CNN features for Reverse Image Search

Mathieu Gaillard [1], **Előd Egyed-Zsigmond** [1], **Michael Granitzer** [2]

1. *Université de Lyon*
   LIRIS UMR5205 CNRS / INSA Lyon
   69621 Villeurbanne Cedex, France
   {mathieu.gaillard,elod.egyed-zsigmond}@insa-lyon.fr

2. *University of Passau*
   Passau, Germany
   michael.granitzer@uni-passau.de

ABSTRACT. *Activations of units within top layers of convolutional neural networks (CNN features) are known to be good descriptors for image retrieval. In this paper, we investigate the use of CNN features for Reverse Image Search. We especially evaluate the robustness of these features against common modifications. In a first part, we present a benchmark to evaluate the retrieval performances of Reverse Image Search systems. To get a baseline, we evaluate well-established methods: Perceptual Hash Functions. In a second part, we further evaluate the retrieval performances of CNN features. We establish that CNN features perform better than Perceptual Hash Functions, even if neural networks are trained on unrelated classification tasks. CNN features are more robust against rotation and cropping. Finally, we give a list of layers from different neural networks that are remarkably good descriptors when used with a cosine distance.*

RÉSUMÉ. *Les activations des unités des couches supérieures des réseaux de neurones à convolution (caractéristiques CNN) se sont révélées être de bons descripteurs pour la recherche d'image. Dans ce papier, nous étudions les caractéristiques CNN pour la recherche d'image inversée. Nous évaluons la robustesse de ces caractéristiques contre des modifications courantes. Dans une première partie, nous présentons un benchmark de systèmes de recherche d'image inversée. Pour référence, nous évaluons des méthodes traditionnelles : comme les fonctions de hachage perceptuel. Dans une seconde partie, nous évaluons les caractéristiques CNN. Nous montrons que les caractéristiques CNN sont plus performantes que les fonctions de hachage perceptuel, même dans le cas de réseaux de neurones entrainés sur des tâches de classification non liées. Les caractéristiques CNN sont plus robustes contre les rotations et le rognage. Finalement, nous dressons une liste de couches de différents réseaux de neurones qui sont de bons descripteurs lorsqu'ils sont comparés avec une distance cosinus.*

KEYWORDS: *Reverse Image Search, Benchmark, Convolutional Neural Networks*

MOTS-CLÉS : *Reverse Image Search, Benchmark, Convolutional Neural Networks*

DOI:10.3166/RIA.21.1-31

1. Introduction

As it is easy for people to take, store and share pictures, a massive number of images appears every day on the internet. To enable indexing, searching, processing and organizing such a quickly growing volume of images, one has to develop new efficient methods capable of dealing with large scale data.

Lately, several systems have been launched and allow the user to find the original of an image in a large image collection given a slightly modified version of it: for example, Google Images, TinEye and Microsoft PhotoDNA. Those, called Reverse Image Search (RIS) systems, are extensively studied because of their applications in the context of intellectual property and crime prevention.

Basically, in image retrieval, one extracts high-dimensional feature vectors from images on which a nearest neighbor search is then performed to query images by similarity. With a very large collection of images, representations should be as small as possible to reduce storage costs. Additionally, a data structure should allow for sublinear-time search in the database of image representations. Traditionally, methods are based on image descriptors such as color histograms, CEDD, SIFT, GIST or perceptual hashing functions.

The idea of perceptual hashing is to map similar (resp. dissimilar) inputs into similar (resp. dissimilar) binary codes according to a selected distance. This approach works very well because binary codes are compact and easy to handle on computers. Furthermore, recent work showed that it is possible to search binary codes in sublinear time for uniformly distributed codes. However, as shown in this paper, state of the art perceptual hashing functions are not robust against cropping and rotations. Thus, the design of robust reverse images search system is still an open question.

Lately, great advances have been made in computer vision, especially in the field of deep learning. Whereas traditionally, the previous works in computer vision were based on hand-engineered features, convolutional neural networks can automatically learn the features to extract from a dataset of images. Moreover, the computational power of the latest GPUs allows learning on huge datasets, which means that learnt features are of a very high quality.

The key observation is that the activations within the top layers of a large convolutional neural network (CNN) provide a high-quality descriptor of the visual content of an image. Even when the convolutional neural network has been trained on an unrelated classification task, its retrieval performance is still very competitive. These features can be used in the context of Reverse Image Search.

The main contribution of this article is the comparison of the retrieval performances of: firstly, traditional perceptual hash functions (DCT, MH and RH), and secondly existing convolutional neural networks trained on ImageNet (VGG, InceptionV3, ResNet50).

This paper is organized as follows:

Section 2 – Reverse Image Search: presents the general concept of Reverse Image Search. This section explains how such a system usually works and what it is commonly used for.

Section 3 – Perceptual hashing: presents the general concept of Perceptual hashing, which is one well-established solution to build a Reverse Image Search system.

Section 4 – Convolutional Neural Networks: presents the concept of convolutional neural networks and shows that they can extract excellent image representations. Those can be used to build a Reverse Image Search system.

Section 5 – Benchmarking: presents our benchmark to compare the robustness of Reverse Image Search techniques. We also present the results of the benchmark obtained with the three perceptual hashing functions presented in section 3.

Section 6 – CNN Features robustness: presents our main contribution. We implemented the benchmark presented in section 5 with features extracted using already existing convolutional neural networks. We then compare their retrieval performances to those of the perceptual hashing functions presented in section 3.

2. Reverse Image Search

This section presents the general concept of Reverse Image Search and explains how such a system usually works and what it is commonly used for. It generally consists of a way of representing images and a distance between these representations. A query consists in a nearest neighbor search using this distance. We briefly give examples of algorithms to extract image representations and distances that can be used to compare them.

2.1. Definition

To better understand the definition and context of the Reverse Image Search problem, we first give the definition of two more general problems: Image Retrieval and Content-based Image Retrieval.

An image retrieval system is a computer system for browsing, searching and retrieving images from a large database of digital images (Baeza-Yates *et al.*, 1999). Examples of such a system are *Google Image*[1] and *Bing Image*[2], where the user writes a text query and have a list of images in return.

Content-based image retrieval (CBIR) is the application of computer vision techniques to the image retrieval problem. "Content-based" means that the search analyses the actual content of the image rather than the metadata such as keywords, tags, or descriptions associated with the image. (Datta *et al.*, 2008) An example of such a system is *Google Image*, where the user can perform a search by image, either by URL or by uploading a file. If one submits an image of a cat, in return we will get images of visually similar cats, but not necessarily the exact same cat.

Reverse image search (RIS) is a content-based image retrieval (CBIR) query technique. The aim of a RIS system is to find the original of an image in a large image collection given a slightly modified version of it. An example of such a system is *TinEye*[3], where the user can submit an image and find out where this exact image appears on internet.

2.2. Usage

The main applications are listed in the FAQ[4] of *TinEye*.

