	-14

TABLE 2.5: Convolution operation

16 * 0	24*-1	32
47*1	18*0	26
68	12	9

23	0
0	0

16	24	32
47*0	18*-1	26
68*1	12*0	9

23	0
50	0

16	24*0	32*-1
47	18*1	26*0
68	12	9

23	-14
50	0

16	24	32
47	18*0	26*-1
68	12*1	9*0

23	-14
50	-14

Bias

What is the relation between the bias and the result of a convolution? The bias add some specific value to the result in every channel, so for the error that we receive from an upper layer every value of the bias needs to change according to the error of the related channel.

It means, we have to sum all the error for each channel, see Equation 2.37, where $\frac{\partial L}{\partial b_{l-1,k}}$ is the error respect to bias at convolutional layer in channel k, and $\frac{\partial L}{\partial y_{l,k}}$ is the error propagated by the upper layer at channel k, also m is the number of rows and n is the number of columns at channel k in upper layer.

$$\delta b_{l-1} = \frac{\partial L}{\partial b_{l-1,k}} = \sum_{i}^{m} \sum_{j}^{n} \frac{\partial L}{\partial y_{l,k,i,j}} \tag{2.37}$$

Weights

There are two really nice tutorials that can be checked to understand the backpropagation for the weights and the input [41], [42].

See Equation 2.38 from [42] where x is the input to the convolutional layer and $\delta^{(y_l)} = \frac{\partial L}{\partial y_l}$ is the error from upper layer.

$$\delta w_{l-1} = \frac{\partial L}{\partial w_{l-1}} = \delta^{(y_l)} * x = x * \delta^{(y_l)} \text{(performing valid convolution)} \tag{2.38}$$

Input
To back propagate the error for lower layers, we simply apply Equation 2.39.

$$\delta y_{l-1} = \delta y_l * \text{flip}(w)(\text{performing full convolution}) \tag{2.39}$$

2.3.2 ReLU

In CNNs the activation function most widely used is Rectified Linear Unit (ReLU), which have a good training speed and superior results compare to others (tanh Figure 2.10, sigmoid Figure 2.9).

Feedforward

Basically the function returns 0 when it receives a negative value and the value itself in any other case. Given x, the output of a ReLU is:

$$y = max(x, 0) \tag{2.40}$$

A plot of ReLU is shown in Equation 4.2

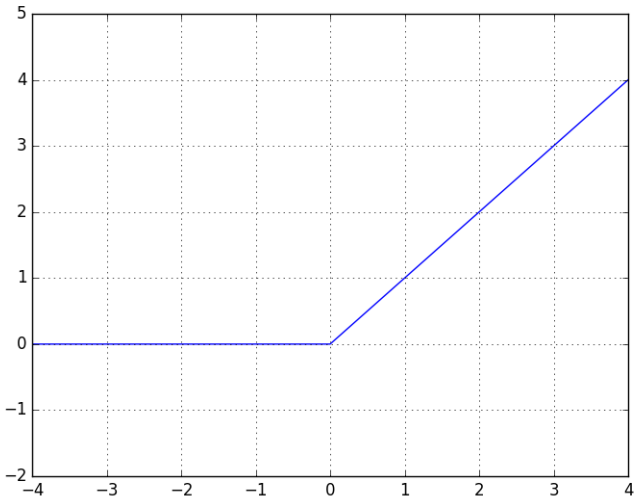

FIGURE 2.8: ReLU

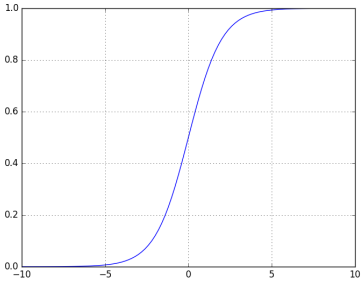

FIGURE 2.9: Sigmoid

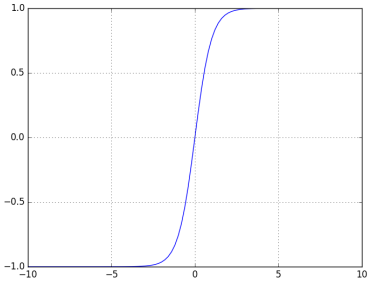

FIGURE 2.10: Tanh

As example, we assume that ReLU input is a 1-channel image of size 4x5 and the output is another 1-channel image of size 4x5 but without negative values.

TABLE 2.6: Feedforward ReLU

TABLE 2.7: Input to ReLU (x)

45	-12	16	90	34
-18	32	12	45	-8
-1	7	8	9	12
-2	-12	14	56	18

TABLE 2.8: Output of ReLU (y(x))

45	0	16	90	34
0	32	12	45	0
0	7	8	9	12
0	0	14	56	18

Backpropagation

The derivative of ReLU is the next:

$$y'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.41}$$

An explanation of the derivative can be found in the site [43]. Nevertheless, ReLU layers are connected to other layer, thus derivative can be rewritten according to [44].

$$\frac{\partial L}{\partial y_{l-1}} = \begin{cases} \frac{\partial L}{\partial y_l}, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.42}$$

Where $\frac{\partial L}{\partial y_{l-1}}$ is the error of the ReLU layer respect to the Loss of the network, $\frac{\partial L}{\partial y_l}$ is the back propagated error from the upper layer and x is the original input to ReLU.

2.3.3 Max-Pooling

"Is a form of non-linear down-sampling. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value" as [45] defined.

It is similar to a convolution operation but instead of apply a filter to any sub-region we just take the max value, also we can use an average pooling (the mean of the subregion) or a L^p-pooling [46], most architectures uses max-pooling for it's low computational cost.

Feedforward

A math representation of this operation:

$$y_{kij} = \max_{(p,q)\in R_{ij}} x_{kpq} \tag{2.43}$$

where y_{kij} is the output of the pooling operator related to the kth feature map in position(i, j), x_{kpq} is the element at (p, q) within the pooling region R_{ij} which represents a local neighborhood around the position (i, j), according to [47]

Backpropagation

To back propagate the error the derivative of the max-pooling operation is needed. According to [42], let be $g(x) = max(x)$:

$$\frac{\partial g}{\partial x_j} = \begin{cases} 1, & \text{if } x_j = max(x) \\ 0, & \text{otherwise} \end{cases} \tag{2.44}$$

So, the derivative can be generalized if we receive the error from an upper layer in the next fashion [44]:

$$\frac{\partial L}{\partial y_{l-1}}(x+p, y+q) = \begin{cases} 0, & \text{if } (y_l(x,y)! = y_{l-1}(x+p, y+q)) \\ \frac{\partial L}{\partial y_l}, & \text{otherwise} \end{cases} \tag{2.45}$$

2.3.4 Batch-Normalization

According to [48] training Deep Neural Networks is a complicated task because the distribution of each layer's input changes during training, as the parameters of the previous layers change. This constrains to use lower learning rates and be careful with parameter initialization.

They propose to normalize a part of the model architecture and perform the normalization for each training mini-batch. This allow to use much higher learning rates and be less careful about initialization. It works as a regularizer. The algorithm is shown in algorithm 1.

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch

Input : Values of x over a mini-batch: $B = x_i \dots m$;
Parameters to be learned: γ, β

Output: $y_i = BN_{\gamma,\beta}(x_i)$

$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean

$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$ // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$ // normalize

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$ // scale and shift

2.3.5 Dropout

In 2014, [49] proposed a technique to prevent overfitting in Deep Neural Networks. The main idea is to randomly drop units (including their connections) from the neural network during training, to prevent that units co-adapt too much. Some other common ways to prevent overfitting are the introduction of weigth penalties such as L1 and L2 regularization or stopping the training as soon as performance on validation data starts to get worse.

Apply dropout to a neural network is like sampling a 'thinned' network from it. This thinned network is formed by all units that survived dropout, see Figure 2.11. A neural net with n units is a collection of 2^n possible neural networks. The training

of a neural network with dropout can be seen as training a collection of 2^n thinned networks.

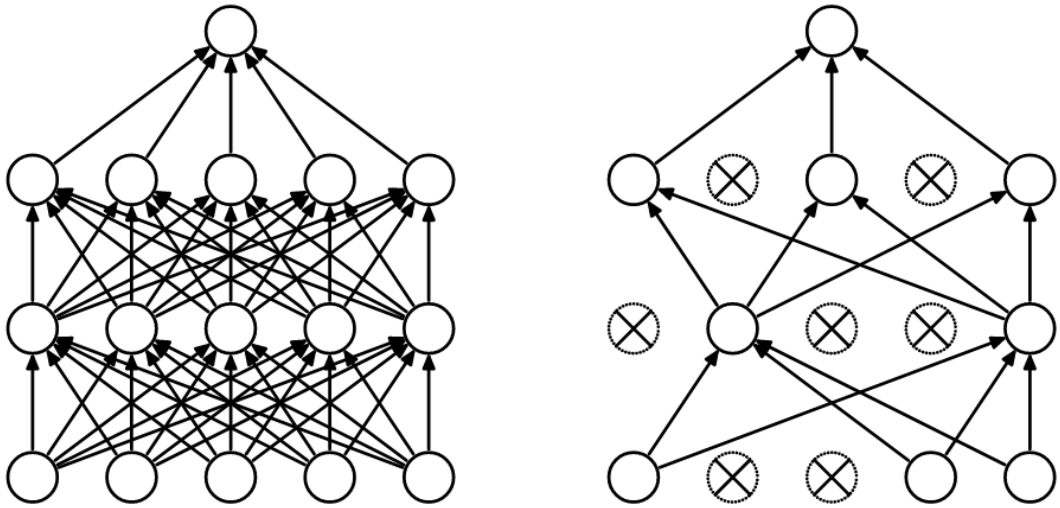

FIGURE 2.11: Dropout. **Left**: Standard neural net with 2 hidden layers. **Right**: Example of a thinned net produced by the application of dropout to neural net on the left. The units with no connected weights are dropped. Extracted from [49].

On the other hand, during test if a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p, see Figure 2.12.

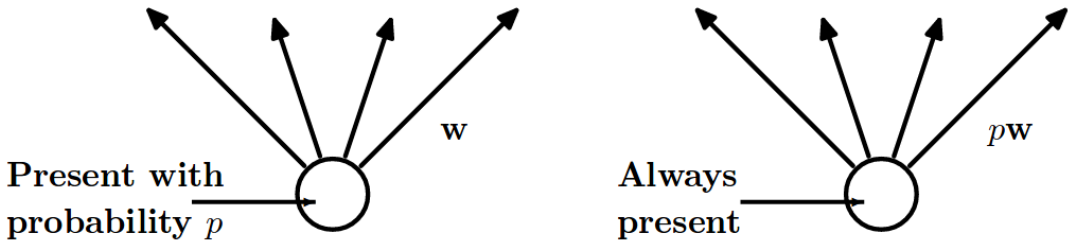

FIGURE 2.12: Dropout. **Left**: In training a unit is present with probability p and connected with weights w to the units in next layer. **Right**: In test time, the unit is present but weights are multiplied by p. Extracted from [49].

They found that the use of dropout leads to significantly lower generalization error on a wide variety of classification problems when is compared with other regularization methods.

2.4 CNN Architectures

Through the years many different combinations of the layers of CNN have appeared trying to learn the most possible relevant information to predict with high accuracy the correct image labels in the classification task, and also some of this have been used as base for some proposals of CBIR.

This section shows some recent architectures that achieved the state-of-the-art performance in the mentioned task.

2.4.1 LeNet

In 1988, *LeCun et. al.* in [50] proposed a model to recognize hand-written digits and called it Lenet-5. This model comprises seven layers, not counting the input which is a 32x32 pixel image, all of which contain trainable parameters (weigths). It is composed by sub-sampling, convolutional and fully-conected layers.

Layer C1 is a convolutional layer with 6 feature maps of size 28x28, each unit of feature map is connected to a 5x5 neighborhood. C1 contains 156 trainable parameters.

Layer S2 is a sub-sampling layer with 6 feature maps of size 14x14. Each unit is connected to a 2x2 neighborhood in the corresponding feature map in C1. The four inputs to a unit in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias. The output is passed through a sigmoidal function. The 2x2 receptive fields are non-overlapping. Layer S2 has 12 trainable parameters.

Layer C3 is a convolutional layer with 16 feature maps. Each unit is connected to several 5x5 neighborhoods in S2, this layer has 1,516 trainable parameters.

Layer C5 is a convolutional layer with 120 feature maps of size 1x1. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Layer C5 has 48,120 trainable connections.

Layer F6, contains 84 units and is fully connected to C5. It has 10,164 trainable parameters. The output layer is composed of Euclidean Radial Basis Function units (RBF), one for each class, with 84 inputs each. The RBF output y_i is computed as follows:

$$y_i = \sum_j (x_j - w_{ij})^2 \tag{2.46}$$

The architecture is shown in Figure 2.13

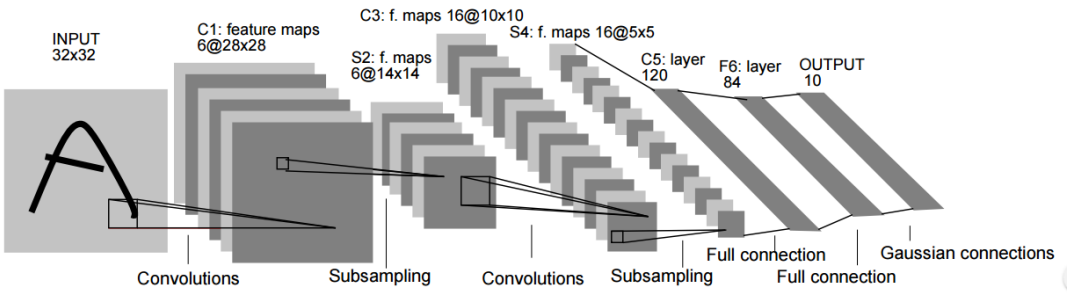

FIGURE 2.13: Lenet-5. Extracted from [50].

