University of Pisa

DEPARTMENT OF INFORMATION ENGINEERING

Master of Science in Computer Engineering

MASTER THESIS

Design and implementation of a Multimedia Information Retrieval Engine for the MSR-Bing Image Retrieval Challenge

SUPERVISORS

Prof. Giuseppe Amato

Prof. Claudio Gennaro

Prof. Francesco Marcelloni

CANDIDATE

Alessia Razzano

ACADEMIC YEAR 2014/2015

Abstract

The aim of this work is to design and implement a multimedia information retrieval engine for the MSR-Bing Retrieval Challenge provided by Microsoft. The challenge is based on the Clickture dataset, generated from click logs of Bing image search. The system has to predict the relevance of images with respect to text queries, by associating a score to a pair (image, text query) that indicates how the text query is good at describing the image content. We attempt to combine textual and visual information, by performing text-based and content-based image retrieval. The framework used to extract visual features is Caffe, an efficient implementation of deep Convolutional Neural Network(CNN).
Decision is taken using a knowledge base containing triplets each consisting of a text query, an image, and the number of times that a users clicked on the image, in correspondence of the text query. Two strategies were proposed. In one case we analyse the intersection among the triplets elements retrieved respectively using the textual query and the image itself. In the other case we analyse the union. To solve efficiency issues we proposed an approach that index visual features using Apache Lucene, that is a text search engine library written entirely in Java, suitable for nearly any application requiring full-text search abilities. To this aim, we have converted image features into a textual form, to index them into an inverted index by means of Lucene. In this way we were able to set up a robust retrieval system that combines full-text search with content-based image retrieval capabilities. To prove that our search of textually and visually similar images really works, a small web-based prototype has been implemented.
We evaluated different versions of our system over the development set in order to evaluate the measures of similarity to compare images, and to assess the best sorting strategy. Finally, our proposed approaches have been compared with those implemented by the winners of previous challenge editions.

Contents

List of Figures

List of Tables

Abbreviations

IR	**I**nformation **R**etrieval
CBIR	**C**ontent **B**ased **I**mage **R**etrieval
TBIR	**T**ext **B**ased **I**mage **R**etrieval
DP	**D**eep **L**earning
AI	**A**rtificial **I**ntelligence
ILSVRC	**I**mageNet **L**arge-**S**cale **V**isual **R**ecognition **C**hallenge
VSM	**V**ector **S**pace **M**odel
DCG	**D**iscounted **C**umulative **G**ain

Chapter 1

Introduction

Recent years have seen a rapid increase of digital data size, thanks to the popularity of the network and development of multimedia technology. Everyday, people generate a huge amount of data and images. However, it is hard to exploit this visual information without organizing it in a way that allows efficient browsing, searching and retrieval. Thanks to the increase in the digital data and the consequent use of on-line text database, researchers started to study better techniques to access this information and have expressed strong interest in the research area of Information Retrieval (IR). The capabilities of retrieval systems grew with increases in processor speed and storage capacity. With the expansion and the progress of these systems, there has been a significant change in the manual methods of acquiring, indexing and searching information. In fact, they became automated and consequently faster.

When digital collections reach dimensions in which the traditional and common cataloguing techniques can no longer be used, IR systems become crucial. With the growth of digitized unstructured information and via high speed networks, there has been a rapid increase of accesses to enormous quantities of that information. Search was the only feasible solution to retrieve relevant items from these large database, so IR systems became ubiquitous.

An area of IR that has been active and expanding in the past few years, is *Image Retrieval*. In many science and industry areas such as biology, medicine, astronomy, government, university and so on, large collections of digital images are used and created. Many of these are the product of digitizing existing collections of analogue photographs, diagrams, drawings, paintings and prints. Many image retrieval systems, both commercial and research, have been built.

In this work, we present our system designed for the MSR-Bing Image Retrieval Challenge provided by Microsoft. The task is to design an effective and efficient strategy to decide the relevance of a text query with respect to an image from the web.