– Find out where an image came from, or get more information about it
– Identify duplicate images

1. www.google.com
2. www.bing.com
3. www.tineye.com
4. www.tineye.com/faq

– Find a higher resolution version of an image
– Locate web pages that make use of an image you have created
– Discover modified or edited versions of an image
– Show that the information provided with an image is false

Following are more specific use cases: A dating site can verify that a profile is authentic. An insurance company can detect fraud. Trademark offices can identify infringing trademarks. A museum can provide a mobile application on which the user can have additional details about a painting. A brand can replace QR codes and connect a printed catalog to an ecommerce platform.

2.3. General framework

Most approaches to Reverse Image Search share a similar pattern. It consists of: firstly, a way of representing an image, and secondly, a distance (or similarity) measure between two representations. As shown in figure 1, RIS systems usually work in two phases: indexing and searching. During the indexing phase, representations of all the images in a collection are extracted and added to a database. The images are not necessarily stored in the database; this reduces its size. Afterward during the searching phase, a query image is presented to the system and its representation is extracted. A nearest neighbor search is then performed with the query representation, using the previously defined distance measure, to search for similar images in the database.

Reverse Image Search with large collections of images imposes two challenging constraints on the methods used. Firstly, for each image, only a small amount of data can be stored; secondly, queries evaluation cost must be very cheap.

2.4. Image representation

The image representation is usually a p-dimensional vector of numerical features or a q-bit binary code. The feature vectors should be compact, in order to store millions of them in a database. Many algorithms are able to extract feature vectors from images based on color, texture, shape. We can classify features in two categories: low-level features, which are minor details of the image that are detected by simple algorithms: edges, corners, and so on, and high-level features, which carry more semantic and thus are better understandable by humans: objects, actions, and so on. High-level features are difficult to extract for computers because it is hard to understand the semantic of an image, this is currently an active research field. In the following part, five well-established methods to extract low-level features from images are described.

A very simple approach is to compute the color histogram of an image. The image representation in this case is an integer vector of N-bins. If there are too many different colors in the image, it is possible to subsample the color space. Comparing two histograms is similar to comparing two discrete probability distributions. It is commonly done using one of the following distance: the population Pearson correla-

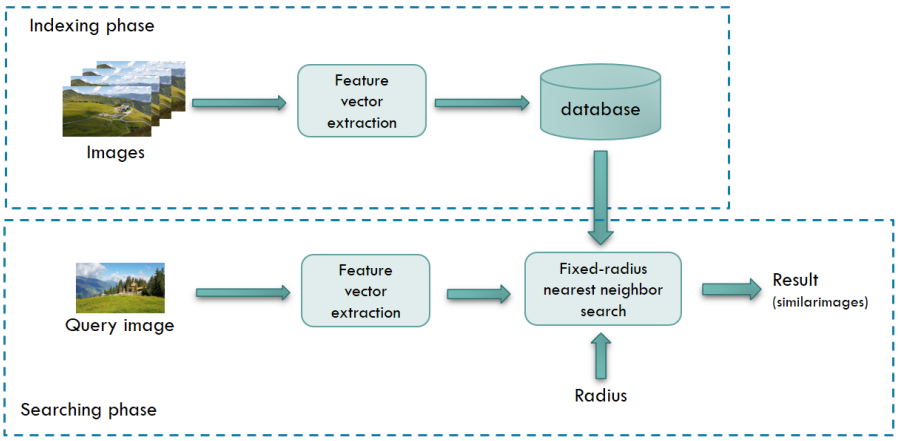

Figure 1. General framework for Reverse Image Search with fixed-radius nearest neighbor search.

tion coefficient, the Chi-Square distance, the Bhattacharyya distance (Bhattacharyya, 1943), the Wasserstein distance, also known as the Earth mover's distance (Rubner *et al.*, 1998).

Color and Edge Directivity Descriptor (CEDD) (Chatzichristofis, Boutalis, 2008) combines, in one histogram, color and texture information. The feature vector size is up to 54 bytes per image. The similarity between two CEDD histograms is measured with Tanimoto coefficient.

Scale-invariant Feature Transform (SIFT) (Lowe, 2004) extracts distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a substantial range of affine distortion. SIFT detects keypoints in the image and compute small descriptor for each of them. The number of keypoints can vary and this is why a feature vector extracted with SIFT does not have a fixed length. To compare two images, their keypoints are matched by identifying their nearest neighbors, which can be very costly dependending on the number of keypoints.

The GIST descriptor was initially proposed in (Oliva, Torralba, 2001). It is based on Gabor filters, which measure the orientations and spatial frequencies, and describes a scene with a set of perceptual dimensions (naturalness, openness, roughness, expansion, ruggedness). The GIST description is a feature vector of dimension 960, which can be compared with an Euclidean distance (Douze *et al.*, 2009).

In section 3, we give a detailed presentation of perceptual hashing, whose idea is to map similar (resp. dissimilar) inputs into similar (resp. dissimilar) binary codes according to a selected distance.

2.5. Distance

To compare two image representations one has to choose a distance or similarity function. Depending on the image representation, many distances are possible. In this subsection, we present two distances between p-dimensional vectors and one distance between binary codes. Those are used later in this document.

2.5.1. Distance between two p-dimensional vectors

If the image representation is a p-dimensional feature vector, following are two distances or similarities between $x \in \mathbb{R}^p$ and $y \in \mathbb{R}^p$.

Minkowski distance, also known as the p-norm distance, is a generalization of Manhattan and Euclidean distances. Euclidean distance can be interpreted as the ordinary straight-line distance between two points in Euclidean space.

$$d_{minkowski}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

When $p = 1$, the Minkowski distance is equivalent to the Manhattan distance. When $p = 2$, it is equivalent to the Euclidean distance. If the function is used to rank vectors according to their distance we can save time by not computing the nth root because this function is monotonically increasing.

Cosine similarity measures the cosine of the angle between two vectors. The cosine similarity is equal to 1 if the angle between vectors is 0. It is equal to 0 if the angle between vectors is $\pi/2$.

$$s_{cosine}(x, y) = \cos \theta(x, y) = \frac{x \cdot y}{\|x\|_2 \|y\|_2}$$

$$d_{cosine}(x, y) = \frac{\cos^{-1} s_{cosine}(x, y)}{\pi}$$

For example, the cosine similarity is used in information retrieval. In this field, documents are often represented as vectors containing the number of occurrences of terms. The cosine similarity is meaningful to measure the similarity of two documents regarding their subjects. In facts, what is important is not the number of times a term appears, but whether it appears or not.

Cosine similarity is related to Euclidean distance if the vectors are normalized to unit length.

$$\|x-y\|^2 = (x-y)\cdot(x-y) = x\cdot x+y\cdot y-2x\cdot y = \|x\|^2+\|y\|^2-2(\|x\|^2\|y\|^2\cos\theta(x,y))$$

Because $\|x\| = \|y\| = 1$, we can conclude:

$$\|x-y\|^2 = 2(1-\cos\theta(x,y))$$

2.5.2. Distance between two binary codes

If the image representation is a q-bit binary code, following is a distance between $a \in [0,1]^q$ and $b \in [0,1]^q$.