2.4.2 AlexNet

In the paper ImageNet Classification with Deep Convolutional Neural Networks [4] Krizhevsky *et al.* proposed an architecture to classify images of ImageNet dataset. Such architecture contains eight learned layers: five convolutional and three fully-connected, it is shown in Figure 2.14 and described in the subsequent paragraph.

The activation function used is Rectified Linear Unit (ReLU) which is described in subsection 2.3.2, also they use a "local response normalization" and is detailed in their correspondent paper.

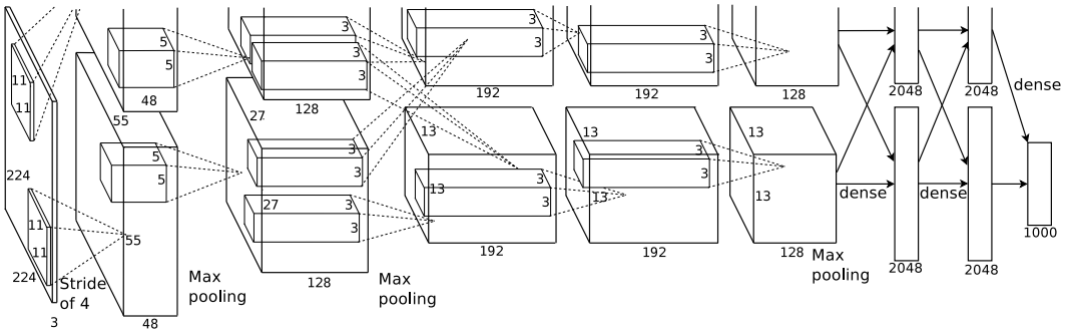

FIGURE 2.14: AlexNet. Extracted from [4].

The model is designed to reside in two GPUs as can be seen in the image kernels of second, fourth and fifth convolutional layers are connected only to previous kernels of the same GPU. Only kernels of the third convolutional layer are connected to all neurons of previous layer.

The input is a 224x224x3 image, the first convolutional layer contains 96 kernels of size 11x11x3 with a stride of 4. Second convolutional layer takes as input the (response-normalized and pooled) output of first layer and have 256 kernels of size 5x5x48. The next three convolutional layers are connected between them without pooling or normalization layers. Layer three has 384 kernels of size 3x3x256, fourth convolutional layer has also 384 kernels but of size 3x3x192 and fifth convolutional layer has 256 kernels of size 3x3x192. Fully-connected layers have 4096 neurons each.

The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.

2.4.3 GoogLeNet

In 2014 Google [51] propose a new deep convolutional network architecture code-named *Inception* which was responsible for setting the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14).

The basic block of layers is called Inception module and it is formed by 1x1, 3x3 and 5x5 convolutions and 3x3 max pooling distributing the input in 4 different flows

which are concatenated at the end of the module. This is represented in Figure 2.15.

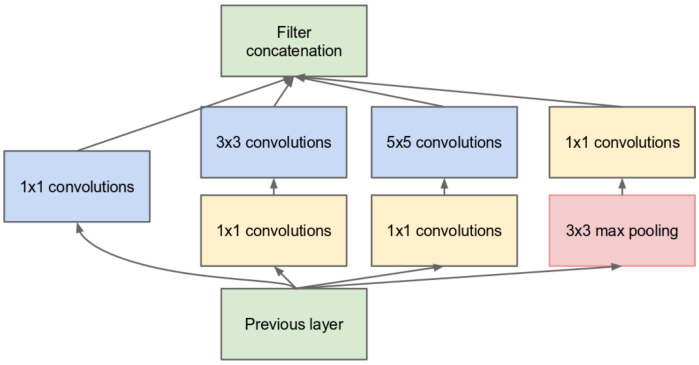

FIGURE 2.15: Inception Module. Extracted from [51].

A summary can be described in the next list, where each tuple represents the type of layer and the output size: (convolution, 112x112x64)→ (max pooling, 56x56x64) → (convolution, 56x56x192) → (max pooling, 28x28x192) → (inception, 28x28x256) → (inception, 28x28x480) → (max pooling, 14x14x480) → (inception, 14x14x512) → (inception, 14x14x512) → (inception, 14x14x512) → (inception, 14x14x528) → (inception, 14x14x832) → (max pooling, 7x7x832) → (inception, 7x7x832) → (inception, 7x7x1024) → (average pooling, 1x1x1024) → (dropout (0.4), 1x1x1024) → (linear, 1x1x1000) → (softmax, 1x1x1000).

This sequence of layers is represented in Figure 2.16.

2.4.4 The All-CNN

In 2015 the idea of discard the subsampling (pooling) layers was landed in the propose of [52], they substitute those layers using convolutional layers with stride of 2. A description of the model they used to experiment for classification for cifar-10 (obtaining 7.25 percent of error) is detailed in Table 2.9. They used LeakyReLU which is a modification or the ReLU function, see Equation 2.47.

$$f(x) = \begin{cases} x, & x \geq 0, \\ \frac{x}{3}, & x < 0. \end{cases} \tag{2.47}$$

2.4.5 Network In Network

Another architecture, was proposed in 2013 by [53], they defined a block as the stack of convolutional layers and a multilayer perceptron (mlpconv layer). They obtained an error of classification in CIFAR-10 of 10.41.

First, they define the calculation of a feature map as shown in Equation 2.48 where (i, j) denotes pixel index in the feature map, $x_{i,j}$ indicates the input patch centered at location (i, j), and k is used to index the channels of the feature map.

TABLE 2.9: All CNN

Layer name	Layer description
input	Input 126x126 RGB image
conv1	2x2conv. 320 LeakyReLU, stride 1
conv2	2x2conv. 320 LeakyReLU, stride 1
conv3	2x2conv. 320 LeakyReLU, stride 2
conv4	2x2conv. 640 LeakyReLU, stride 1, dropout 0.1
conv5	2x2conv. 640 LeakyReLU, stride 1, dropout 0.1
conv6	2x2conv. 640 LeakyReLU, stride 2
conv7	2x2conv. 960 LeakyReLU, stride 1, dropout 0.2
conv8	2x2conv. 960 LeakyReLU, stride 1, dropout 0.2
conv9	2x2conv. 960 LeakyReLU, stride 2
conv10	2x2conv. 1280 LeakyReLU, stride 1, dropout 0.3
conv11	2x2conv. 1280 LeakyReLU, stride 1, dropout 0.3
conv12	2x2conv. 1280 LeakyReLU, stride 2
conv13	2x2conv. 1600 LeakyReLU, stride 1, dropout 0.4
conv14	2x2conv. 1600 LeakyReLU, stride 1, dropout 0.4
conv15	2x2conv. 1600 LeakyReLU, stride 2
conv16	2x2conv. 1920 LeakyReLU, stride 1, dropout 0.5
conv17	1x1conv. 1920 LeakyReLU, stride 1, dropout 0.5
softmax	10-way softmax

$$f_{i,j,k} = max(w_k^T x_{i,j}, 0) \tag{2.48}$$

The calculation performed by mlpconv layer is shown as follows:

$$\begin{aligned} f_{i,j,k_1}^1 &= max({w_{k_1}^1}^T x_{i,j} + b_{k_1}, 0) \\ &\vdots \\ f_{i,j,k_n}^n &= max({w_{k_n}^n}^T f_{i,j}^{n-1} + b_{k_n}, 0) \end{aligned} \tag{2.49}$$

Here n is the number of layers in the multilayer perceptron. A general overview of the architecture is shown in Figure 2.17.

2.4.6 Spatially-Sparse Convolutional Neural Network

In 2014, [54] proposed a new model to process spatially-sparse inputs as handwritten characters where the relevant information resides in a few pixels. With that approach they achieve a error of 8.37% and 29.81% in cifar-10 and cifar-100, repectively. They said that the speed in which pooling is applied affects how a CNN generalize from training to test data, i.e. slow max-pooling retains more spatial information.

Definition of the architecture of DeepCNet: they defined two values associated with the model l and k. There are $l + 1$ convolutional layers separated by l layers of $2x2$ max-pooling layers. The number of filters in the n-th convolutional layer is taken to be nk. The spatial size of the filters is $3x3$ in the first layer, and then $2x2$ in the sebsequent layers of the model. At the top of the network is an output layer. The

activation function is ReLU and the output layer is a softmax.

When the input layer have a dimension of NxN with $N = 3 \times 2^l$, then the final convolutional layer produces a fully connected hidden layer. This network is know as DeepCNet(l, k).

They provide DeepCNet(4,100) as an example, with an input layer of size $N = 48 = 3 \times 2^4$ the layers of the model will be:

$$\text{input}-100C3-MP2-200C2-MP2-300C2-MP2-400C2-MP2-500C2-\text{output} \tag{2.50}$$

Another example can be seen in Figure 2.18 where the spatial sizes decrease from 96 down to 1.

The size of the hidden layers decays exponentially, so the cost of evaluate the network is roughly the cost of the first layers: $O(N^2k^2)$. The number of parameters that need to be trained is:

$$3^2Mk + 2^2k(k+1) + 2^2(2k)(3k) + \cdots + 2^2(lk)((l+1)k) = O(l^3k^2) \tag{2.51}$$

Also, inspired by the *Network in network* of [53] they proposed a DeepCNiN using a NiN layer as a convolutional layer where the filters have spatial size of 1×1. It is a single layer network that increase the learning power of a convolutional layer without changing the spatial structure. They placed NiN layers after each max-pooling layer and after the last convolutional layer. Using k and l on the same way that DeepCNet they call this proposal DeepCNiN(l, k). For example, DeepCNiN(4,100) is:

$$\begin{aligned} &\text{input} - 100C3-MP2 - 100C1 - \ldots \\ &\quad\cdots - 200C2-MP2 - 200C1 - \ldots \\ &\quad\cdots - 300C2-MP2 - 300C1 - \ldots \\ &\quad\cdots - 400C2-MP2 - 400C1 - \ldots \\ &\quad\cdots - 500C2-500C1 - \text{output} \end{aligned} \tag{2.52}$$

2.4.7 Deep Residual Network

In December of 2015 [55] proposed the use of shortcut connections in a block of layers, i.e. add the input of the block to the resulted output. They defined a residual block as:

$$y = F(x, W_i) + x. \tag{2.53}$$

Where x and y are the input and output vectors of the layer considered. $F(x, W_i)$ is a function that represents the residual mapping to be learned. For example, in Figure 2.19 $F = W_2\theta(W_1x)$ because the block has two layers, in which θ denotes ReLU. $F + x$ is performed by a shortcut connection and element-wise addition. A second ReLU is applied after the addition.

x and F dimensions should be equal in Equation 2.53, otherwise a linear projection W_s is performed by the shortcut connection to match the dimensions:

$$y = F(x, W_i) + W_s x. \tag{2.54}$$

When the dimensions increase they propose two options: A) Shortcut still performs identity mapping, adding a padding of zeros in x to match the dimensions; B)Equation 2.54 is used to match dimensions (by 1×1 convolutions).

The implementation of this model for CIFAR-10 classification task achieves 6.43% of error.

A representation of a Residual Network is shown in Figure 2.20

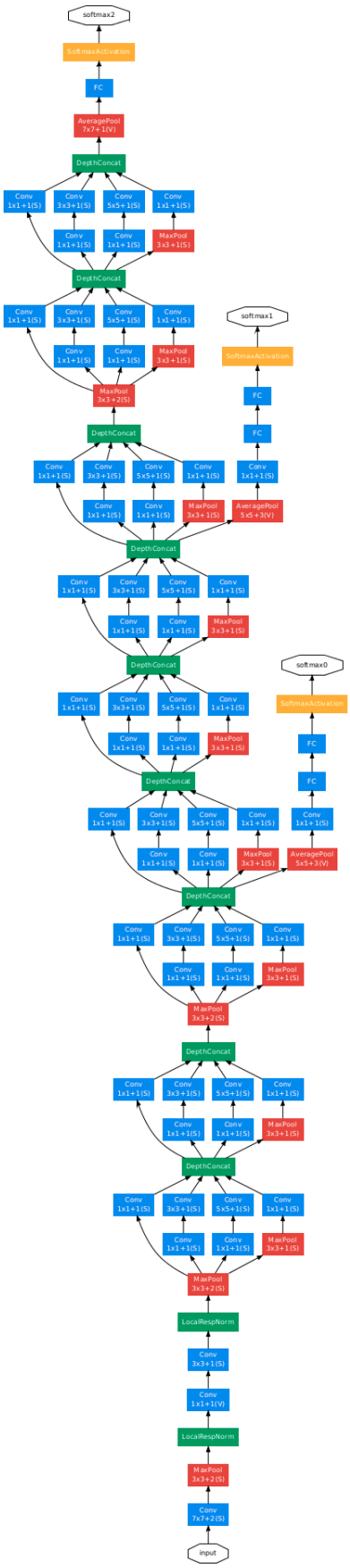

FIGURE 2.16: GoogLeNet. Extracted from [51].

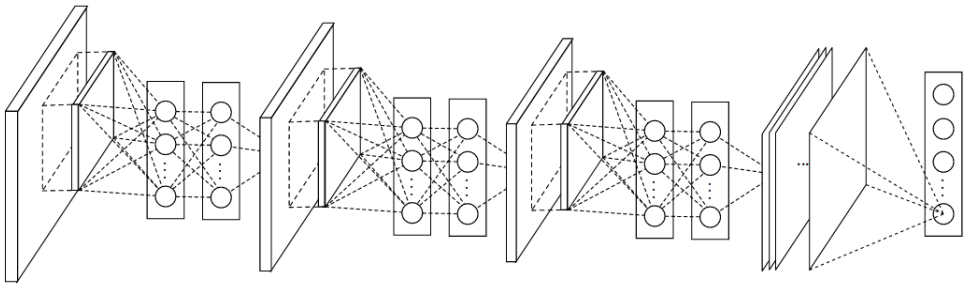

FIGURE 2.17: Network In Network. Extracted from [53].