In the rest of the chapter the main contributions of this thesis, an introduction to the challenge task, the dataset and the evaluation metric are reported. Finally, we briefly show and discuss the structure of this thesis.

1.1 Contributions of this thesis

This thesis has four main contributions: 1) definition and implementation of two strategies for evaluating the relevance of images with respect to text queries, using deep convolutional neural networks, 2) using a text retrieval engine to efficiently index and retrieve visual deep features, 3) definition and implementation of a web based image retrieval prototype and 4) evaluation and comparison of the proposed strategies.

Strategy to sort images

Given a query and a list of images, a list of relevance scores should be returned. We propose two strategies to sort images according to relevance to a text query: intersection and union of retrieved candidates. For each query-image pair, we retrieve visually and textually similar triplets from the knowledge base. Visually similar triplets are retrieved indexing images by using deep Convolutional Neural Network features (deep CNN) and searching in the knowledge base the triplets having most similar images to the given image. Then, we consider the text query and we retrieve the triplets having the most similar text to the query text. At the end of these procedures, we have two set of candidates, selected from the knowledge base, one coming from the visual search and the other coming from the text search. These two candidate sets are analysed to decide the relevance of the image to the text query, by either first performing their intersection or union. Their descriptions and details are reported in Chapter 4.

Extraction and indexing of visual deep features

Visual feature (content) extraction is at the basis of content-based Image Retrieval. Visual features, such as color, texture and shape information of images are extracted automatically. Similarity between images is computed by evaluating the similarity between their visual features. In our work, we use features obtained using deep Convolutional Neural Networks (deep CNN), since they have demonstrated very high effectiveness in this type of tasks. We adapted the Convolutional Neural Networks (CNN) from ImageNet to the MSR-Bing Challenge. The framework we use is Caffe, an efficient implementation of deep CNN. We extract features from the output of the 6-th layer in the network, that is the first fully connected layer, and we store and indexing them through Apache Lucene.

Web-based prototype

To allow interactive use of the entire system, we also implemented a multimedia information retrieval engine. The user inserts a textual query and the IR engine returns a list of textually similar images. It is also possible to perform visual searching by clicking on the images returned by previous search. Since in our dataset we have base$_{64}$ encoded jpeg image thumbnail, we also implemented a RESTful Web Service that decodes and returns them.

Test procedure

We evaluated different versions of our system over the Development set, provided by the MSR-Bing Image Retrieval Challenge, in order to evaluate the measures of similarity used to compare images, and to assess the best sorting strategy. We also tried to perform an exhaustive search. We have compared: a) the standard measure of similarity that is called cosine similarity with an approximate measure, b) the sequential scan of the entire knowledge base with an efficient index for similarity search, c) different values of the candidates sets size, ranging from 1 to 10000. More details are reported in Chapter 5.

1.2 MSR-Bing Image Retrieval Challenge

The MSR-Bing Image retrieval challenge [1] is executed on the Clickture dataset generated from click logs of Bing image search. The contestants are asked to develop systems to assess the effectiveness of query terms in describing the images crawled from the web for image search purposes. The task of MSR-Bing Retrieval Challenge is to predict the relevance between a pair of image and textual query. For this purpose, a contesting system is asked to produce a floating-point score on each image-query pair that reflects how relevant the query could be used to describe the given image, with higher numbers indicating higher relevance.