Hamming distance is the number of positions at which the bits are different. It measures the edit distance between two binary codes if the only allowed operation to transform the code is to flip a bit.

$$d_{hamming}(a,b) = \sum_{i=1}^{n}(a_i \oplus b_i)$$

On computers, this distance is straightforward to compute because it is the number of ones (population count) in the XOR of two binary codes. Since the population count and XOR are two basic operations in modern CPUs, the computation of Hamming distance is very efficient.

2.6. Nearest Neighbor Search

Nearest Neighbor Search (NNS) is the problem of finding in a database all the items whose distances to a query item are the smallest. Two variants are interesting for CBIR and RIS: k-nearest neighbors search and fixed-radius near neighbors.

K-nearest neighbors search aims to find the k nearest neighbors of a given query point. Usually the results are ordered by decreasing similarity (i.e. increasing distance). When applied to CBIR it is interesting in order to explore an image collection, because if the first results are not interesting, one can just look at the following results.

Fixed-radius near neighbors aims to find all points that are within a radius of a given query point. Even if it is possible to sort the results according to the distance, the results should be considered as a set of unranked images. When applied to CBIR, this method is interesting for identifying content because only relevant images are returned, thus the result is composed of a varying number of images. The determination of an adequate radius, in accordance with the actual application, is critical. Information retrieval research has shown that precision and recall follow an inverse relationship (Datta *et al.*, 2008). If the radius is too low, the precision is better at the expense of the recall, and vice versa. Depending on the application, one can want to

favor precision, to authenticate content because there is fewer false-positives, or recall, to identify content because the user can deal with false positives.

Suppose our dataset is composed of p-dimensional feature vectors in a Euclidean space $D \equiv \{x_i\}_{i=1}^{n}$ where $x_i \in \mathbb{R}^p$. Let $z \in \mathbb{R}^p$ be a query feature vector, the one-nearest neighbor of the query z is defined as:

$$KNN_1(z) = \underset{1 \leq i \leq n}{\arg\min} \|z - x_i\|^2$$

Let $r \in \mathbb{R}^+$ be a radius, the fixed-radius nearest neighbors of the query z are defined as:

$$FRNN_r(z) = \{y \in D \mid \|y - z\| \leq r\}$$

For p-dimensional feature vectors, there exist algorithms for exact nearest neighbor search such as the k-d tree, R-tree or MVP tree. But unfortunately, because of the curse of dimensionality, they are not efficient for high dimensional data (more than 20 dimensions). For this reason, Approximate Nearest Neighbor Search (ANNS) is gaining more interest because it allows for faster searching time with only small actual errors. In any case, we can refine a list of approximate nearest neighbors by pruning the items with the actual distance on the original features. The Locality Sensitive Hashing (LSH) framework is one approach to Approximate Nearest Neighbor Search and is detailed later in this part.

3. Perceptual Hashing

This section presents the general concept of Perceptual Hashing, which is one well-established solution to build a Reverse Image Search system. Firstly, we give a detailed definition of Perceptual Hashing. Secondly, we show that it can be used in other contexts, such as digital watermarking. Finally, we give three examples of traditional perceptual hashing functions: The Discrete Cosine Transform (DCT) based perceptual hash function, the Marr-Hildreth (MH) based perceptual hash function and the Radial Variance (RV) based perceptual hash function.

3.1. Definition

A perceptual hash function is a type of hash function that has the property to return analogous outputs if inputs are similar. This allows one to make meaningful comparisons between hashes in order to measure the similarity between the source data. The definition of a hash function according to (Menezes *et al.*, 1996) is:

> A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values.

In the case of a perceptual hash function, four more properties should be present according to (Zauner, 2010):

> Let H denote a hash function which takes one media object (e.g. an image) as input and produces a binary string of length l. Let x denote a particular media object and $\widetilde{x}$ denote a modified version of this media object which is "perceptually similar" to x. Let y denote a media object that is "perceptually different" from x. Let x' and y' denote hash values. $0,1^l$ represents binary strings of length l. Then the four desirable properties of a perceptual hash are identified as follows.
>
> A uniform distribution of hash-values; the hash-value should be unpredictable.
>
> $$\mathbb{P}(H(x) = x') \approx \frac{1}{2^l} \quad \forall x' \in \{0,1\}^l$$
>
> Pairwise independence for perceptually different media objects.
>
> $$\mathbb{P}(H(x) = x' | H(y) = y') \approx \mathbb{P}(H(x) = x') \quad \forall x', y' \in \{0,1\}^l$$
>
> Invariance for perceptually similar media objects.
>
> $$\mathbb{P}(H(x) = H(\widetilde{x})) \approx 1$$

Distinction of perceptually different media objects. It should be impossible to construct a perceptually different media object that has the same hash-value as another media object.

$$\mathbb{P}(H(x) = H(y)) \approx 0$$

Most of the time, to achieve these properties, the perceptual hash function extracts some features of media objects that are invariant under slight modifications. For example, knowing how a compression algorithm works, it is possible to find some invariant features and then design a perceptual hash based on them. Some examples of perceptual hash functions for images are detailed later in this section.

3.2. Usage

Although Perceptual hash functions can be used in the context of CBIR or RIS for representing images, they are also useful in other contexts. Due to the properties inherited from the hash functions they can be used to authenticate media objects even if they are slightly modified, for example lossy compressed. The image creator can generate a perceptual hash from his original work. Then, as explained in (Zauner, 2010), he has two options.

On the one hand it is possible to sign the perceptual hash with a private key in order to have a digital signature that is robust to slight modifications, for example JPEG compression. This measure protects the receiver of the media object because he can check its authenticity.

On the other hand, it is possible to use the perceptual hash for digital watermarking. This measure protects the content author. For example, a different watermark can be applied on different images. Then if an illegal copy is found the copyright owner can infer who is responsible for the data leak.

3.3. Examples

In this section, we present the three examples of implementations of perceptual hash functions, which take images and output binary codes, from (Zauner, 2010).

The Discrete Cosine Transformation (DCT) based perceptual hash function takes advantage of the property that low-frequency DCT coefficients are mostly stable under image modifications to construct a 64 bits binary code, which are compared with a Hamming distance.

The Marr-Hildreth (MH) operator, also denoted as the Laplacian of Gaussian (LoG), is a special case of a discrete Laplace filter. It is an edge and contour detection based image feature extractor. The MH operator generates vectors encoded on 576 bits.

The Radial Variance hash is based on the Radon transform that is the integral transform which consists of the integral of a function over a straight line. It is robust against various image processing steps (e.g. compression) and more robust than the DCT and MH based perceptual hash functions against geometrical transformations (e.g. rotation up to 2 degrees).

4. Convolutional Neural Networks

This section presents the concept of convolutional neural networks. Firstly, we briefly explain how they work. Secondly, we show that they can extract excellent image representations in the context of image retrieval, thus their use for Reverse Image Search is of interest. Finally, we present three existing models, which perform particularly well on the ImageNet dataset. Retrieval performances of these neural networks are evaluated in section 6.