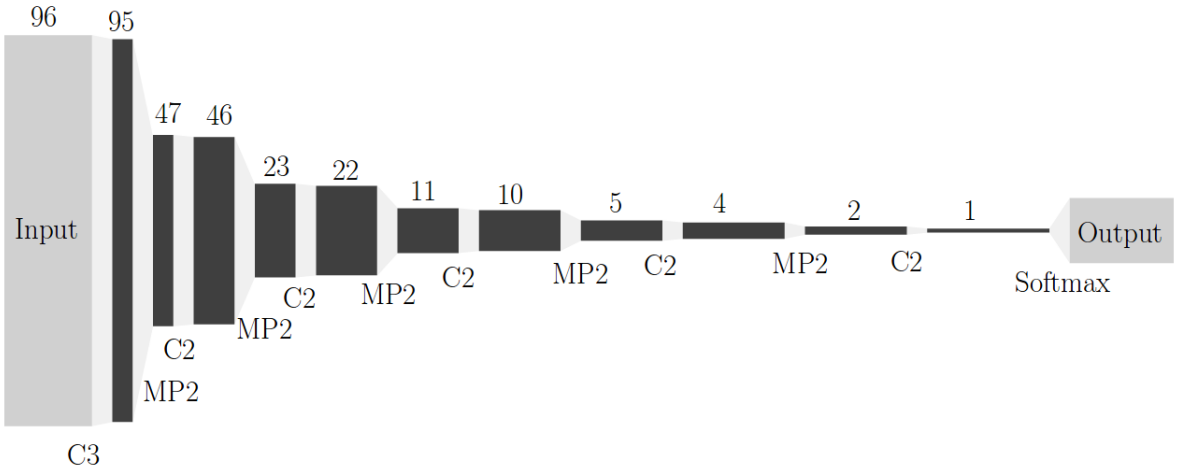

FIGURE 2.18: DeepCNet. Extracted from [54].

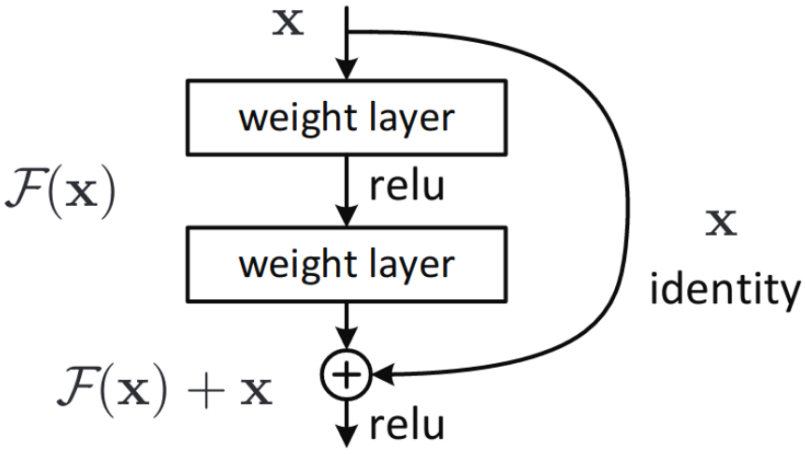

FIGURE 2.19: Residual Block. Extracted from [55].

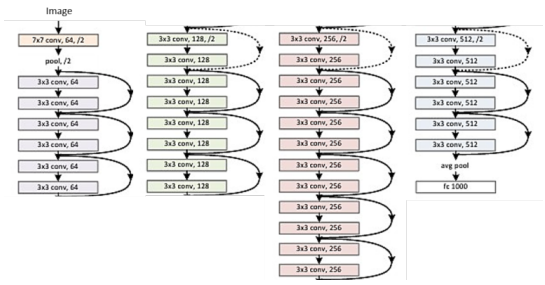

FIGURE 2.20: Residual Network Example. Extracted from [55].

Chapter 3

Literature Review

3.1 Generating hash codes for CBIR

When we talk about the generation of hash codes for image retrieval we can classify the methods as [56]:

- Supervised, which uses extra information like class labels or calculating pair-wise similarities. For example: Binary Reconstruction Embedding (BRE) [57], Minimal Loss Hashing (MLH) [58], Kernel Supervised Hashing (KSH) [59], CNNH[60], CNNH+ [60], CNNBH [61] and DLBHC [62].
- Unsupervised, which uses just data itself. For example: Spectral Hashing (SH) [63], Locality Sensitive Hashing (LSH) [64] and Iterative Quantization (ITQ) [65].

Most of the supervised methods use hand-crafted visual features as input to represent images (e.g., GIST [66]) which implies significant domain knowledge and extensive parameter tuning which is particularly undesirable [16]. Nevertheless, with the introduction of CNN for image classification [4], [51], [67] and retrieval [56], [60], [62] the use of raw image pixels as input became usual. Therefore, the risk of discard useful information disappears.

3.1.1 Unsupervised Approaches

Spectral Hashing

[63] proposed an algorithm to preserve similarity in a pair-wise fashion, looking for a coding space where the distance is expected to be small if in the original space the similarity was large.

Their proposal consist in the next steps that are detailed in their publication:

- Find the principal components of the data using PCA.
- Calculate the k smallest *single-dimension* analytical eigenfunctions of a weighted laplacian L_p using a rectangular approximation along every PCA direction.
- Pick the M eigenfunctions with the smallest eigenvalues among Md eigenfunctions.
- Threshold the analytical eigenfunctions at zero, obtaining the binary codes.

Locality Sensitive Hashing

In 1999, [64] presented the generation of a family of hash functions to map similar input items to the same hash code with a higher probability than dissimilar items.

The algorithm takes as input two parameters: a set of points P and the number of hash tables l. Each hash table T_i have associated a random hash function $g_i(\cdot)$, where $i = 1, \ldots, l$. Each point (or item) p_j is stored on bucket $g_i(p_j)$ of T_i.

To retrieve the top-k similar items (nearest neighbors) for a query point q the elements in bucket $g_i(q)$ of T_i are collected.

Iterative Quantization

In 2013, [65] proposed a two step algorithm to generate binary codes with similarity-preserving in an unsupervised manner.

They defined: $X \in \mathbb{R}^{n \times d}$ as the data matrix where n indicate the number of data points and d is the number of features, w_k is a column vector of hyperplane coefficients, the $sgn(v)$ function that returns 1 if $v \geq 0$ and 0 otherwise, and $W \in \mathbb{R}^{d \times c}$ is a matrix with columns w_k where c denotes the length of the code that will be generated.

The algorithm is based in minimize the quantization loss:

$$Q(B, R) = \|B - VR\|_F 2 \tag{3.1}$$

where $B = sgn(XW)$, $\|\cdot\|_F$ denotes the Frobenius norm, $V = XW$ and R is a orthogonal $c \times c$ matrix.

The two steps consists in given a fixed R update B, and given a fixed B update R [1], this updates are alternate iteratively until a locally optimal solution is found.

3.1.2 Supervised Approaches

Binary Reconstructive Embeddings

In 2009, [57] proposed to transform the original data to a low-dimensional space using a set of hash functions and a kernel. It aims to minimize the difference between Euclidean distance in the input space and the Hamming distance between binary codes.As is explained in [68] the objective function is:

$$min \sum_{(i,j) \in \epsilon} \left(\frac{1}{2} \|x_i - x_j\|_2^2 - \frac{1}{M} \|y_i - y_j\|_2^2 \right)^2 \tag{3.2}$$

where ϵ is a set of pairs of items, and M is the desired number of dimensions in the binary space.

[1] This two updates are explained in detail in the correspondent publication [65]

The kernel hash function used is:

$$y_{nm} = h_m(x) = sgn\left(\sum_{t=1}^{T_m} w_{mt} K(s_{mt}, x)\right) \tag{3.3}$$

where $\{s_{mt}\}_{t=1}^{T_m}$ are sampled data items, $K(\cdot,\cdot)$ is a kernel function, and $\{w_{mt}\}$ are the weights to be learnt.

The optimization of the objective function is performed using a coordinate-descent algorithm, which is presented in [57].

Minimal Loss Hashing

In 2011, [58] propose an algorithm trying to improve the results of [57]. Given p-dimensional points $\{x_i\}_{i=1}^{N}$ and S a set of pairs for which a similarity label is available. Similarity labels are given by $\{s_{ij}\}_{(i,j)\in S}$, such that x_i and x_j are similar when $s_{i,j} = 1$, and dissimilar otherwise.

Assuming mappings from $\mathbb{R}^p$ to H (where $H \equiv \{0,1\}^q$) given by $b(x;w) = thr(Wx)$ where $W \in \mathbb{R}^{q\times p}$ in which q denotes the length of the codes, $w \equiv vec(W)$, and $thr(Wx)$ is 1 only if the i^{th} element of (Wx) is positive with respect to a hyperplane in the input space, and 0 otherwise.

For a pair of binary codes $h, g \in H$, and a label $s \in \{0,1\}$, the loss function $L(h,g,s)$ penalizes distant binary codes for similar points and close binary codes for disimilar points. So, to learn w, the empirical loss to minimize is the next:

$$L(w) = \sum_{(i,j)\in S} L(b(x_i;w), b(x_j;w), s_{ij}) \tag{3.4}$$

Kernel Supervised Hashing

In 2012, [59] proposed the generation of a balanced hash function, i.e., the number of bits that are 1 and -1 is the same when all the bits in the generated binary codes are counted. To achieve that, they propose to minimize the next function:

$$f(x) = \sum_{j=1}^{m}\left(k(x_{(j)}, x) - \frac{1}{n}\sum_{i=1}^{n} k(x_{(j)}, x_i)\right) a_j \tag{3.5}$$

where $x_{(1)}, \ldots, x_{(m)}$ are m samples selected at random from data set X, a_j is a coefficient, n is data set size, m is a constant smaller than n, and k is a kernel function defined as follows: $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

3.1.3 Deep Learning Approaches

CNNH

In 2014, [60] propose the use of a deep neural network model to achieve good results in the approximate nearest neighbor (ANN) search in large-scale datasets. Their approach works as follows: given n images $I = \{I_1, I_2, \ldots, I_n\}$ and a matrix of similarity between pairs S (defined in Equation 3.6) learn a set of q hash functions

based on S and I.

$$S_{ij} = \begin{cases} +1, & I_i, I_j \text{ are semantically similar} \\ -1, & I_i, I_j \text{ are semantically disimilar} \end{cases} \tag{3.6}$$

The proposal consists in two stages: first a matrix of approximate hash codes H is generated by minimizing the reconstruction errors of Equation 3.7 where $\|\cdot\|_F$ is the Frobenius norm, $H \in \{-1, 1\}^{n \times q}$ is randomly initialized, and q is the length of the codes. Second H is used to train a deep model to learn image feature representation and a set of hash functions (see Figure 3.1), such model alternates convolution and pooling layers to end in a fully connected layer, the output of the model is a binary code of length q (CNNH) and the discrete class label of the image (CNNH+).

$$\min_{H} \|S - \frac{1}{q} H H^T\|_F^2 \tag{3.7}$$

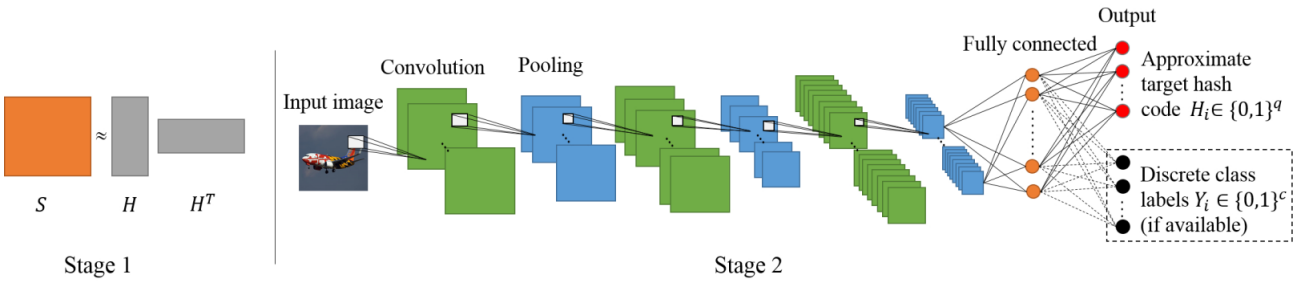

FIGURE 3.1: CNNH model

DNNH

In 2015 [56] used a triplet of images as input to a deep model aiming to learn a mapping "$F : I \rightarrow \{0, 1\}^q$ *such that an input image* I *can be encoded into a* q*-bit binary code* $F(I)$*, with the similarities of images being preserved*".

They use the loss function shown in Equation 3.8 where l_2 norm is used and $F(I)$ is the binary code for image I, $F(I^+)$ is the binary code for a image similar to I and $F(I^-)$ is the binary code for a image disimilar to I

$$\begin{aligned} & l_{triplet}(F(I), F(I^+), F(I^-)) \\ & = max(0, \|F(I) - F(I^+)\|_2^2 - \|F(I) - F(I^-)\|_2^2 + 1) \\ & \text{s.t. } F(I), F(I^+), F(I^-) \in [0, 1]^q \end{aligned} \tag{3.8}$$

The model they used is based on Network in network [53] and shown in Figure 3.2, also they introduced a divide-and-encode module which slice the output of the last convolutional layer into q slice of equal length and use it as input for a fully connected layer. The output of the model is the application of the sigmoid function to get a binary code.