1.2.1 Dataset

The dataset is called **Clickture-Lite** and was sampled from one-year click logs of Microsoft Bing Image search engine. It comprises three parts: (1) the *Training Dataset*, (2) the *Development Dataset*, and (3) the *Test* (or *Evaluation*) *Dataset*. The dataset structure is shown in Fig. 1.1

The Training Dataset (or knowledge base) is a uniform random sample of Bing user click log. It takes all clicked <query, image> pairs from one year's of Bing Image Search log in the EN-US market. A pair has N instances in the set if it has been clicked N times within that year. The

[1] http://research.microsoft.com/en-us/projects/irc/

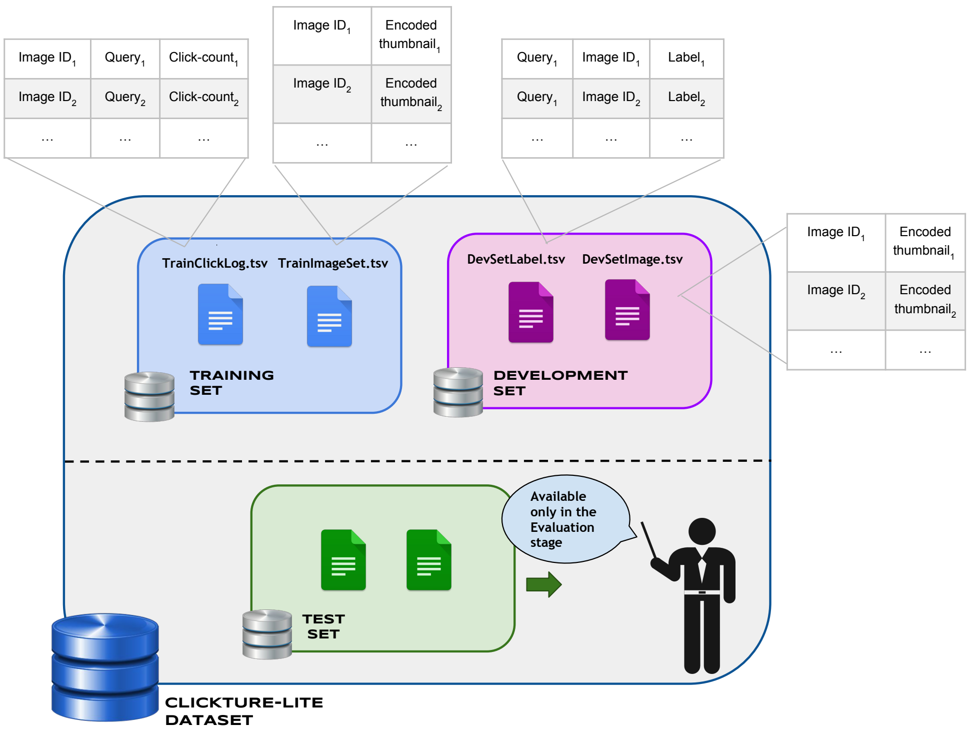


Figure 1.1: *Clickture-Lite dataset structure*

Training Dataset includes 1 million images. This dataset is divided into two files: *TrainCLickLog.tsv* and *TrainImageSet.tsv*.

TrainClickLog.tsv file consists of 23.1 million triads:

$$<\text{Image ID } K \ \langle tab \rangle \ \text{Query } Q \langle tab \rangle \ \text{click count } C >$$

which means the image K was clicked C times in the search results of query Q in one year. Image K is represented by a unique "key" which is hash code generated from the image URL, query Q is a textual word or phrase and query-count C is an integer which is no less than one. One image may correspond to one or more entries in the file. One query may also appear in multiple entries that are associated with different images.

TrainImageSet.tsv file consists of 1 million entries, one for each image of the Training dataset:

$$\text{image ID } \langle tab \rangle \text{ base64 encoded JPEG image thumbnail.}$$

where the thumbnail is processed so that the larger dimension of width and height is at most 300 pixels.