4.1. Definition

A Convolutional Neural Network (CNN) is a class of feed-forward neural network. CNN are mainly applied in the fields of computer vision and natural language processing. It has been first presented in (LeCun *et al.*, 1998). Recent progresses have led to a gain in interest in this method and also in a broader field called Deep Learning, which is described in (LeCun *et al.*, 2015). This section is mainly based on this latter article. It has been a great discover because this class of network is able to automatically learn the features to extract from a dataset. Whereas traditionally, the previous works in computer vision were based on hand-engineered features. This difference is essential because the key success factor of CNN is the amount of data and computation power available instead of field knowledge.

CNN are inspired by our visual cortex, in particular, the connectivity pattern between layers of artificial neurons is restricted to a region as it is in our visual system with receptive fields. With this special connectivity pattern, CNN take advantage of natural signals, which are often locally highly correlated. Another property of natural signals is the fact that the local statistics of images are invariant to location. If a motif can appear in one location of an image, it could also appear somewhere else.

CNN are built on top of four key ideas: local connection, shared weights, pooling and the use of many layers. The architecture of a typical CNN is composed of these types of layers; in a first part: convolution layer, ReLU, pooling layer and in a second part: fully connected layer, loss layer. Each type of layer is going to be presented in one of the next several paragraphs.

The convolution layer applies a set of discrete convolutions on the output of the previous layer. Each convolution is done using a kernel (also called: mask, filter bank) and produces a 2D feature map. The feature maps contain high activations if their convolutions have detected an interesting motif. The layer's parameters are the weights in the kernels. This layer can also be seen as a layer with local connections, so that a neuron is only connected to neurons of the previous layer in the same region. Local connections give the ability to the layer to detect local patterns from the previous layer. Moreover, the weights of the local connections are shared among all neurons in the layer, this means that the same local pattern can be detected in the same way anywhere in the previous layer.

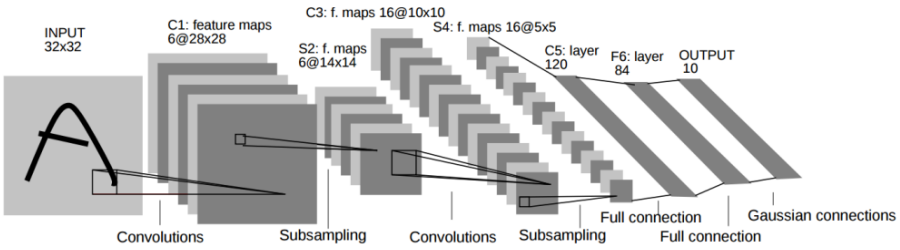

Figure 2. Architecture of LeNet-5, a basic Convolutional Neural Network

The ReLU (Rectified Linear Units) applies a non-linear function $f(x) = \max(0, x)$ to the output of the previous convolution layer. Other non-linear activation function are available: hyperbolic tangent or sigmoid function for example. The ReLU have become very popular because they improve the speed of convergence during the training of neural networks (Krizhevsky *et al.*, 2012).

The pooling layer splits the output of the previous layer into a grid, then on each cell of the grid a pooling function is applied and outputs only a single value. Thus, the pooling layer is a down-sampling step, the resolution of the grid can vary and affects the sampling rate. This layer merges multiple semantically similar features into one. Due to the down-sampling, the position of the feature is not accurately preserved, therefore an invariance to small shifts or distortions is spawned. In fact, the position of a feature is not as important as its relative position to others features. A typical pooling function outputs the maximum value of a local patch.

Repeating these layers several times in this order: convolution, ReLU, pooling, one can exploit the fact that high level features are composed of lower level features. The first group of layers can detect small details such as edges and contours. The second layer can detect higher level features such as motifs, and so on. This process is repeated so that firstly, edges are merged into motifs, motifs into parts and finally parts into objects. The architecture of a very basic CNN is shown in figure 2.

After this first part composed of convolution layers, ReLU, pooling layers, a second part of the neural network is composed of fully-connected layers and a loss layer. This second part is a classical feed-forward neural network used to classify the features coming from the first part. The training of CNN is possible using the backpropagation algorithm. Because of the local connectivity and the shared weights, there is less parameters in the model thus the network is less prone to overfitting and generalizes better.

4.2. Feature extraction

Recent work showed that the representation learned by the CNN is a good descriptor for image retrieval. With the AlexNet network (Krizhevsky *et al*., 2012), one can use the feature activations induced by an image at the last hidden layer to represent an image. Experiments has shown that semantically similar images get a similar feature vector in a Euclidean space, even if the images are not close in L2. The only drawback is that the representation is a real-valued vector of dimension 4096. Therefore, the computation of the distance between two vectors is expensive, moreover the storage cost could also be an issue. To tackle this issue, the authors suggest a dimensionality reduction with an auto encoder.

In a later paper (Babenko *et al*., 2014), the performance of feature vectors extracted with CNN in the context of image retrieval is studied more in depth. The conclusion is that CNN performs well for image retrieval, even if the network is trained on an unrelated classification task. The performances can be improved with a fine tuning on a dataset similar to the retrieval dataset. For example, by using a neural network trained on ImageNet to retrieve landscape images, the performances are good. Unsurprisingly, after a retraining on a dataset composed of landscape images the performances become even better. The compression with PCA is also investigated, and the retrieval performance is not too much affected when compared to other state of the art descriptors. Image features extraction with CNN, for all these reasons, seems to be very promising for image retrieval.

4.3. Models

In this section, three models, which perform very well on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), are presented. All of them are implemented in the Keras (Chollet *et al*., 2015) library and are used later in this thesis.

The VGG network (Simonyan, Zisserman, 2014) was created in 2014 for ILSVRC. The authors investigated the effect of depth on the accuracy. For that purpose, they use a simple convolutional neural network with very small (3x3) convolution filters with stride and pad of 1, 2x2 maxpooling with stride 2, and push the depth to 16-19 layers, which was a lot at the time. VGG16 and VGG19 are detailed in table 1.

The InceptionV3 network (Szegedy *et al*., 2015) was created in 2015. Instead of stacking convolutional and pooling layers sequentially on top of each others, some layers are in parallel and their results are merged periodically. As shown in this article from Google Research Blog[5], the key idea of this network is to stack Inception modules, which are composed of convolutional and pooling layers. By using fewer weights than previous neural networks, for instance about 30x fewer parameters than VGG19, the computational cost of InceptionV3 is suitable for big-data or mobile scenarios.

5. https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html

Table 1. Configurations of VGG16 and VGG19 networks. The convolutional layer parameters are denoted as "'conv(receptive field size)-(number of channels)"'. The ReLU activation is not shown for brevity.

VGG16	VGG19
input (224x224 RBG image)	
conv3-64	conv3-64
conv3-64	conv3-64
maxpool	
conv3-128	conv3-128
conv3-128	conv3-128
maxpool	
conv3-256	conv3-256
conv3-256	conv3-256
conv3-256	conv3-256
	conv3-256
maxpool	
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
	conv3-512
maxpool	
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
	conv3-512
maxpool	
FC-4096	
FC-4096	
FC-1000	
soft-max	

The ResNet50 network (He *et al*., 2015) was created in 2015 for ILSVRC. The authors use a 50 layers Residual Network, whose key idea is to introduce shortcuts connections into a sequential network. As networks are becoming deeper, the learning is more difficult because of the vanishing/exploding gradient problem. In particular, with the network depth increasing, accuracy gets saturated and then degrades rapidly. The idea behind shortcuts in Residual Networks is to by-pass a set of layers if they are not useful to improve the accuracy of the whole network. Thus, in theory, it is possible to stack as many building blocks as possible without affecting the accuracy, because unnecessary blocks can be skipped.