DHN

In 2016, [69] propose another deep approach called Deep Hashing Network (DHN) to improve the results of [56] using AlexNet [4] and replacing the last fully connected layer *fc8* with a *fch* layer of K hidden units. *fch* tranforms the output of *fc7*

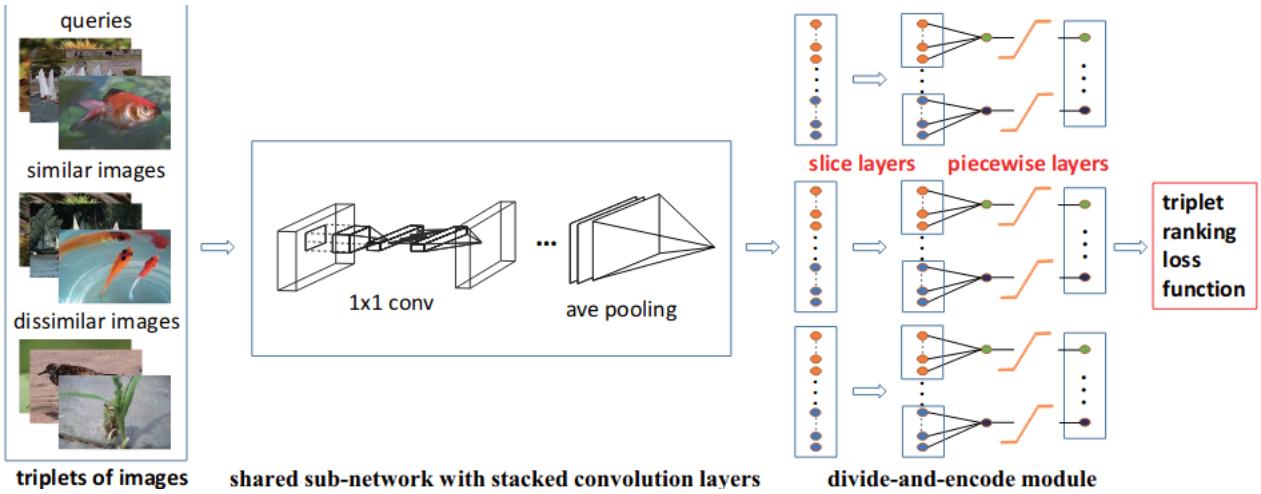

FIGURE 3.2: DNNH model

to K-dimensional hash coding by $h_i = z_i^l$, where $l = 8$ is the total number of layers and z_i^l is the hidden representation of the *fch* layer. $S = \{s_{ij}\}$ represent the pairwise similarity labels.

The optimization problem of DHN is given by:

$$\min_{\Theta} C = L + \lambda Q, \tag{3.9}$$

where $\lambda = 1/\epsilon$, ϵ is the diversity parameter, L is the pairwise cross-entropy loss (Equation 3.10), Q is the pairwise quantization loss (Equation 3.11), and Θ denotes the set of network parameters to be learned. The architecture of the proposal is shown in Figure 3.3.

$$L = \sum_{s_{ij} \in S} (\log(1 + exp(\langle z_i^l, z_j^l \rangle)) - s_{ij} \langle z_i^l, z_j^l \rangle) \tag{3.10}$$

$$Q = \sum_{s_{ij} \in S} \left(\| |z_i^l| - 1 \|_1 + \| |z_j^l| - 1 \|_1 \right) \tag{3.11}$$

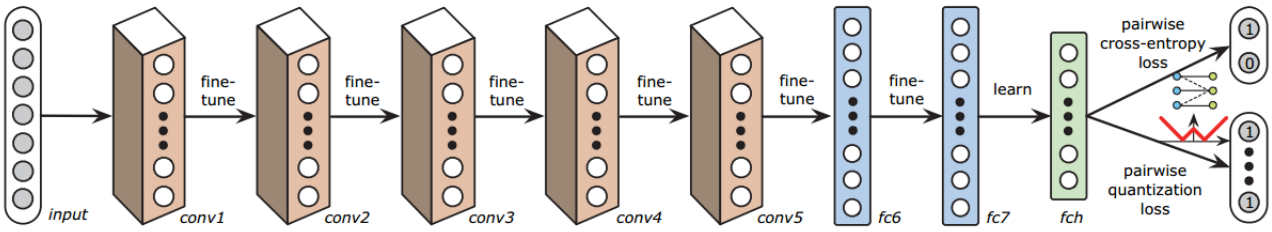

FIGURE 3.3: DNH model

CNNBH

In 2015, trying to improve the results of [60], [61] propose the use of another deep model to generate binary codes for image retrieval, but instead of generate an estimate H_i code they trained the model using the labels of the dataset as a classification

task, after that they extract the activation of the first fully connected layer and binarize it.

The architecture of the model they used is given in the following way: $1 \times 28 \times 28 - 32C5P2 - MP3S2 - 32C5 - H32 - D0.5 - H10$, i.e., input layer is an image with one channel and 28×28 pixels, a convolutional layer with 32 filters of size 5×5 with zero-padding of 2, after that a max pooling layer of size 3×3 and stride $= 2$, followed by another convolution layer with 32 filters of size 5×5, which is connected to a hidden layer with 32 units, a dropout layer with $p = 0.5$, and an output layer of 10 neurons which activation is a softmax function.

DLBHC

Another proposal was published in 2015 by Yahoo Taiwan [62], they used a similar idea to [61]. Using as base the architecture of AlexNet [4], Lin *et al.* propose a three modules framework for image retrieval (see Figure 3.4):

First, train AlexNet on the ImageNet dataset which contains ≈ 1.2M images divided in 1000 classes.

Second, the model is retrained with a specific dataset (target domain dataset) and a hidden layer H (latent layer) is added between F_7 and F_8.

Third, image retrieval. Given an input (query image) q, it traverses the model until layer H which activation is binarized to get the correspondent code for q, then a coarse-level search retrieves all the images that have the same binary code as a candidate pool p, after that a fine-level search is performed, in this phase similarity computation is performed between q and p to retrieve the results which similarity is better than some threshold or the top-k if a fixed number of images should be retrieve.

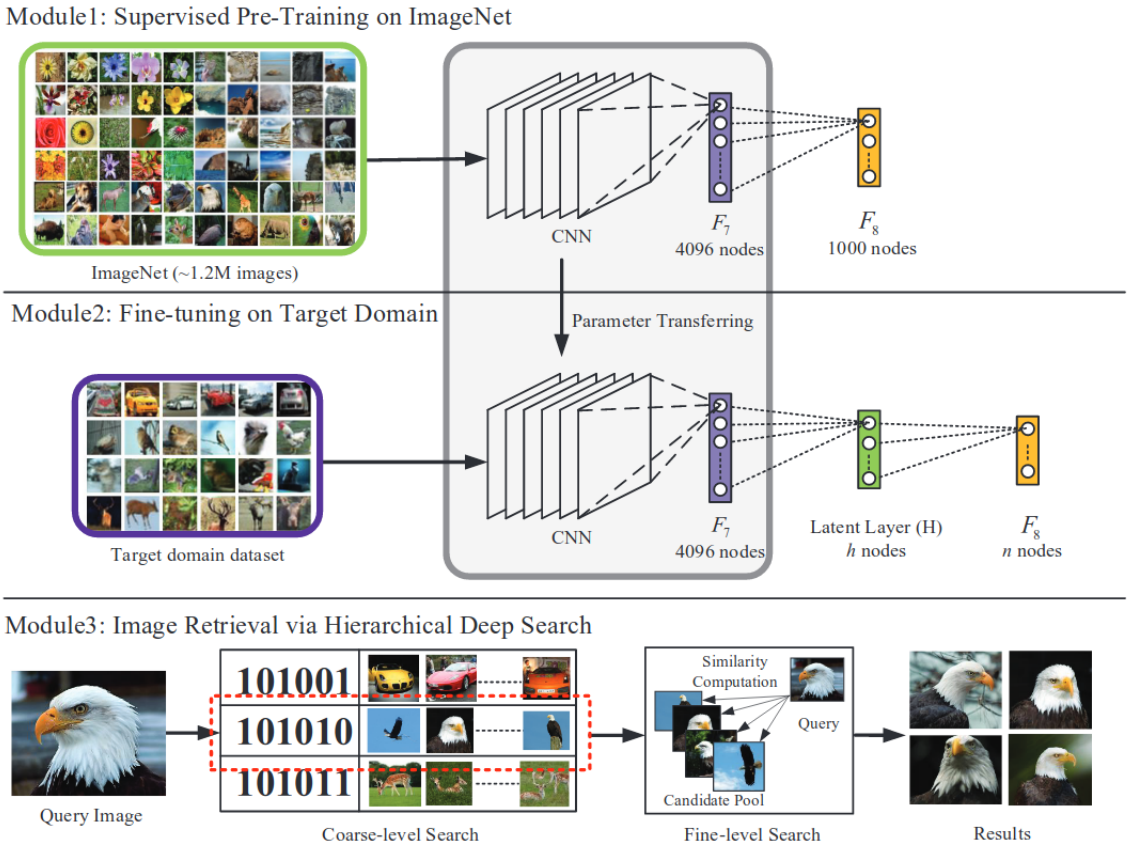

FIGURE 3.4: DLBHC model

Chapter 4

Deep Residual Hashing

The most widely used architectures of CNN [4], [50] uses convolution, ReLU and max pooling as the basic sequence of layers and repeat it until they have a fully conected layer at the end of the model. But [55] proposed an architecture where pooling layers are substituted by little convolutions with filters of size 3×3 and *stride* of 2. This emulates subsampling and extracts features in the same process. Achieving, therefore state-of-the-art results in image classification with a deeper and simpler model than previous architectures [4], [51], [54].

4.1 Deep Residual Hashing Neural Network

We propose a novel method called Deep Residual Hashing Neural Network (DRHN). The base of our model is a residual block [55] which is formed by the following operations:

- Convolution as is presented in Eq. 4.1, where w is a square filter of size m with c channels and x is the input with a zero padding set to one, this is performed without kernel rotation.
- Rectified linear unit (ReLU), is shown in Eq. 4.2, where x is the input.
- We use the Batch normalization as defined in [48] is shown in algorithm 1 where ϵ is a constant, and
- Element-wise addition (Add) is defined in Eq. 4.3. Is repeated for all channels c and positions ij in matrices of same dimensions x and z.

$$y_{ij} = \sum_{c=0}^{C-1} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{abc} x_{(i+a)(j+b)c} \tag{4.1}$$

$$y_{ij} = max\{x_{ij}, 0\} \tag{4.2}$$

$$y_{cij} = x_{cij} + z_{cij} \tag{4.3}$$

$$y = Add(BN_{\gamma_2,\beta_2}(conv(BN_{\gamma_1,\beta_1}(Relu(conv(x, w1))), w2)), x) \tag{4.4}$$

A Residual Group is the join of n Residual Blocks, which are defined in Eq. 4.4. The Average Pooling Layer calculates the average of all values in a channel. Hash Layer is defined in Subsection 4.2. The output of the model is a fully connected layer (see Equation 4.6) with Softmax function (Equation 4.5, where z is a K-dimensional vector with real values and $\sigma(z)_j$ is a K-dimensional vector of probabilities that add up to 1).

TABLE 4.1: Layers of Deep Residual Hashing Neural Network

Layer	Convolution dim.	Output dim.
Input		3x32x32
$BN_{\gamma,\beta}(Relu(conv(x, w1)))$	16x3x3x3	16x32x32
Residual Group (n)	16x16x3x3	16x32x32
Residual Group (n) with inc. dim.	*32x16x3x3, 32x32x3x3	32x16x16
Residual Group (n) with inc. dim.	*64x32x3x3, 64x64x3x3	64x8x8
Residual Group (n)	64x64x3x3	64x8x8
Residual Group (n)	64x64x3x3	64x8x8
Residual Group (n) with inc. dim.	*128x64x3x3, 128x128x3x3	128x4x4
Average Pooling Layer		128
Hash Layer		h
Fully Connected Layer with Softmax		10

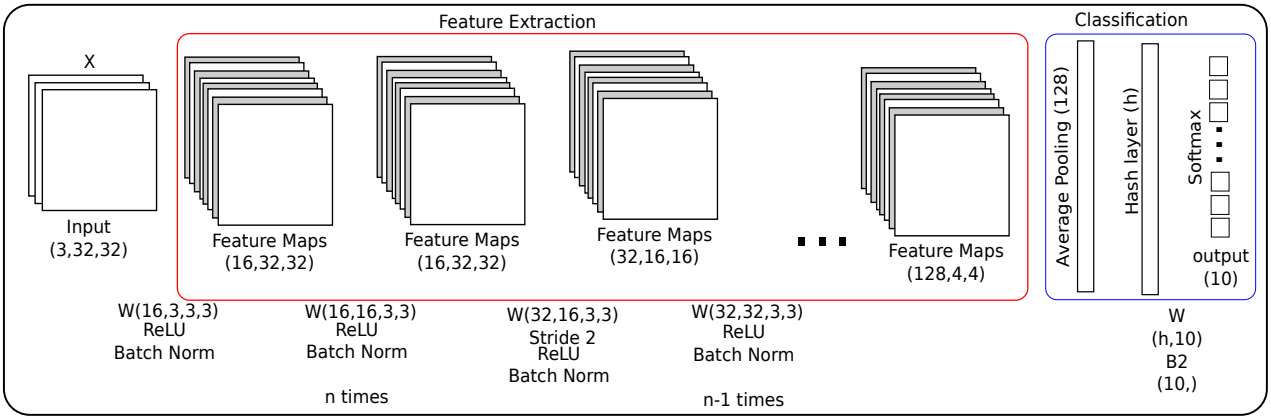

FIGURE 4.1: The model for Deep Residual Hashing

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, K. \tag{4.5}$$

The architecture of the proposed model is shown in Figure 4.2. Table 4.1 details the layers of the model indicating the name of the layer (or group of layers), the dimension of the related filters (number of filters $\times$ channels $\times$ height $\times$ width) and the dimensions of the output (channels $\times$ height $\times$ width).