The Development dataset is provided to verify and fine-tune participants algorithm. It is created to have consistent query distribution, judgment guidelines and quality as the Test Dataset, which is used in the evaluation stage. Thus, participants can test their systems using Dev set, when the Test set is not yet available. Development and Test dataset may different in size. The former is divided into two files: *DevSetLabel.tsv* and *DevSetImage.tsv.*

DevSetLabel.tsv file contains 80K triads with 1000 distinct text queries and almost 80K images:

query $\langle tab \rangle$ image ID $\langle tab \rangle$ judgment

where judgment is a label with three different level: *Excellent*, *Good*, *Bad*

DevSetImage.tsv file format (identical to TrainImageSet.tsv format):

image ID$\langle tab \rangle$ base64 encoded JPEG image thumbnail

Also in this case, the thumbnail is processed so that the larger dimension of width and height is at most 300 pixels.

The details of the three datasets are reported in Table 1.1 and Table 1.2.

	Training	**Dev**	**Test 2014**
# of queries	11 M	1 K	1 K
# of images	1 M	79 K	77 K
# of triads	23 M	79 K	77 K

Table 1.1: *Statistics of training, development and testing set*

	#Distinct Queries	**#Distinct unigrams**
Training Dataset	11.701.890	7.174.869
Development Dataset	1.000	4.144

Table 1.2: *Number of unigrams in both Training and Dev Dataset*

Properties of clicked queries and images

Clicked queries are in general relevant to the corresponding image. However, there may be also noises in it, because some images may be clicked by mistake or because they attracted users attention. Queries may also contain typos (i.g: crhistmas instead of Christmas or barak obama, instead of barack obama in Fig.1.3). So, some clicked images may be associated with queries with typos. The query count is a good indicator of confidence or relevance of the query to the image. By grouping similar images into one cluster, it is possible to reduce the noises in the clicked queries. Near-duplicate and similar clicked queries for an image are not merged

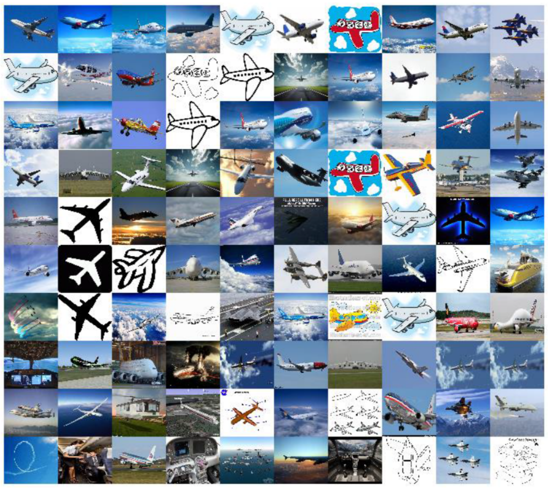


Figure 1.2: *Top 100 clicked images of query "airplane"*

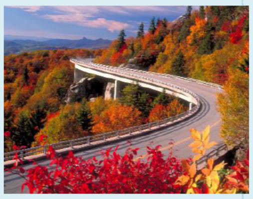
		
fall :113;fall pictures :85;fall leaves :48;fall backgrounds :33;fall images :28;fall foliage :21;fall colors :18;fall pics :16;fall trees :14;autumn images :13	barack obama :414;barak obama :60;barack obama pictures :44;barrack obama :21;presidents :12;pictures of barack obama :3;pictures of barak obama :2;images of barak obama :2;barrak obama :1;barack obama image :1	food :513;food pictures :13;pictures of food :11;food pics :5;picture of a food :4;fast food :3;food images :3;resturant food :2;foood :2;food picture :2

Figure 1.3: *Examples of clicked queries with click counts.*

or removed. Some queries have a large number of clicked images, while a large number of queries (69%) only have one clicked image. It is possible to aggregate all queries with their query counts on the same image key, which is generated from the image URL instead of image content. By comparing the thumbnails or MD5 hashes of the thumbnails we can find exactly duplicate images.

1.2.2 Evaluation metric

In the evaluation stage, participants are asked to download the Evaluation set which contains two files in text format. One is a list of key-query pairs and the other one is a list of key-image pairs. They have the same structure as the Development set files. The teams run their algorithms to estimate a relevance score for each pairs in the first file, and the image content which is a Base$_{64}$ encoded jpeg file, can be found in the second file through the key.