Table 2. Contindency table for information retrieval

	Relevant	Non-relevant
Retrieved	True positive (tp)	False positive (fp)
Not retrieved	False negative (fn)	True negative (tn)

5. Benchmarking

In a previous work (Gaillard, Egyed-Zsigmond, 2017), we designed a new framework to benchmark reverse image search engines. Our approach is based on evaluation measures of information retrieval systems described in (Manning *et al.*, 2008). We model the reverse image search engine as an information retrieval system that returns an unranked set of documents for a query. If many documents are retrieved, the user is in charge of choosing the one that best suits his needs. This section presents our benchmark along with the results we obtained with the three perceptual hashing functions presented in section 3.

5.1. Metrics

To process a query, the reverse image search engine order the indexed images by relevance. Results are retrieved using a fixed-radius nearest neighbor search. Thus, each image, whether relevant for the query or not, can be retrieved or not. This notion can be made clear by examining the following contingency table 2.

The effectiveness of the system is measured with the following metrics: Precision (P) is the fraction of retrieved documents that are relevant,

$$precision = \mathbb{P}(\text{relevant}|\text{retrieved}) = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = \frac{tp}{tp + fp}$$

Recall (R) is the fraction of relevant documents that are retrieved.

$$recall = \mathbb{P}(\text{retrieved}|\text{relevant}) = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = \frac{tp}{tp + fn}$$

A single measure that trades off precision versus recall is the F-measure, which is the weighted harmonic mean of precision and recall:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \qquad F_{\beta=1} = \frac{2PR}{P + R}$$

It is possible to change the weights in the harmonic mean of the F measure in order to tune it. This is done by changing the β parameter. Values of $\beta < 1$ emphasize precision, while values of $\beta > 1$ emphasize recall. This is important in order to benchmark the system in accordance with its application.

5.2. Protocol

In order to compare the effectiveness and the speed of different image search methods, we created a comparison protocol. We choose 25,000 images from the MIR-FLICKR dataset[6] because they are representative of what is commonly seen on Internet.

To be able to calculate automatically the precision and recall of the results, we applied 6 small modifications on each image, that gave us a dataset with 175 000 images. We measured the results precision and recall.

The modifications (illustrated on Figure 3) applied on the images are:

1. Gaussian blur ($r = 4$, $\Sigma = 2$)
2. Black and white transformation
3. Resize to half height and width
4. JPEG compression with $quality = 10$
5. Clockwise rotation by 5 degrees
6. Crop by 10% at the right side of the image.

These modifications represent the basic cases of small changes images usually undergo over the web. The benchmark is programmed so that it is easy to change the set of modifications.

Our benchmark protocol for a generic reverse image search engine is detailed in this section.

1. Select $N + M$ images that are representative to an application (see section 2.2) with no duplicated images. In our case $N = 24,000$ and $M = 1,000$ (for reproducibility, the first 1,000 images from the dataset in alphabetical order)

2. Split them into 2 sets of N base images and M non-indexed images.

3. Select K modifications and from the N base images, generate K new image sets containing $K \times N$ modified images. In our case $K = 6$, and the modifications are those enumerated above.

4. Index the N base images and the $K \times N$ modified images according to the image representation method.

5. Make search queries with:

6. http://press.liacs.nl/mirflickr

Figure 3. Illustration of the image modifications.

a) The M images from the non-indexed image set,

b) The N images from the base image set.

c) The $K \times N$ images from the modified sets.

6. Analyse the search results and compute the mean precision, the mean recall and the mean F-measure of all queries.

a) For the M images of the non-indexed set, there should be no relevant result image. Thus, the result should be empty.

b) When querying with one of the other $(K + 1) \times N$ already indexed images, the relevant results are the $K + 1$ images that are the modified version of the query image. Thus, the result of each query should contain K+1 images that come from the same base image as the query image.

Because our system use a fixed-radius nearest neighbor search, we model it as an information retrieval system that returns an unranked set of documents. To determine the best radius, we repeat this protocol for several different radii.

It is possible to use the protocol in two different ways. Firstly, by applying all modifications to have an overview of the performances of a method. Thus, it is possible to compare two methods according to a complete workload. Secondly, by applying a single modification, for example to identify the strength and weakness of a method according to some modifications.

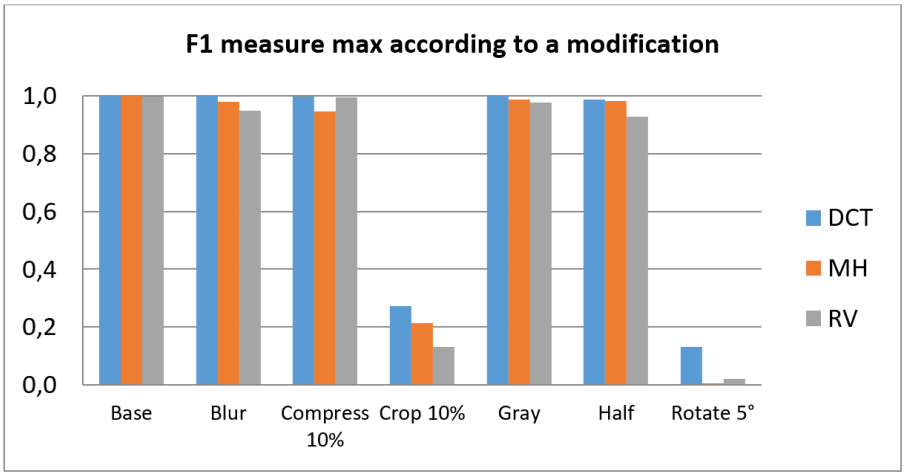

Figure 4. Maximum F1-measure of DCT, MH and RV perceptual hash functions against single modifications.

We implemented the protocol[7] as Linux shell commands and C++, using online available libraries.

5.3. Results

We evaluate the methods described in section 3: DCT, MH and RV based perceptual hash functions from (Zauner, 2010).

5.3.1. Retrieval performance against single modification

We index the N base images and then make K search queries each with one of the N modified images. We compute the F-measure for various radius and take the maximum.

The functions are robust against Gaussian blur ($r = 4, \Sigma = 2$), JPEG compression (quality 10%), grayscale filter, and scale to half the size. However the functions are not robust against crop (10% on the right) and rotate (5 degrees clockwise) modifications as illustrated on Figure 4.