When the number of channels in the output is increased, the first convolution of the block is performed with *stride* of 2 and the dimensions indicated with * in Table 4.1. Otherwise convolution is performed with *stride* of 1. Element-wise addition at the end of a block with increase dimention is possible if an identity shortcut is performed [55] i.e., add a zero padding to x for dimentions to match.

4.2 Hash Layer

The Hash Layer (H) is a fully conected layer with a sigmoid activation function s:

$$H = s(Wx + b) \in \mathbb{R}^h \tag{4.6}$$

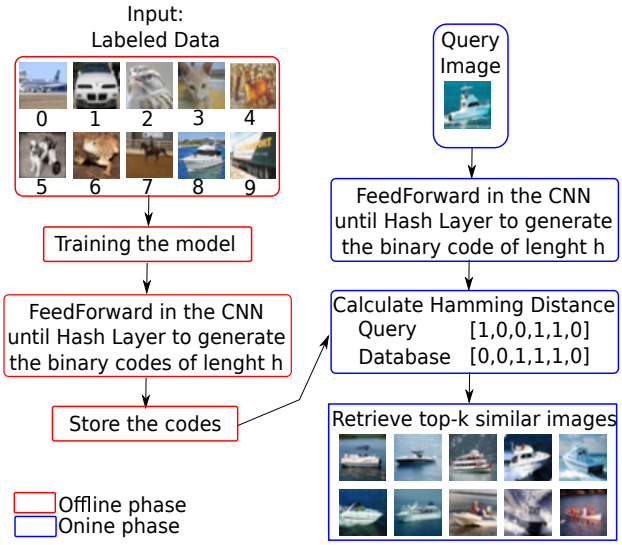

FIGURE 4.2: The steps for Deep Residual Hashing

where $x \in \mathbb{R}^d$ is the input of the layer, $b \in \mathbb{R}^h$ is a bias and $W \in \mathbb{R}^{h \times d}$ represent the weights that connect the units of x with H.

Given an image $I \in \mathbb{R}^{c \times z \times w}$, where c represents the channels of the image, z the height and w the width, the layers of the model from Input layer to H form a hash function that performs the mapping from $\mathbb{R}^{c \times z \times w}$ to $\mathbb{R}^h$.

Consequently, to obtain the binary code related to I as described by [62], we extract the output of H, and binarize the activation by a threshold to obtain the correspondent code. For each element in H we apply the sign function:

$$sign(H^i) = \begin{cases} 1, \text{if } H^i > 0.5 \\ 0, \text{otherwise} . \end{cases} \tag{4.7}$$

Once we have the binary codes we can perform image retrieval as follows: Let $IM = \{I_1, I_2, \ldots, I_n\}$ and $IM_H = \{H_1, H_2, \ldots H_n\}$ denote the dataset with n images and the corresponding binary codes, respectively. Given a query image I_q and its binary code H_q we can return the elements of IM where the Hamming distance between H_q and $H_i \in IM_H$ is lower than a threshold, or we can return the top-k if we need a specific k number of images returned by the query.

4.3 Workflow

To sum up, the way DRHN works is shown in Figure 4.2. In the offline phase the model is trained and the binary codes of data are stored. In online phase the user's query traverses the model until hash layer H is reached and its binary code is used to search the closest stored codes in database.

An example of the image process in the network is shown in Figure 4.3 where the output of first layer is an activation map of 16 channels. Also, an example of the feedforward in the layer until the generation of the binary code is shown in Figure 4.4.

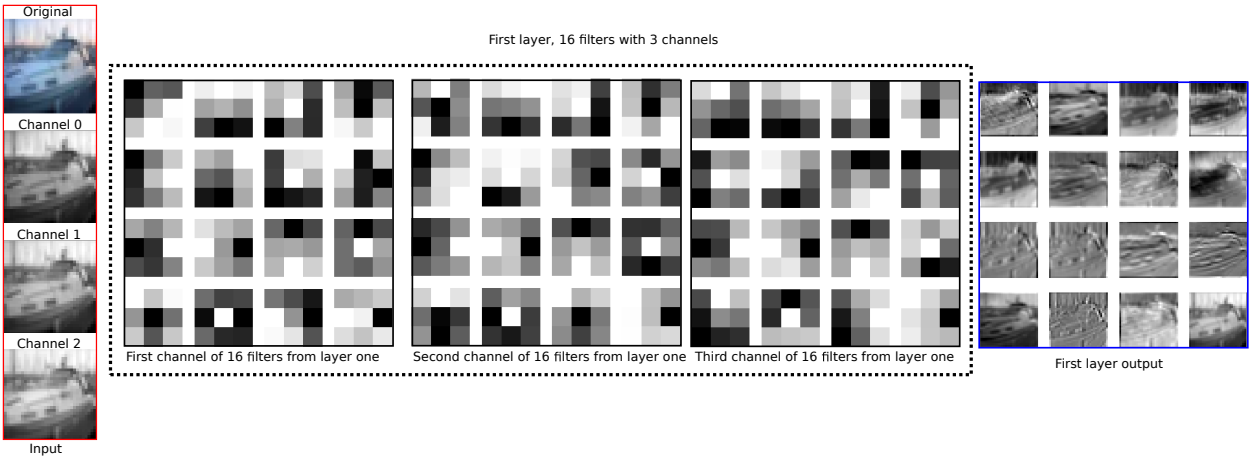

FIGURE 4.3: Input, filters and output of first layer in DRHN-5

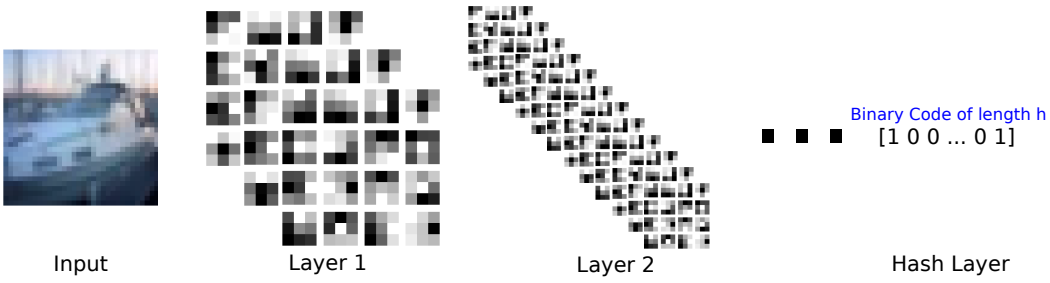

FIGURE 4.4: Example of binary code generated by the model given an rgb image

Chapter 5

Experiment and Results

5.1 Experiments and Results

In this section, we show the evaluation of the proposed method on the next dataset:

- **CIFAR-10 Dataset** [70] consists of 60,000 color images divided in 10 classes, each one with 6,000 images of 32x32 pixels. The dataset is splitted into training and validation set, 50,000 and 10,000 images respectively. We perform augmentation of the dataset by mirroring the images.

We compare DRHN with n=15 against three unsupervised methods LSH [64], SH [63] and ITQ [65], and ten supervised methods CNNH[60], CNNH+ [60], KSH [59], MLH [58], BRE[57], ITQ-CCA [65], DNNH [56], DHN [69], CNNBH [61] and DLBHC [62].

After that, we compare the precision (Eq. 5.1) when our proposal have a variation in the depth using n=5,9,10,11 and 15.

We used the implementation of [55][1] in Lasagne [71] [72] as base to build and train the proposed model.

5.1.1 Evaluation Metrics

According to [73] and [62] we use a ranking based criteria as evaluation metric. Given a query (image in our case) q and a similarity measure (Hamming Distance), we can assign a rank for each dataset image. We calculate the precision of the top k ranked images with respect to a query q as $P@K$ using the indicator function $X(i)$:

$$P@K = \frac{\sum_{i=1}^{k} X(i)}{k} \tag{5.1}$$

$$X(i) = \begin{cases} 1, \text{if } L(q) = L(i) \\ 0, \text{otherwise} \end{cases} \tag{5.2}$$

where $L(q)$ denotes query image q label and $L(i)$ denotes ith ranked image label. Also we used mean average precision (mAP) which is a standard evaluation metric in CBIR to perform a fair comparison:

$$mAP = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{m_q} \sum_{k=1}^{m_q} P@K \text{ (if } k^{th} \text{ item was relevant)} \tag{5.3}$$

[1] https://github.com/Lasagne/Recipes/blob/master/papers/deep_residual_learning/Deep_Residual_Learning_CIFAR-10.py

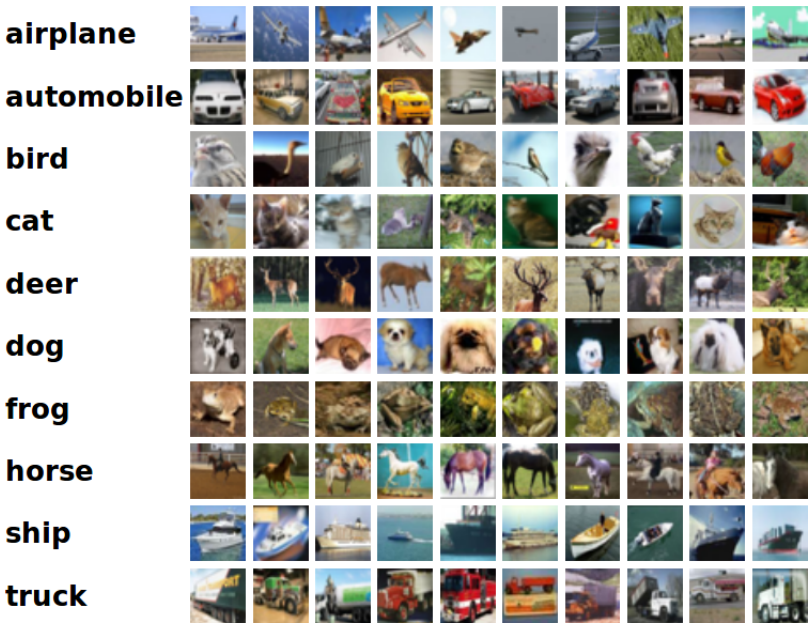

FIGURE 5.1: Random images from the ten classes of CIFAR-10.

where $|Q|$ represent the number of queries and m_q is the number of result images for a given query q. This metrics give us a simple way to compare the results of the proposed method but we only measure the effectiveness taking into account if the images retrieved belong to the same class, discarting the visual similarity, this still an open problem as [74] said related to the evaluation metrics: "*While precision and recall are a useful tool in information retrieval, in the context of image databases, they are not sufficient because the selection of a relevant set in an image database is much more problematic than in a text database because of the more problematic definition of the meaning of an image. [...] In the case of an image, relevance is much less stable because of the larger number of interpretation, of a image separated from a linguistic context. Moreover, no analysis in terms of semiotically stable constitutive elements can be done therefore, the correlation between image relevance and low-level feature is much more vague*".

5.1.2 Results on CIFAR-10 dataset

Performance of Image Classification

We trained the model for the image classification task. Using as cost function the negative log likelihood (also know as cross-entropy), which indicates how far is the probability of predicted label from expected label. It is detailed in Equation 5.4, where x represent the number of samples in the training set and a_y^L is the output of softmax function.

$$C = -\frac{1}{n}\sum_x (ln(a_y^L)) \tag{5.4}$$

After the 43th epoch we obtained an accuracy (Equation 5.5 where N denotes the number of elements in the dataset, y_i is the expected label, $\hat{y}_i$ is the predicted label and cl is a function that returns one if both label are equal) of 93.70% (which is slightly better than the original implementation accuracy $\approx$ 93.25%). It means that

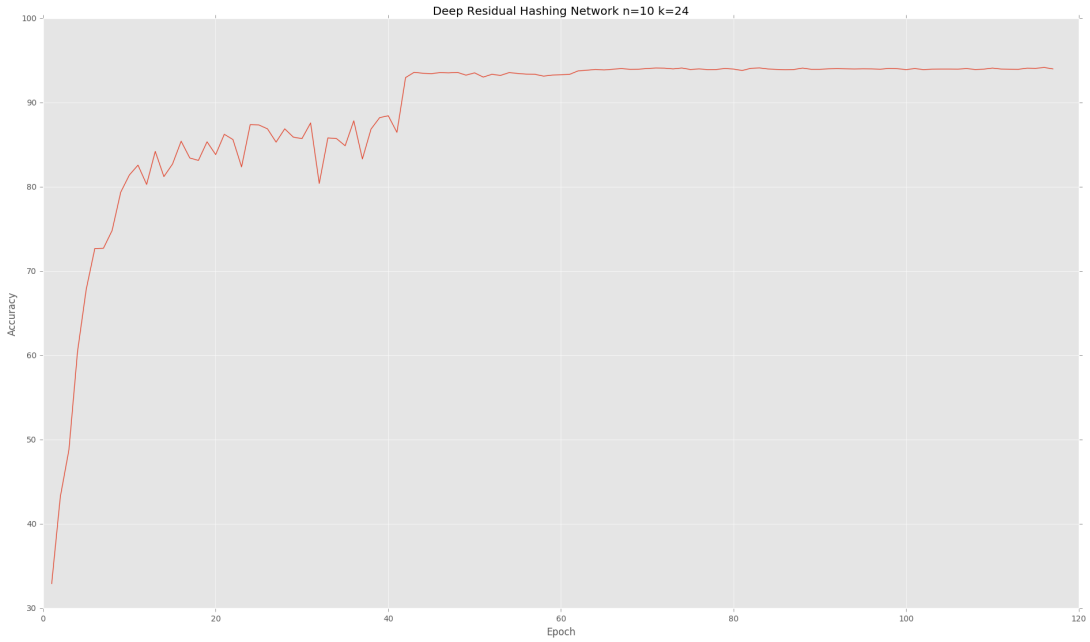

FIGURE 5.2: Accuracy increase with respect to number of epoch in DRHN with n=10, k=24.

the binary layer does not affect in a negative way the performance of the model. The confussion matrix can be seen in Figure 5.4.