For each query, images are ordered based on the prediction scores. *Discounted Cumulated Gain*

(DCG) is applied to measure ranking results against the manually labelled Development set:

$$DCG_P = \alpha_{25} \sum_{k=0}^{P} \frac{2^{rel_k} - 1}{\log_2(k+1)} \tag{1.1}$$

Where $P = \min(25, \text{number of images})$, $\alpha_{25} = 0.01757$ is a normalizer to make the score equals to 1 for 25 Excellent result, rel_k is graded relevance score of the result in position k in the labelled dataset and $rel_k = Excellent = 3, Good = 2, Bad = 0$. It is the manually judged relevance for each image with respect to the query. The average of the $\mathrm{DCG_S}$ on all the queries is the final evaluation result. In other words, given the scores returned by the system for the whole image list, scores are sorted and the top 25 images are used for measurements. DCG@25 appreciates predicting the "Excellent" labelled images as top-ranked, and gives penalty when the "Bad" labelled images as top-ranked. With $\log_2(k+1)$ as denominator, the higher positions have more influence on the final DCG@25 than the lower positions. In the Development set, many queries have less than 25 Excellent images responding to the query. Thus, the optimal DCG@25 over 1000 queries in Development set is not 1, but it is about 0.68, while in random guess it is 0.47.

1.3 Thesis structure

In Chapter 2, related work about winners and teams that have participated in previous editions of the challenge are reported. We will describe the different methods, the implemented choices they used and the results they obtained.

In Chapter 3, background on information retrieval, image retrieval and text-based retrieval, are described. We report also a brief introduction to Deep Learning and Caffe framework, used for images features extraction task.

In Chapter 4, the proposed approach is discussed. In particular we describe: how we used deep features and how we indexed them; the intersection and union strategies used to sort images; the web-base prototype we developed.

In Chapter 5, the experiments done together with the obtained results are reported.

In Chapter 6, the reached goals of the system are described and some of the possible enhancements and future work are discussed.

Chapter 2

Related work

In this chapter some examples of works of teams that participated to past challenge editions are reported.

In the 2014 edition, six teams successfully submitted results. Table 2.1 shows the evaluation results of top four teams, including the master runs and all runs.
The *Kepler* [1] team combined the deep learning features with the average similarity measurement and page rank. Re-ranking is also applied on the candidate images for a given query. The *Orange* [2] team used search based approach which was also applied in 2013 challenge. They replaced nearest neighbour scoring with support vector machines. The *BUPT* [3] team compared and integrated three typical methods (SVM-based, CCA-based, PAMIR) to conduct the large-scale cross-modal retrieval task. The final output is an aggregation of three methods. Concept level features are utilized. The *MUVIS* system [4] combined a text processing module with a module performing PCA-assisted perceptron regression with random sub-space selection (P2R2S2). (P2R2S2) uses OverFeat (CNN based representation) as a starting point and transforms them into more descriptive features via unsupervised training. The relevance score for each query-image pair is obtained by comparing the transformed features of the query image and the relevant training images.

We also describe the methods and results obtained by the winner team of the 2013 edition.

2.1 Kepler Team [1]

Kepler team participated to the challenge edition of 2014. This team uses the approach applied by [6], in which they use text clustering results to form sets image clusters. The sets image clusters serve as experts to score query-image pairs. When an expert scores a query-image pair, it calculates the similarity between the test image and the top 5 images inside the image

Team	RunID	DCG	Rank (Master)	Rank (All)
Kepler[1]	0	0.56346	1	
	1	0.56608		1
	2	0.56474		
Orange[2]	0	0.53728	2	3
	1	0.53317		
	2	0.50593		
BUPT[3]	0	0.51225	3	4
	1	0.50451		
	2	0.50564		
MUVIS[4]	0	0.51156	4	
	1	0.54629		2
	2	0.50435		