We also tested, the evolution of our accuracy measures with different degrees of the modifications. We noticed that the different hash methods were quite sensible

7. See our implementation on: https://github.com/mgaillard/pHashRis

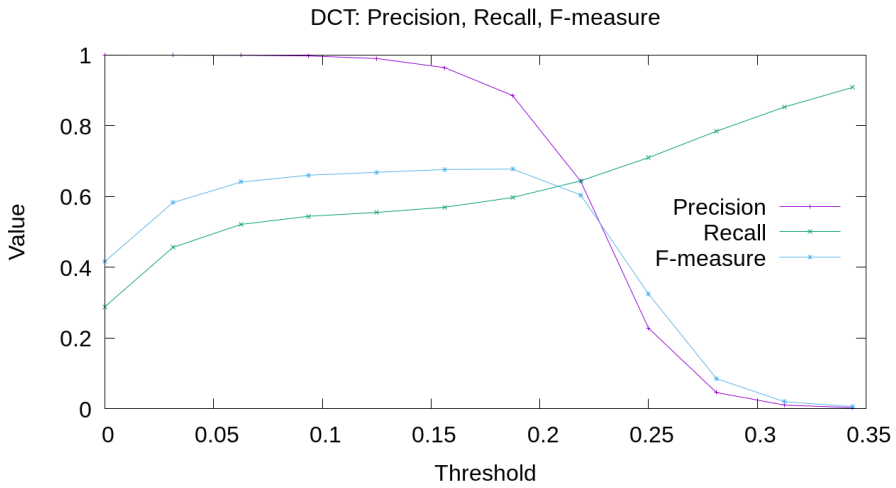

Figure 5. Precision/Recall/F-measure curves of the DCT based perceptual hash function search results according to different radius values

to rotations (above 2 degrees) or to cropping (above 5%), but resisted very well to compression, blur or resize.

5.3.2. Retrieval performance against all modifications

To evaluate the speed and retrieval performance of a reverse image search engine, we run the protocol described in the previous section. The results are in figures 5, 6, 7. We evaluate the retrieval performance, calculating the mean precision, recall and F-measure of the returned results. The calculation is repeated for different radius values. The radius here is a normalized distance value, below which two images are considered as similar. We can see that all functions have equivalent retrieval performances with a maximum F-measure of about 60%.

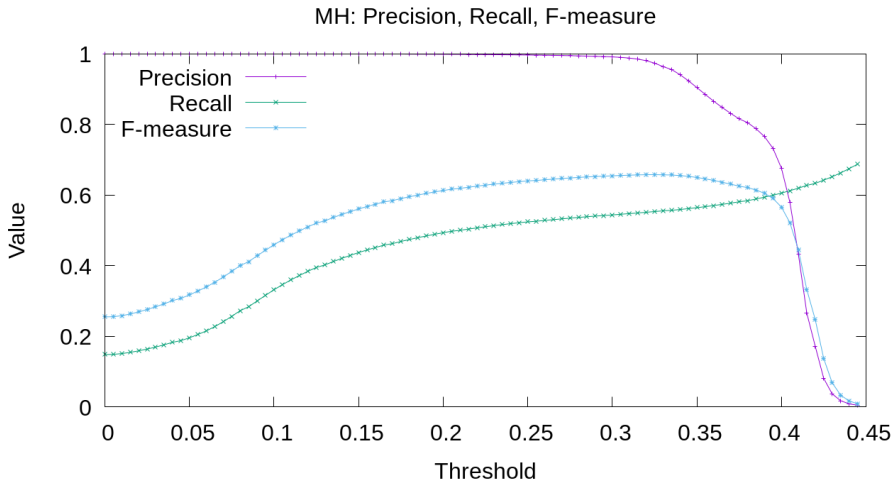

Figure 6. Precision/Recall/F-measure curves of the Marr-Hilderth based perceptual hash function search results according to different radius values

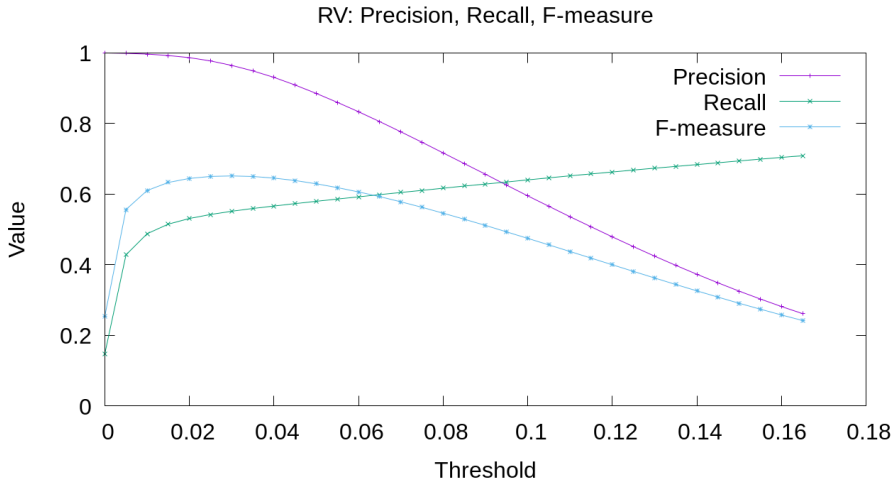

Figure 7. Precision/Recall/F-measure curves of the Radial Variance based perceptual hash function search results according to different radius values

6. CNN Features

In section 4, we advocate the use of convolutional neural networks to extract features from images. In this section, we evaluate the robustness of CNN features against modifications using the benchmark presented in section 5. We assess the robustness of existing networks trained on an unrelated classification task. The aim being to find an already good representation, according to a specific distance.

6.1. Protocol

The protocol is inspired from our previous benchmark for Reverse Image Search described in section 5. It is based on a set of images that are modified according to some transformations. We then consider a base image and its modified versions as similar amongst themselves, and all other pairs of images as dissimilar. All images are queried and the average precision, recall and Fmeasure are computed over all queries.

6.1.1. Preparation

Following are the steps to prepare images for the benchmark.

1. Select N images that are representative to an application (see section 2.2) with no duplicated images. In our case, N is equal to either 200, 2500 or 25,000.

2. Select K modifications and from the N base images, generate K new image sets containing $K \times N$ modified images. In our case $K = 6$, and the modifications are those enumerated in section 5.2: blur, grayscale, resize, JPEG compression, rotation and cropping.

3. With a convolutional neural network, extract the features from all images. More details about this step are given in section 6.2.

6.1.2. RIS benchmark

To get a precise idea of the retrieval performance using CNN features, we implement the same benchmark as in section 5. We compute all distances between all pairs of images and apply a threshold below which images are considered as similar. Then we compute the mean precision, the mean recall and the mean F-measure. The protocol is repeated for various thresholds, the threshold that maximizes the F-measure can be considered as the best radius for the reverse image search system.

1. Make search queries with the $(K + 1) \times N$ images from the base and modified sets.

2. Analyse the search results and compute the mean precision, the mean recall and the mean F-measure of all queries.

a) The relevant results are the $K + 1$ images that are the modified versions of the query image. Thus, the result of each query should contain K+1 images that come from the same base image as the query image.

The resulting plots can be seen in the Result section 6.4 of this chapter. Ideally, the maximum F-measure should be near to 1.