Why we stopped training at epoch 43? It is well know that a correct answer for a stop criteria in training a model doesn't exist. Some people use a fixed number of epochs just to stop in some future moment, sometimes when a desired accuracy is reach the model is considered trustworthy enough and stop learning, and in other ocasions when the error in the training set or testing set is below a certain threshold.

We decided to stop training in a specific number of epoch because after experiment in some models ($n = [8, 9, 10]$) they show a similar behavior, such demeanor is shown in the images below. Figure 5.2 shows that after epoch 43 accuracy increases just a little after the double of epochs, on the other hand Figure 5.3 shows how the loss in both sets is decreasing until epoch 43, after that epoch the loss in testing set begin to increase even when training loss keep descending, it means that the model begin to overfitting and is not so useful to predict a correct label for new elements.

Finally the accuracy for the model using $n = [1 \dots 16]$ and generating binary codes of length $k = [12, 24, 32, 48, 128]$ is shown in Table 5.1.

$$
\begin{aligned}
ACC &= \frac{\sum_{i=1}^{N} cl(y_i)}{N} \\
cl(y_i) &= \begin{cases} 1, \hat{y}_i == y_i \\ 0, \text{otherwise} \end{cases}
\end{aligned}
\tag{5.5}
$$

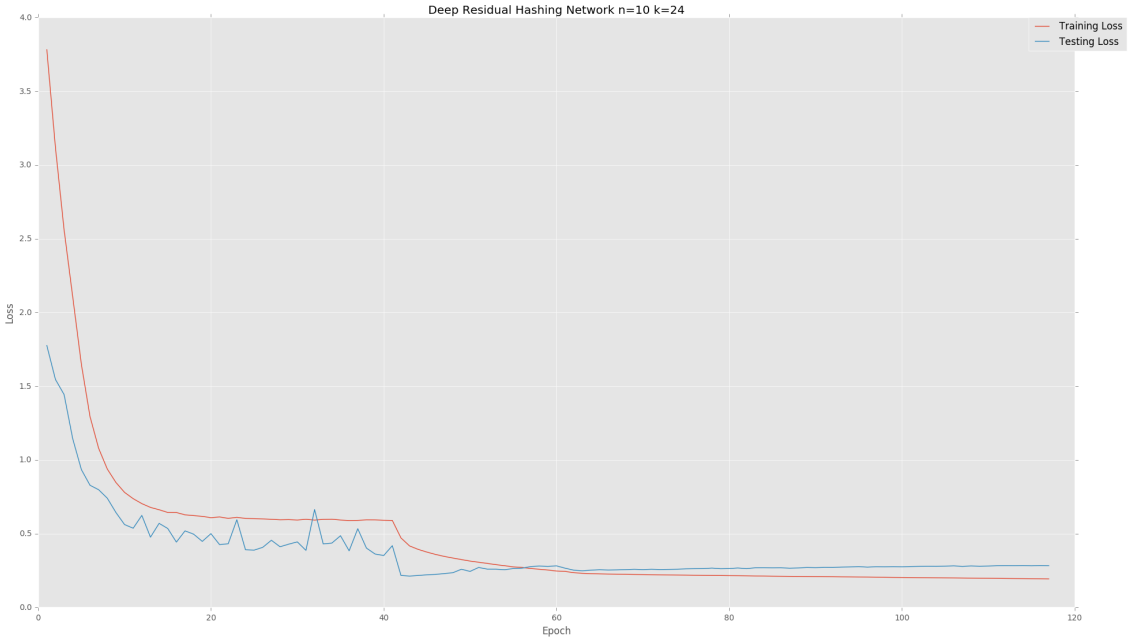

FIGURE 5.3: Loss decrease with respect to number of epoch in DRHN with n=10, k=24.

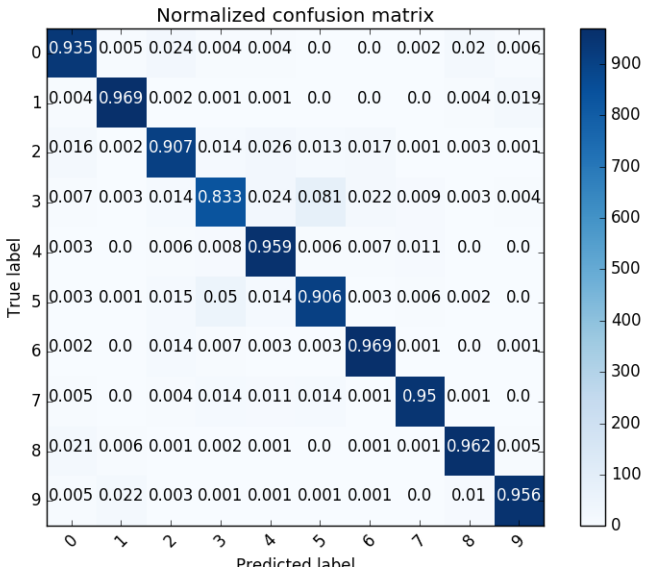

FIGURE 5.4: Image Classification Confusion Matrix for DRHN with n=9.

TABLE 5.1: Accuracy for DRHN with $n = [1 \dots 16]$ and $k = [12, 24, 32, 48, 128]$ on CIFAR-10

Model	12 bits	24 bits	32 bits	48 bits	128 bits
DRHN-1	90.41	90.53	90.74	90.51	90.21
DRHN-2	92.08	91.96	91.80	91.87	91.76
DRHN-3	92.08	92.34	92.41	92.31	92.57
DRHN-4	92.76	92.32	92.80	92.24	92.83
DRHN-5	93.07	92.57	92.65	93.06	93.21
DRHN-6	93.00	93.32	92.75	93.44	92.73
DRHN-7	93.05	93.08	93.21	93.30	93.03
DRHN-8	93.07	93.06	93.50	93.17	92.89
DRHN-9	93.18	93.11	93.46	93.70	93.22
DRHN-10	93.74	93.61	93.75	93.21	93.36
DRHN-11	93.46	92.98	93.46	93.56	93.25
DRHN-12	93.03	93.85	93.61	93.52	93.46
DRHN-13	93.43	93.49	93.45	93.28	93.40
DRHN-14	93.01	93.58	93.82	93.50	93.36
DRHN-15	93.55	93.27	93.22	93.62	93.49
DRHN-16	93.42	93.26	93.26	93.30	93.50

TABLE 5.2: mAP comparison of different hashing methods on CIFAR-10 dataset.

Method	12 bits	32 bits	48 bits
DRHN-15	**92.65**	**92.23**	**92.91**
DLBHC	89.3	89.72	89.73
CNNBH	53.2	61.0	61.7
DHN	55.5	60.3	62.1
DNNH	55.2	55.8	58.1
CNNH+	46.5	52.1	53.2
CNNH	43.9	50.9	52.2
KSH	30.3	34.6	35.6
ITQ-CCA	26.4	28.8	29.5
LSH	12.1	12.0	12.0

Performance of Image Retrieval

The performance of image retrieval can be seen at Figure 5.5[2]. We plot the results of retrieving relevant images using 48 bits binary codes and Hamming distance between the query image q and the ith retrieved image. Our approach achieves better perfomance than other methods (supervised and unsupervised). It obtains a 92.91% precision retrieving 1000 images, which improves by almost 3% the performance compared to [62].

In Table 5.2, we compare the mean average precision (mAP) of some hashing methods at different number of bits when we retrieve the top 1000 images.

Figure 5.7 shows the retrieval results of four classes in CIFAR-10: frog, horse, ship and plane. The relevant images have the same label and similar appearance.

[2]We thank to [62] for the repository with the information available for Figure 5.5 and Table 5.2.

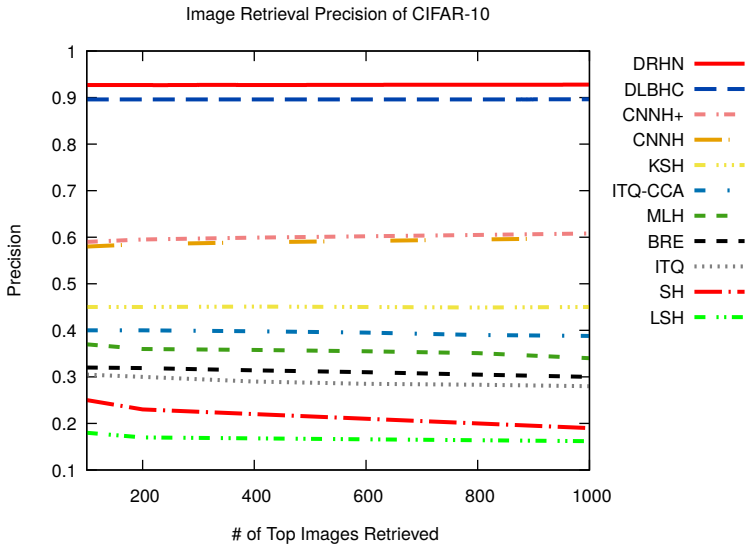

FIGURE 5.5: Image retrieval precision with 48 bits on CIFAR-10 dataset.

TABLE 5.3: mAP comparison of different depth in our method on CIFAR-10 dataset.

Method	12 bits	24 bits	32 bits	48 bits	128 bits
DRHN-16	92.53	92.32	92.15	92.25	92.06
DRHN-15	92.65	92.02	92.23	**92.91**	91.92
DRHN-14	91.78	**92.74**	92.93	92.19	91.65
DRHN-13	92.52	92.58	92.18	92.30	**92.12**
DRHN-12	91.55	92.71	92.78	92.89	91.85
DRHN-11	**92.66**	92.35	92.57	92.57	91.65
DRHN-10	91.93	92.55	**92.94**	91.95	91.69
DRHN-9	91.97	91.92	92.35	92.58	91.46
DRHN-8	92.17	92.03	92.49	91.95	91.22
DRHN-7	91.76	91.74	92.30	92.50	91.30
DRHN-6	91.85	92.49	91.36	92.23	91.05
DRHN-5	91.75	91.21	91.21	91.89	91.65
DRHN-4	91.32	90.83	91.43	90.71	91.04
DRHN-3	90.81	91.04	90.99	90.78	91.04
DRHN-2	90.40	90.22	90.62	90.66	89.63
DRHN-1	87.50	88.01	88.59	88.43	87.83

TABLE 5.4: Average time per epoch in training for CIFAR-10.

Model	Time (s)	Model	Time (s)
DRHN-1	66.41	DRHN-9	858.01
DRHN-2	122.22	DRHN-10	893.73
DRHN-3	184.57	DRHN-11	846.24
DRHN-4	263.72	DRHN-12	925.23
DRHN-5	316.51	DRHN-13	1008.97
DRHN-6	493.20	DRHN-14	1118.85
DRHN-7	976.02	DRHN-15	1213.72
DRHN-8	746.16	DRHN-16	989.50

Something remarkable is the absence of false positive results.

Variation of the Depth in the Model

We variate the depth of the model using n=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] and the number of bits in the binary code using h=[12, 24, 32, 48 and 128]. In Table 5.3, the mAP is shown for the correspondent model and number of bits. Also in Figure 5.6, we present how the precision with respect to the number of retrieved images changes.

The time required to train a model is important, therefore in Table 5.4 the average time per epoch is shown. Also in Table 5.5 the time to generate the binary code for an image is show in miliseconds.

In Figure 5.8 the result of querying three different boat images using DRHN-9 and h=48 and 128 is shown.

In Figure 5.9 the query of one boat image using all the trained models and h = 128 is shown.

5.1.3 Results on CIFAR-100 dataset

To evaluate the scalabity of the proposal with a dataset of more classes we experimented on CIFAR-100. Using the same values for n and l, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] and [12, 24, 32, 48, 128] respectively.

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). Here is the list of classes in the CIFAR-100:

Superclass aquatic mammals, fish, flowers, food containers, fruit and vegetables, household electrical devices, household furniture, insects, large carnivores, large man-made outdoor things, large natural outdoor scenes, large omnivores and herbivores, medium-sized mammals, non-insect invertebrates, people, reptiles, small mammals, trees, vehicles 1, vehicles 2.