Table 2.1: *Challenging results*

cluster. The key idea of the Orange team's solution is using some images to represent the query, then the relevance problem between text query and the testing image is transformed into the similarity assessment problem between testing image and some training images. This team utilizes the testing images which come before the current one. Instead, Kepler team says that it is important to use other testing images to assess the relevance of query-image pairs. Their observation is that the similar training queries found in the large training click log, only share some words with the testing query, which results the found images in the training set are only partially matched to the testing query. Kepler team utilizes the retrieved images in the training data to represent the text query, so it builds up the relation between text query and testing image through the retrieved images. Other solutions include training some concept classifiers from the training data, and in the testing phase, find the matched concepts and then apply the concept classifiers on the testing image. The prediction scores from the concept classifiers reflect the relevance between the text query and the testing images. This method transform the relevance between query and image into the prediction from concept classifiers trained on the positive images and negative images.

2.1.1 Experiments under online condition

They adapt CNN from ImageNet [7] to the MSR-Bing challenge. They test with some pre-trained models. The first one is DeCaf [8] that has the same structures as Alex's ConvNet: 5 convolutional layers and 3 fully connected layers. From Table 2.2 we can see that features pool5 and fc6 have very similar performance, while fc7 has a bit lower performance than those two.

Compared with the features from fc6, which is the first fully connected layer, the features from VLAD (Vector of Locally Aggregated Descriptors) have much larger dimension, which means much longer time to compute distance or other operations.

feature	**DCG@25**	**Dimension**
pool5	0.5372	6 x 6 x 256
fc6	**0.5379**	4,096
fc7	0.5314	4,096
VLAD	0.5313	98,304

Table 2.2: *Performance of Decaf and VLAD on Dev set*

They also use OverFeat [9] network and they get similar performance on the MSR-Bing dataset. To normalize the output from deep learning layers, they compare L2 normalization with SSR (signed square root) normalization. SSR is applied to the output of fc6 as follows on each dimension of feature x:

$$x_i = sgn(x_i) \cdot \sqrt{|x_i|} \tag{2.1}$$

and then they apply L2 normalization on the feature after SSR.

2.1.2 Experiments under offline condition

For these experiments, they use Caffe [10] toolkit. The assessment between images become more accurate when it can see more images in the testing phase. The image list contains similar images, and they can form some clusters under certain similarity metrics. Page rank works well under this assumption, so they apply it under deep learning features. The relevance score r for the whole image list can be calculated as:

$$r = (\alpha P + (1-\alpha)1^T) \cdot r \tag{2.2}$$

They test the performance under different parameter α.

2.1.3 Timing

In the testing phase it takes 21 min 27s to decode the base64 string into JPEG images and did some preprocessing before feeding into the DCNN. This is done by 60 CPU cores using MapReduce framework. Then 4 GPU run together, each one processes one query at time. It takes 39.7 min to process all the 320.000 query-image pairs. for the query-image pairs assessment, it takes about 10 min, which is extremely fast.

2.2 Orange Team [2]

This team refines the click data by removing some meaningless words and merging queries with duplicate images. Then, they find positive images for each query by click counts. Finally, they train a set of classifiers for different queries in the training set. They use search-based method to find positive examples for a given query and train ad-hoc classifier for it. Positive images can be found comparing their associated clicked queries and the given query. Queries can be treated as tags annotated by users. To measure the similarity between two textual queries, they counting the overlap of them.
For a given topic, different kinds of function can be used to classify the test image, on the basis of the positives found by performing text search. They use two different methods. One is the *nearest neighbour scoring* with a learned visual distance, and the other one is a *cluster-based method.* The latter groups images into cluster by their related queries and scores the relevance of the new test image by comparing it with the learned cluster. It is a little bit slower than the first method, but it is more accurate. The approach that they use is the follow:

- In text search procedure, they use default similarity in Lucene[1] to measure the similarity of two queries.
- A click voting scheme is used to identify positive images based on the result of the text search.
- SVM (Support Vector Machine) trained on positive images and randomly selected negative images is used to classify the tested image.