6.2. Models

In this section, we describe models compared in the benchmark. To extract the features, we use a pretrained neural network to predict the label of an image. The feature vector consists of the activations of the units of one specific layer. If the layer is a convolutional or pooling layer, the results are 2D features maps. In this case, the activations are flattened using a global average or maximum pooling. We compare the models described in section 4.3: *VGG16*, *VGG19*, *ResNet50*, *InceptionV3*. We also evaluate *Xception*, which is also a model available in Keras. If needed the feature vectors can be normalized to unit length in Euclidean distance. To compare the feature vectors, two distances are used: Euclidean and Cosine.

In the following sections, we describe, for all models, all layers that we compared. The name of layers are the same as in the Keras library. For the models, the naming convention we use in the program is: [network]_[layer]. If the feature vector is normalized with L2 we add *_norm_l2* at the end. For example, *ResNet50_flatten_1_norm_l2* is extracted from the *flatten_1* layer of the *ResNet50* network and is normalized as unit length in Euclidean distance.

6.2.1. VGG

In VGG16 and VGG19, available layers ordered by increasing depth are:

- block{i}_pool_avg for $i \in \{3, 4, 5\}$
- block{i}_pool_max for $i \in \{3, 4, 5\}$
- flatten
- fc1
- fc2
- predictions

6.2.2. ResNet50

In ResNet50, available layers ordered by increasing depth are:

- activation_{i}_avg for $i \in \{4, 7, 10, 13, 16, 19, 22, 28, 31, 34, 37, 40, 43, 46\}$
- activation_{i}_max for $i \in \{4, 7, 10, 13, 16, 19, 22, 28, 31, 34, 37, 40, 43, 46\}$
- avg_pool_avg
- avg_pool_max
- flatten_1
- predictions

6.2.3. InceptionV3

In InceptionV3, available layers ordered by increasing depth are:

- mixed{i}_avg for $i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- mixed{i}_max for $i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- predictions

6.2.4. Xception

In Xception, available layers ordered by increasing depth are:

- add_{i}_avg for $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- add_{i}_max for $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- block14_sepconv1_act_avg
- block14_sepconv1_act_max
- block14_sepconv2_act_avg
- block14_sepconv2_act_max
- predictions

6.3. Implementation

Our implementation is on GitHub[8]. It consists of three main programs and a bash script to orchestrate their executions.

The first program is a Python script that extracts features from images using the neural network library Keras (Chollet *et al.*, 2015). The feature vectors can be saved in HDF5 files, which are then read during the execution of the second and third programs.

The second program is the implementation of the distance benchmark. It computes statistics on the distribution of distances between similar and dissimilar images. Those are first written in a file and then used to generate a plot with gnuplot.

The third program is the RIS benchmark. It is written in C++ with OpenMP, to speed up the computation of distances in high dimensional spaces thanks to multi-threading and SIMD. This benchmark computes the mean precision, recall and F-measure of a RIS system based on the previously extracted features. The results are first written in a file and then used to generate a plot with gnuplot.

The workflow of the bash script is:

1. Images are modified according to the K modifications with ImageMagick[9].

2. Features are extracted from the $(K + 1) \times N$ images according to all possible models.

8. https://github.com/mgaillard/CNNFeaturesRobustness
9. www.imagemagick.org

Table 3. Benchmark with all modifications on 25,000 images: maximum Fmeasure

Model	Dimensions	Distance	Radius	Precision	Recall	Fmeasure
VGG16_flatten	25088	cosine	0,51	0,954	0,943	0,936
VGG19_flatten	25088	cosine	0,50	0,955	0,934	0,931
ResNet50_activation_46_max	2048	cosine	0,17	0,968	0,913	0,928
ResNet50_activation_43_max	2048	cosine	0,14	0,952	0,919	0,921
VGG16_block5_pool_max	512	cosine	0,23	0,947	0,898	0,904
VGG19_block5_pool_max	512	cosine	0,23	0,948	0,892	0,902
InceptionV3_mixed9_max	2048	cosine	0,20	0,926	0,892	0,881
InceptionV3_mixed9_avg	2048	cosine	0,20	0,924	0,865	0,866

3. The distance benchmark is executed for all features according to Euclidean and Cosine distance.

4. The RIS benchmark is executed for all features according to Euclidean and Cosine distance.

6.4. Results

Because the benchmark is computationally intensive, to find the best model we proceed in three steps. Firstly, we compare all models with all distances but only with 200 images. We retain only the models whose F-measure is above 90%. Secondly, we repeat this process with 2,500 images. The best models with 2500 images ordered by decreasing F-measure are:

- *VGG16_flatten* with Cosine distance; $Fmeasure = 0,952$
- *VGG19_flatten* with Cosine distance; $Fmeasure = 0,949$
- *ResNet50_activation_46_max* with Cosine distance; $Fmeasure = 0,948$
- *ResNet50_activation_43_max* with Cosine distance; $Fmeasure = 0,941$
- *VGG16_block5_pool_max* with Cosine distance; $Fmeasure = 0,931$
- *VGG19_block5_pool_max* with Cosine distance; $Fmeasure = 0,929$
- *InceptionV3_mixed9_max* with Cosine distance; $Fmeasure = 0,918$
- *InceptionV3_mixed9_avg* with Cosine distance; $Fmeasure = 0,911$

Finally, we compare these models with all the 25,000 images from the MIR-FLICKR collection.

6.5. Retrieval performance against all modifications

The results of the benchmark with all modifications are shown in table 3. Each line represents the maximum F-measure of each model along with the corresponding radius, precision and recall.

	Base	Blur	Gray	Resize	Compr	Rotate	Crop
VGG16_flatten	1,000	0,985	0,975	0,998	0,983	0,930	0,998
VGG19_flatten	1,000	0,984	0,971	0,998	0,982	0,920	0,998
ResNet50_activation_46_max	1,000	0,983	0,964	0,998	0,968	0,928	0,998
ResNet50_activation_43_max	1,000	0,984	0,955	0,998	0,966	0,927	0,999
VGG16_block5_pool_max	1,000	0,955	0,926	0,996	0,966	0,848	0,995
VGG19_block5_pool_max	1,000	0,954	0,926	0,996	0,966	0,846	0,995
InceptionV3_mixed9_max	1,000	0,910	0,953	0,994	0,917	0,857	0,994
InceptionV3_mixed9_avg	1,000	0,906	0,914	0,993	0,892	0,866	0,999

Table 4. Maximum Fmeasure of CNN models against single modifications.

We can see that despite the fact that the F-measure decreases by about 3 points with 25,000 images compared to 2,500 images, the F-measures are still very good. For comparison, on the same benchmark, the DCT based perceptual hash function yields to a F-measure of about 0.6 (cf. section 5.3).

6.6. Retrieval performance against single modification

The results of the benchmark with single modifications are shown in table 4. Each column represents the maximum F-measure of each model against a single modification. The radius at which the F-measure is maximum can vary depending on the model and the modification. Thus, this figure gives an idea of the best retrieval accuracy against a single modification. This is why individually the F-measure is better than in the benchmark against all modifications. Because in this latter, the radius is the same for all modifications.