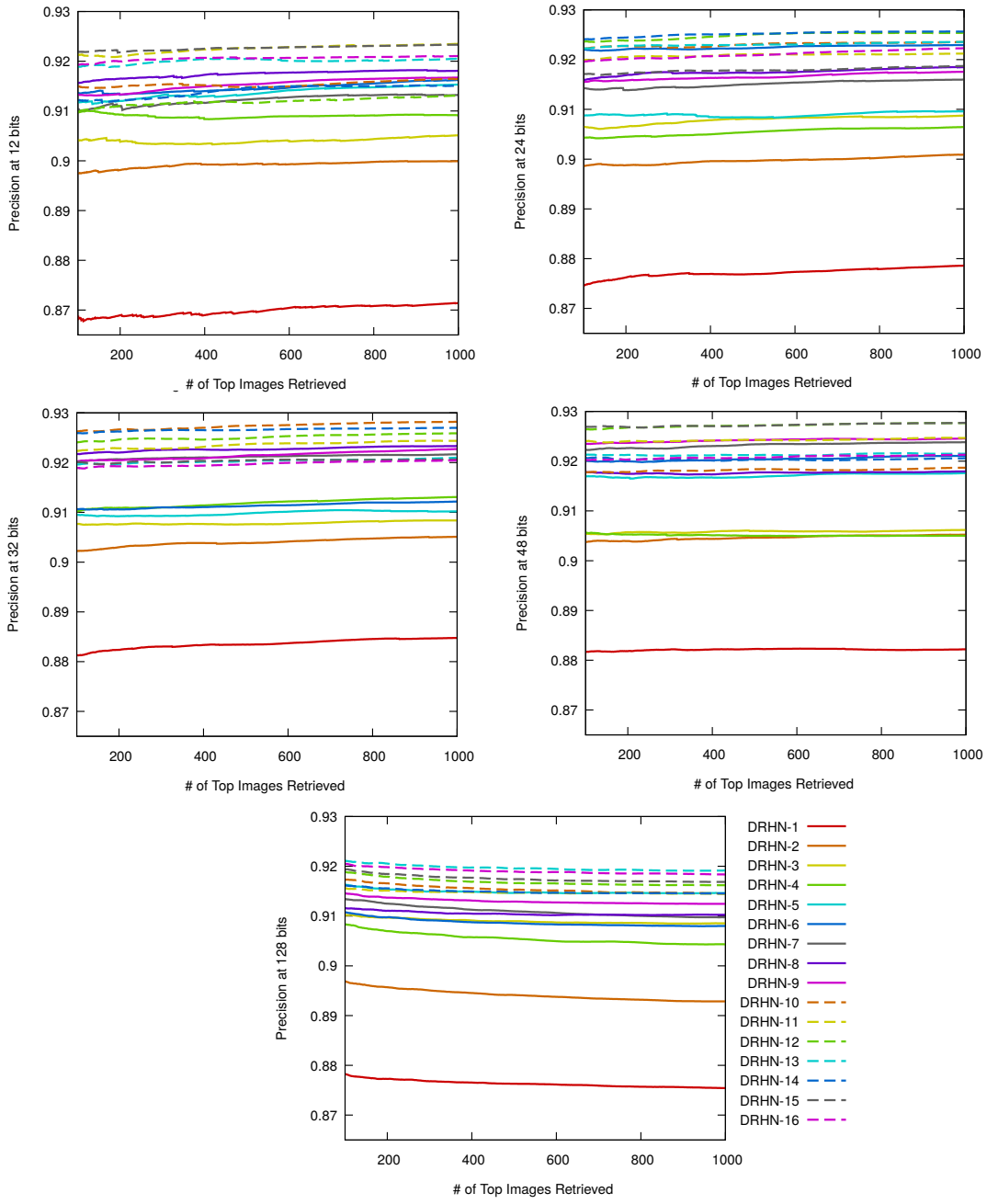

FIGURE 5.6: Image retrieval precision with 12, 24, 32, 48 and 128 bits on CIFAR-10 dataset.

TABLE 5.5: Average time to generate image code (ms) in CIFAR-10

	12 bits	24 bits	32 bits	48 bits	128 bits
DRHN-1	4.742	4.740	4.611	4.681	4.434
DRHN-2	7.459	7.487	7.308	8.554	7.301
DRHN-3	12.065	11.951	12.181	12.759	11.719
DRHN-4	15.585	15.547	15.396	15.466	15.195
DRHN-5	19.171	20.864	19.608	19.375	21.466
DRHN-6	24.742	23.099	23.048	22.829	23.353
DRHN-7	28.367	29.662	28.236	27.878	28.090
DRHN-8	32.157	32.456	32.519	35.739	35.644
DRHN-9	42.584	36.746	36.796	37.188	37.602
DRHN-10	41.369	41.465	39.899	40.693	40.899
DRHN-11	45.037	45.581	45.704	45.617	47.807
DRHN-12	45.984	46.679	47	46.013	46.119
DRHN-13	50.138	50.132	50.546	50.083	50.481
DRHN-14	55.317	55.431	55.111	55.547	55.202
DRHN-15	60.405	60.070	60.767	61.635	60.037
DRHN-16	66.277	65.623	65.376	67.263	67.551

Classes beaver, dolphin, otter, seal, whale,aquarium fish, flatfish, ray, shark, trout,orchids, poppies, roses, sunflowers, tulips, bottles, bowls, cans, cups, plates, apples, mushrooms, oranges, pears, sweet peppers, clock, computer keyboard, lamp, telephone, television, bed, chair, couch, table, wardrobe, bee, beetle, butterfly, caterpillar, cockroach, bear, leopard, lion, tiger, wolf, bridge, castle, house, road, skyscraper, cloud, forest, mountain, plain, sea, camel, cattle, chimpanzee, elephant, kangaroo, fox, porcupine, possum, raccoon, skunk, crab, lobster, snail, spider, worm people baby, boy, girl, man, woman, crocodile, dinosaur, lizard, snake, turtle, hamster, mouse, rabbit, shrew, squirrel, maple, oak, palm, pine, willow, bicycle, bus, motorcycle, pickup truck, train, lawn-mower, rocket, streetcar, tank, tractor

After 43 epochs the model achieved $[57.41 \approx 69.69]\%$ of accuracy in classification task with the values of n. The accuracy for classification is shown in Table 5.6 and the mAP for image retrieval is shown in Table 5.7. Finally, variation of precision with respect to number of images retrieved is shown in Figure 5.10.

TABLE 5.6: Accuracy for DRHN with $n = [1 \dots 16]$ and $k = [12, 24, 32, 48, 128]$ on CIFAR-100

Model	12 bits	24 bits	32 bits	48 bits	128 bits
DRHN-1	56.32	61.22	61.17	62.24	63.42
DRHN-2	61.48	63.92	64.22	65.15	65.78
DRHN-3	62.87	64.63	65.59	66.66	67.59
DRHN-4	63.24	65.67	66.09	67.30	68.60
DRHN-5	63.77	65.94	66.93	67.52	68.71
DRHN-6	64.32	66.88	67.33	68.56	69.02
DRHN-7	64.27	66.76	67.96	68.37	69.28
DRHN-8	65.01	67.10	67.87	68.03	69.18
DRHN-9	64.28	67.03	68.24	68.60	70.03
DRHN-10	66.29	67.43	68.29	68.25	69.58
DRHN-11	63.92	67.77	67.68	68.91	70.43
DRHN-12	64.19	67.14	68.38	68.82	69.46
DRHN-13	65.30	67.13	68.14	69.62	70.17
DRHN-14	65.55	67.95	68.30	69.75	70.24
DRHN-15	67.15	67.40	67.63	69.52	69.78
DRHN-16	65.89	67.32	68.31	68.89	70.69

TABLE 5.7: mAP comparison of different depth in our method on CIFAR-100 dataset.

Method	12 bits	24 bits	32 bits	48 bits	128 bits
DRHN-16	61.89	60.61	62.28	62.37	**63.63**
DRHN-15	**63.51**	60.82	60.95	**63.77**	62.10
DRHN-14	62.14	**62.82**	61.63	63.46	62.14
DRHN-13	62.15	60.88	60.28	63.45	61.55
DRHN-12	60.73	61.75	**62.57**	63.39	61.67
DRHN-11	59.78	60.96	60.03	61.80	63.20
DRHN-10	62.56	59.44	62.33	63.03	62.38
DRHN-9	59.00	60.98	60.86	61.77	62.28
DRHN-8	61.41	60.90	61.50	60.20	61.57
DRHN-7	59.50	59.70	62.47	62.01	59.88
DRHN-6	59.63	59.71	61.40	61.91	60.64
DRHN-5	58.20	58.26	58.82	60.48	58.64
DRHN-4	57.49	58.20	58.62	59.32	58.69
DRHN-3	56.73	56.34	56.73	57.64	56.60
DRHN-2	53.76	55.08	55.78	55.97	52.29
DRHN-1	45.71	51.09	50.01	51.01	49.33

FIGURE 5.7: Top 10 retrieved images from CIFAR-10 using DRHN-9 with different bit numbers.

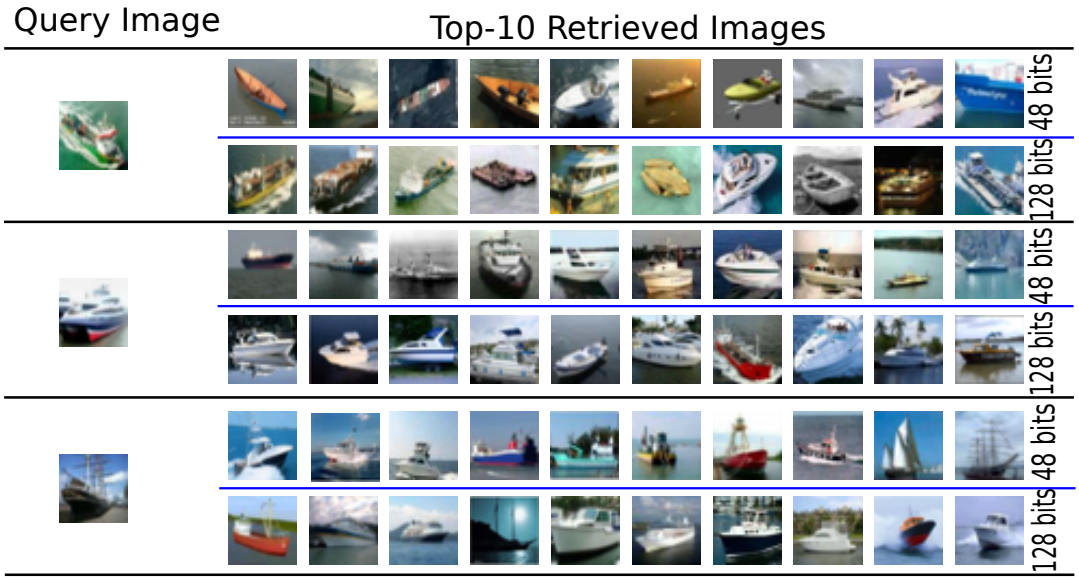

FIGURE 5.8: Top 10 retrieved images for boat images in CIFAR-10 using DRHN-9 and h=[48, 128].

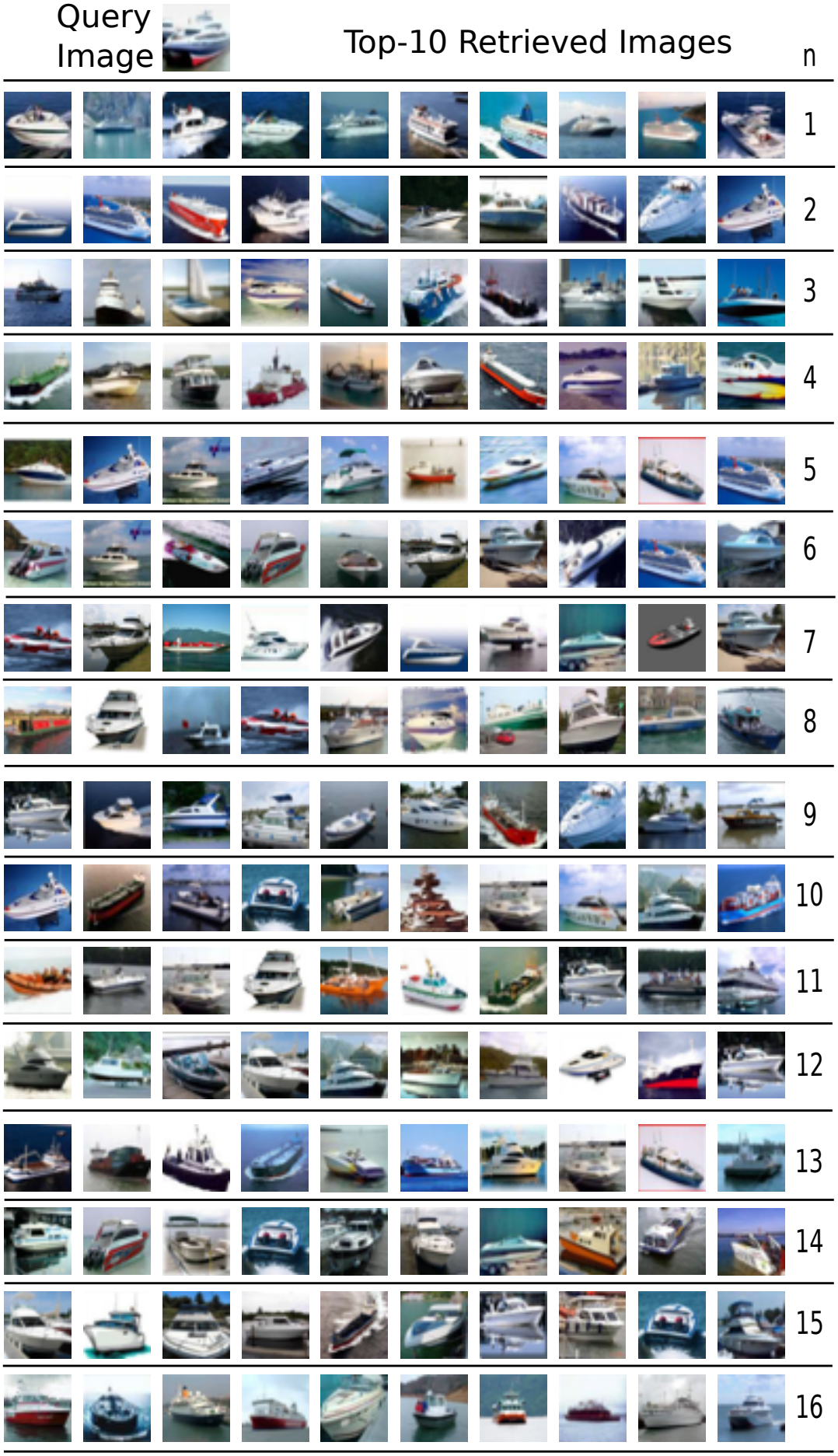

FIGURE 5.9: Top 10 retrieved images for a boat image query in CIFAR-10 using DRHN-[1 . . . 16] and h=[128].