2.2.1 Searching similar queries and finding positive images using Lucene

Lucene's default text similarity

Lucene is an open-source java library for textual document retrieval. To measure the similarity of two queries, they use lexical overlap. They treat the short query pieces of words as documents, each with a corresponding image and a click count. Lucene uses a modified version of Vector Space Model (VSM) to calculate the similarity of two docs. The scoring function used is:

$$Score(q, q_r) = norm(q) \cdot \sum_{tinq} tf(t) \cdot idf^2(t) \tag{2.3}$$

Each row of the log is a triad (query, image, click count). They take each row as a document and indexed it by query with Lucene. Given a query, it is possible to find its similar queries in index and its corresponding image and click count. Before indexing queries, some pre-processing are needed (stemming and removing stop words).

[1]https://lucene.apache.org

Finding positive images through click voting

After searching with Lucene, they get a list of triads sorted by similarity score of query. They use similarity voting scheme to determine positive images for the given topic, in the following way:

$$Score(p) = \sum_{qinlist} sim(q) \cdot count(q,p) \tag{2.4}$$

where $count(q,p)$ is the click count of query q and image p, and $sim(q)$ is the similarity score given by the last equation. After voting, the top N images with max $Score(p)$ will be reserved as positive samples for subsequent classification procedure.

2.2.2 Image-query relevance assessment

Image representation and scoring using SVM

They extract 192 dimensional CSIFT (Scale Invariant Feature Transform descriptor with color invariant characteristics) descriptors that are mapped into BoW (Bag of Words) vector using a codebook with 1 millions clusters. The codebook is trained on randomly selected images using approximate K-means method described in [11]. They train SVM classifiers, with linear kernel and implemented with Liblinear [12], for each topic. The positive samples are provided by text search procedure, while the negative samples are randomly selected from the training set.

Cluster-based method

Cluster-based method is used to assess the relevance of query-image pair. It can be divided into two stages: off-line and on-line. In *off-line* phase, the query logs in the database are filtered by the Porter Stemming Algorithm. Images are grouped into different categories by their associated queries. Firstly, images are hashed into a lookup table keyed by unigram. Entries form the initial categories. If the number of images in one entry exceeds a given threshold, those images are sub-grouped based on bigram. They generate 10K categories that act as âĂIJexpertsâĂİ to score the test image. In *on-line* phase, they use positive images generated by text search to identify M most related categories and the assess the test images by comparing its visual representation with the categories feature. The query-image pair goes through the "text Search" module. Several images in the database are indexed based on the query term. The image cluster histogram is generated over the initial list to describe the cluster distribution. M maximum bins are selected as the experts to judge the relevance base on the visual similarities between test image and images that the experts have seen. A specific weight is assigned to each expert's judgement:

$$\alpha_m = \beta^{r \cdot m} \cdot \gamma_m \tag{2.5}$$

Where $r \cdot m$ is the rank of expert C_m in the list and β is 0.95. The final relevance score *Rel(Q, I)* between query Q and I is computed as weighted sum:

$$Rel(Q,I) = \sum_{m} \alpha_m \cdot S(I,C_m) \tag{2.6}$$

where $S(I,C_m)$ is the visual relevance between image I and C_m, which is equal the average of visual similarities between I and top-N image in C_m. N is set to 5.
They submit three runs to the challenge as final result. Run 1 and 2 use cluster-method scoring method, while run 3 uses linear SVM to scoring the tested image. Performance of linear SVM is worse that the cluster-based method. The performance of all three runs on final test is very close to that on development set, which can be seen at Table 2.3

Run	**method**	***DCG*$_{25}$**		**time**
		Dev	**Test**	
1	Lucene + Cluster-based	0.529	0.537	5s
2	Lucene + Cluster-based	0.525	0.533	5s
3	Lucene + Linear SVM	0.510	0.506	1s

Table 2.3: *Submitted Runs of Orange Team*

All the evaluation are run on Linux server with 32 cores @ 2.2Ghz and 65 GB RAM. There are some out-of-range words or phrase. In this case neither cluster-based method nor SVM scoring can proceed with these topic. So, they just make random prediction for them.