We can see that all models work well and their F-measures are at least above 0.84. All models are very robust against resize to half-size and cropping 10% on the right, with a F-measure greater than 0.99. The robustness against Gaussian blur, grayscale filter and JPEG compression is good, with a F-measure greater than 0.9. Rotation is the harder modification, with a F-measure of about 0.85 for *VGG16_block5_pool_max*, *VGG19_block5_pool_max*, *InceptionV3_mixed9_max* and *InceptionV3_mixed9_avg*, and about 0.92 for the others. Models based on *ResNet50* yields to the best results for all modifications.

7. Conclusion

The potential of CNN Features for reverse image search, even with slight modifications, is proven. Image representations extracted with convolutional neural networks on unrelated classification tasks are considerably robust against modifications. We didn't expressly compared CNN Features against other state of the art descriptors, but our experiments, along with the content of other publications presented in chapter 4, tend to prove that CNN Features are state of the art descriptors for reverse image search and more generally for image retrieval.

In section 5, we compare different perceptual hash functions, which can be used as image representation, and show that they are not robust against cropping and rotation. In section 6, we compare features extracted with off the shelf convolutional neural networks and find a set of models that have very good retrieval performances against modifications. Especially, these are quite robust against rotation and very robust against cropping.

Robustness is certainly caused by preprocessing and data augmentation during training. Because the input of the convolutional neural network has always a constant size, images are first resized, thus features are robust against scaling. During training, it is common to generate more samples by transforming original images. For example, during the training of VGG (Simonyan, Zisserman, 2014), to obtain the fixed size input images, training images are randomly cropped from rescaled training images. Therefore, the neural network is forced to learn a representation that is robust against cropping. More generally, by choosing the modifications made during the data augmentation step, one could give the neural network the ability to be robust against these modifications.

Two different distances are used to compare feature vectors: Euclidean and Cosine. However, it appears that cosine distance yields better results than Euclidean distance on the networks we tested. We think that it is because, ReLU units are activated with the presence of a feature. The level of presence of the feature being not as important as the fact that it is present.

Of course, CNN features are very robust against modifications but they are not suited for large scale applications. Indeed, the dimensionality of CNN feature vectors is very high. For more details, see the dimensions column of table 3. As explained in section 2.6, it is currently impossible to index p-dimensional feature vectors when p is greater than 20. For a large-scale application, one should reduce the dimension of the CNN feature vectors. The most popular approach is Locality Sensitive Hashing (LSH) with random projections (Wang *et al.*, 2014), which hashes p-dimensional into binary codes while preserving angular distance. Other methods, which take into account the data distribution, are available for example: Semantic Hashing (Salakhutdinov, Hinton, 2007),(Salakhutdinov, Hinton, 2009), Minimal Loss Hashing (Norouzi, Fleet, 2011) and Triplet Ranking Loss (Norouzi *et al.*, 2012).

In order to make CNN features more scalable, our next challenge is to develop methods to reduce the dimensionality of the CNN feature vectors while preserving similarity.

References

Babenko A., Slesarev A., Chigorin A., Lempitsky V. (2014). Neural codes for image retrieval. In *European conference on computer vision*, pp. 584–599.

Baeza-Yates R., Ribeiro-Neto B. *et al*. (1999). *Modern information retrieval* (Vol. 463). ACM press New York.

Bhattacharyya A. (1943). On A Measure of Divergence Between Two Statistical Populations Defined by their Probability Distributions. *Bulletin of the Calcutta Methematical Society*, Vol. 35, No. 1, pp. 99–109.

Chatzichristofis S. A., Boutalis Y. S. (2008). Cedd: Color and edge directivity descriptor: A compact descriptor for image indexing and retrieval. In *Proceedings of the 6th international conference on computer vision systems*, pp. 312–322. Berlin, Heidelberg, Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=1788524.1788559

Chollet F. *et al*. (2015). *Keras.* https://github.com/fchollet/keras. GitHub.

Datta R., Joshi D., Li J., Wang J. Z. (2008, May). Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, Vol. 40, No. 2, pp. 5:1–5:60. Retrieved from http://doi.acm.org/10.1145/1348246.1348248

Douze M., Jégou H., Harsimrat S., Amsaleg L., Schmid C. (2009, July). Evaluation of GIST descriptors for web-scale image search. In *CIVR 2009 - International Conference on Image and Video Retrieval*, p. 19:1-8. Santorini, Greece, ACM. Retrieved from https://hal.inria.fr/inria-00394212

Gaillard M., Egyed-Zsigmond E. (2017). Large scale reverse image search - A method comparison for almost identical image retrieval. In *Actes du xxxvème congrès inforsid, toulouse, france, may 30 - june 2, 2017*, pp. 127–142. Retrieved from http://inforsid.fr/actes/2017/INFORSID_2017_paper_34.pdf

He K., Zhang X., Ren S., Sun J. (2015). Deep residual learning for image recognition. *CoRR*, Vol. abs/1512.03385. Retrieved from http://arxiv.org/abs/1512.03385

Krizhevsky A., Sutskever I., Hinton G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

LeCun Y., Bengio Y., Hinton G. (2015). Deep learning. *Nature*, Vol. 521, No. 7553, pp. 436–444.

LeCun Y., Bottou L., Bengio Y., Haffner P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324.

Lowe D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, Vol. 60, No. 2, pp. 91–110.

Manning C. D., Raghavan P., Schütze H. (2008). *Introduction to information retrieval*. New York, NY, USA, Cambridge University Press.

Menezes A. J., Vanstone S. A., Oorschot P. C. V. (1996). *Handbook of applied cryptography* (1st ed.). Boca Raton, FL, USA, CRC Press, Inc.

Norouzi M., Fleet D. J. (2011). Minimal loss hashing for compact binary codes. In *Proceedings of the 28th international conference on machine learning (icml-11)*, pp. 353–360.

Norouzi M., Fleet D. J., Salakhutdinov R. R. (2012). Hamming distance metric learning. In F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25*, pp. 1061–1069. Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/4808-hamming-distance-metric-learning.pdf

Oliva A., Torralba A. (2001, May). Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision*, Vol. 42, No. 3, pp. 145–175. Retrieved from http://dx.doi.org/10.1023/A:1011139631724

Rubner Y., Tomasi C., Guibas L. (1998). A metric for distributions with applications to image databases. In, pp. 59–66. IEEE Publishing.

Salakhutdinov R., Hinton G. (2007). Semantic hashing. *RBM*, Vol. 500, No. 3, pp. 500.

Salakhutdinov R., Hinton G. (2009). Semantic hashing. *International Journal of Approximate Reasoning*, Vol. 50, No. 7, pp. 969–978.

Simonyan K., Zisserman A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, Vol. abs/1409.1556. Retrieved from http://arxiv.org/abs/1409.1556

Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, Vol. abs/1512.00567. Retrieved from http://arxiv.org/abs/1512.00567

Wang J., Shen H. T., Song J., Ji J. (2014). Hashing for similarity search: A survey. *CoRR*, Vol. abs/1408.2927. Retrieved from http://arxiv.org/abs/1408.2927

Zauner C. (2010). Implementation and benchmarking of perceptual image hash functions.