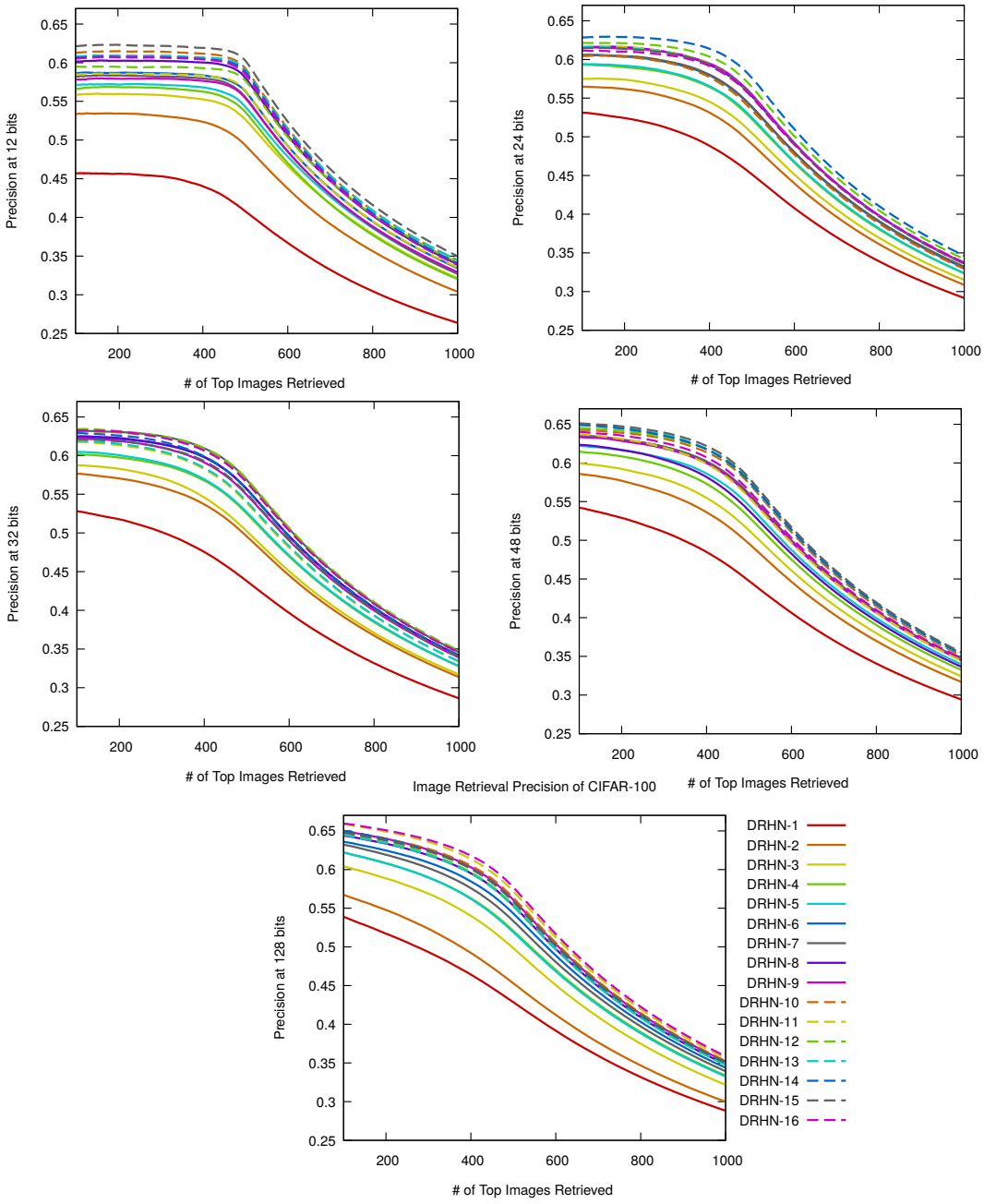

FIGURE 5.10: Image retrieval precision with 12, 24, 32, 48 and 128 bits on CIFAR-100 dataset.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed a deep model of CNN based on the work of [55] to generate hashing codes in a supervised manner, allowing an efficient search of images based on their content. This method takes fully advantage of data labels (instead of requiring a triplet of images as other methods do) and raw data thanks to the properties of CNN. Experimental results show that we can outperform several state-of-the-art results on image retrieval on a popular dataset (CIFAR-10).

Furthermore we observe that the mAP in retrieval task increases about model depth. Having a short binary code leads to similar results in different image queries of same class however with the appropiate combination of model depth and binary code length we can obtain a good visual result.

6.2 Future Work

Train the model for retrieval task in more complex image datasets (with hundreds or thousands of classes) to measure the proposal's scalability.

Experiment with supervised methods that are not based on deep learning, because of time and compute consumption.

6.3 Publications

One contribution to ICANN 2017 (26 th International Conference on Artificial Neural Networks) entitled *Deep Residual Hashing Network for Image Retrieval*. It is included in the Springer book *Artificial Neural Networks and Machine Learning - ICANN 2017*, which belong to the series *Lecture Notes in Computer Science* (LNCS).

Bibliography

[1] H. Tamura and N. Yokoya, "Image database systems: A survey", *Pattern recognition*, vol. 17, no. 1, pp. 29–43, 1984.

[2] L. Zheng, Y. Yang, and Q. Tian, "Sift meets cnn: A decade survey of instance retrieval", *ArXiv preprint arXiv:1608.01807*, 2016.

[3] J. Sivic, A. Zisserman, *et al.*, "Video google: A text retrieval approach to object matching in videos.", in *Iccv*, vol. 2, 2003, pp. 1470–1477.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] R. da Silva Torres and A. X. Falcao, "Content-based image retrieval: Theory and applications.", *RITA*, vol. 13, no. 2, pp. 161–185, 2006.

[6] H. Müller, N. Michoux, D. Bandon, and A. Geissbuhler, "A review of content-based image retrieval systems in medical applications—clinical benefits and future directions", *International journal of medical informatics*, vol. 73, no. 1, pp. 1–23, 2004.

[7] R. d. S. Torres, C. B. Medeiros, M. A. Gonçcalves, and E. A. Fox, "A digital library framework for biodiversity information systems", *International Journal on Digital Libraries*, vol. 6, no. 1, pp. 3–17, 2006.

[8] J.-S. Hong, H.-Y. Chen, and J. Hsiang, "A digital museum of taiwanese butterflies", in *Proceedings of the fifth ACM conference on Digital libraries*, ACM, 2000, pp. 260–261.

[9] B. Zhu, M. Ramsey, and H. Chen, "Creating a large-scale content-based airphoto image digital library", *IEEE Transactions on Image Processing*, vol. 9, no. 1, pp. 163–167, 2000.

[10] S. A. Gulhane, "Content based image retrieval from forensic image databases", *International Journal of engineering Research and Applications*, vol. 1, no. 5, pp. 66–70,

[11] A. Khokher and R. Talwar, "Content-based image retrieval: Feature extraction techniques and applications", in *International Conference on Recent Advances and Future Trends in Information Technology (iRAFIT2012)*, 2012, pp. 9–14.

[12] P. M. Ferreira, M. A. Figueiredo, and P. M. Aguiar, "Content-based image classification: A non-parametric approach", *Quer y Simil arity Featu re Supe r*,

[13] N. P. Ramaiah, E. P. Ijjina, and C. K. Mohan, "Illumination invariant face recognition using convolutional neural networks", in *Signal Processing, Informatics, Communication and Energy Systems (SPICES), 2015 IEEE International Conference on*, IEEE, 2015, pp. 1–4.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *European Conference on Computer Vision*, Springer, 2014, pp. 818–833.

[16] S. Conjeti, A. G. Roy, A. Katouzian, and N. Navab, "Deep residual hashing", *ArXiv preprint arXiv:1612.05400*, 2016.

[17] T Karthikeyan, P Manikandaprabhu, and S Nithya, "A survey on text and content based image retrieval system for image mining", *International Journal of Engineering*, vol. 3, no. 3, 2014.

[18] H.-W. Yoo, D.-S. Jang, S.-H. Jung, J.-H. Park, and K.-S. Song, "Visual information retrieval system via content-based approach", *Pattern Recognition*, vol. 35, no. 3, pp. 749–769, 2002.

[19] R. C. Veltkamp and M. Tanase, "Content-based image retrieval systems: A survey", 2001.

[20] T. Tuytelaars, K. Mikolajczyk, *et al.*, "Local invariant feature detectors: A survey", *Foundations and trends® in computer graphics and vision*, vol. 3, no. 3, pp. 177–280, 2008.

[21] A. I. Awad and M. Hassaballah, *Image Feature Detectors and Descriptors: Foundations and Applications*. Springer, 2016, vol. 630.

[22] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes", *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 9, pp. 1704–1716, 2012.

[23] Y. Uchida, "Local feature detectors, descriptors, and image representations: A survey", *ArXiv preprint arXiv:1607.08368*, 2016.

[24] C. Harris and M. Stephens, "A combined corner and edge detector.", in *Alvey vision conference*, Manchester, UK, vol. 15, 1988, pp. 10–5244.

[25] K. Mikolajczyk and C. Schmid, "Indexing based on scale invariant interest points", in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, IEEE, vol. 1, 2001, pp. 525–531.

[26] ——, "An affine invariant interest point detector", *Computer Vision—ECCV 2002*, pp. 128–142, 2002.

[27] P. R. Beaudet, "Rotationally invariant image operators", in *Proc. 4th Int. Joint Conf. Pattern Recog, Tokyo, Japan, 1978*, 1978.

[28] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A comparison of affine region detectors", *International journal of computer vision*, vol. 65, no. 1-2, pp. 43–72, 2005.

[29] T. Lindeberg, "Feature detection with automatic scale selection", *International journal of computer vision*, vol. 30, no. 2, pp. 79–116, 1998.

[30] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features", *Computer vision–ECCV 2006*, pp. 404–417, 2006.

[31] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking", in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, IEEE, vol. 2, 2005, pp. 1508–1515.

[32] D. G. Lowe, "Object recognition from local scale-invariant features", in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Ieee, vol. 2, 1999, pp. 1150–1157.

[33] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features", *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[34] C.-F. Tsai, "Bag-of-words representation in image annotation: A review", *ISRN Artificial Intelligence*, vol. 2012, 2012.

[35] T. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers", in *Advances in neural information processing systems*, 1999, pp. 487–493.

[36] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification", *Computer Vision–ECCV 2010*, pp. 143–156, 2010.

[37] S.-I. Amari, "Natural gradient works efficiently in learning", *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[38] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation", in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, IEEE, 2010, pp. 3304–3311.

[39] G. Amato, C. Gennaro, and P. Savino, "Mi-file: Using inverted files for scalable approximate similarity search", *Multimedia tools and applications*, vol. 71, no. 3, pp. 1333–1362, 2014.

[40] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998, ISBN: 0132733501.

[41] G. Gwardys. (2016). Convolutional neural networks backpropagation: From intuition to derivation, [Online]. Available: https://grzegorzgwardys.wordpress.com/2016/04/22/8/.

[42] H. Kuwajima. (2014). Memo: Backpropagation in convolutional neural network, [Online]. Available: http://kawahara.ca/what-is-the-derivative-of-relu/.

[43] J Kawahara. (2016). What is the derivative of relu?, [Online]. Available: http://es.slideshare.net/kuwajima/cnnbp.

[44] boris.ginzburg@intel.com. (2014). Lecture 3: Cnn: Back-propagation, [Online]. Available: http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/pdf_lectures/.

[45] L. lab. (). Convolutional neural networks (lenet), [Online]. Available: http://deeplearning.net/tutorial/lenet.html.

[46] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio, "Learned-norm pooling for deep feedforward and recurrent neural networks", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2014, pp. 530–546.

[47] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks", in *International Conference on Rough Sets and Knowledge Technology*, Springer, 2014, pp. 364–375.

[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *ArXiv preprint arXiv:1502.03167*, 2015.

[49] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.", *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[51] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[52] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net", *ArXiv preprint arXiv:1412.6806*, 2014.

[53] M. Lin, Q. Chen, and S. Yan, "Network in network", *ArXiv preprint arXiv:1312.4400*, 2013.

[54] B. Graham, "Spatially-sparse convolutional neural networks", *ArXiv preprint arXiv:1409.6070*, 2014.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[56] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3270–3278.

[57] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings", in *Advances in neural information processing systems*, 2009, pp. 1042–1050.

[58] M. Norouzi and D. M. Blei, "Minimal loss hashing for compact binary codes", in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 353–360.

[59] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels", in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, IEEE, 2012, pp. 2074–2081.

[60] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning.", in *AAAI*, vol. 1, 2014, p. 2.

[61] J. Guo and J. Li, "Cnn based hashing for image retrieval", *ArXiv preprint arXiv:1509.01354*, 2015.

[62] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen, "Deep learning of binary hash codes for fast image retrieval", in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 27–35.

[63] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing", in *Advances in neural information processing systems*, 2009, pp. 1753–1760.

[64] A. Gionis, P. Indyk, R. Motwani, *et al.*, "Similarity search in high dimensions via hashing", in *VLDB*, vol. 99, 1999, pp. 518–529.

[65] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[66] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope", *International journal of computer vision*, vol. 42, no. 3, pp. 145–175, 2001.

[67] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *ArXiv preprint arXiv:1409.1556*, 2014.

[68] J. Wang, T. Zhang, N. Sebe, H. T. Shen, *et al.*, "A survey on learning to hash", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[69] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval.", in *AAAI*, 2016, pp. 2415–2421.

[70] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images", 2009.

[71] S. Dieleman *et al.*, *Lasagne: First release.* Aug. 2015. DOI: 10.5281/zenodo.27878. [Online]. Available: http://dx.doi.org/10.5281/zenodo.27878.

[72] Theano Development Team, "Theano: a Python framework for fast computation of mathematical expressions", *ArXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688.

[73] J. Deng, A. C. Berg, and L. Fei-Fei, "Hierarchical semantic indexing for large scale image retrieval", in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, IEEE, 2011, pp. 785–792.

[74] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years", *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 12, pp. 1349–1380, 2000.