2.3 Bupt Team [3]

This team compares and integrates three typical methods: SVM-based, CCA-based, PAMIR to conduct large-scale cross modal retrieval task with concept-level visual feature. They study these three methods and propose a fusion strategy. Low level features such as BoW of SIFT, HoG and GIST are extremely used in image classification and retrieval. The unsupervised feature learning with deep structure, e.g. the convolutional neural network (CNN) is capable of extracting visual feature with high level semantic consistency and have achieved higher performance on large scale image classification. So, they train a CNN model on 1119 categories selected from the training dataset, and use probabilities over the 1119 categories output from CNN as the concept level representation of an image.
Given a textual query, they find positive images that are related to the query via the index created with Lucene, and randomly selected negative images irrelevant to the query. A linear SVN classifier is trained on these retrieved relevant and irrelevant training images and is used

to rank the test images. They also use canonical correlation analysis (CCA) that conducts multi-view feature extraction and dimension reduction by cross-modal correlation maximization. CCA is used to maximize the correlation of the co-occurred image-text pairs to learn two linear transformations to project visual and textual features to a comparable feature space. Then, the retrieval can be performed by ranking the similarity calculated by inner product of the projected vectors of image and text. The third kind of methods is the query space projection approach (PAMIR) [13], which projects the visual content into the query space and optimizes a ranking performance-related criterion to project the images to the text space, for the retrieved task. Finally, the retrieved outputs of the three approaches are aggregated as the output of the system.

2.3.1 Preprocessing

1. **Click log Processing** In training dataset, an image may correspond to several textual descriptions and vice versa. There are many spelling mistakes and stop words in descriptions. So they first remove all the words in the stop-word list and only kept the nouns in the description by using the WordNet. All the textual information of an image are collected into one textual description, where the weight of a word is calculated by the sum of the click count of the original queries that contain the word. A threshold τ is used to filter the fused description to get a clean textual description. It can assume three possible values: $\overline{f}_{avg}, \frac{1}{2}\overline{f}_{avg} and 0$, where $\overline{f}_{avg}$ denotes the average weight of all the words in the fused description.

2. **Textual features** With the textual vocabulary obtained from preprocessing stage, they obtain a word indicator or frequency vector for the fused textual description of each image. They define five kinds of textual feature vectors. The first three are those obtained by filtering out words less frequent than the three thresholds. The fourth feature is obtained by only taking stop-words removal and setting the threshold to $\frac{1}{2}\overline{f}_{avg}$. The fifth feature is borrowed from text retrieval for text representation.

3. **Visual features** To describe image contents, two concept-level features are used. The first one is the category probability vector on the 1000 categories of a CNN trained with ImageNet data. The second one is the category probability vector on the selected categories of a CNN trained using the Bing dataset. Top-2000 frequently occurred words are chosen as the categories and their corresponding images are considered positive examples.

2.3.2 Rank aggregation

PAMIR [13] shows better performance than CCA-based approach when the number of query words recognized in training dataset is from one to five. SVM-based approach achieves good performance when training data for the query is abundant. It is chosen when the number of query words is more than 7. The prediction results of three methods are normalized to [-1, 1], then linear ranking-SVM is employed to learn the weight of each model. The performance is shown in Table 2.4.

Conbination	**Performance**
CCA & PAMIR	50.93
CCA & SVM	50.30
PAMIR & SVM	50.92
Linear Ranking-SVM	50.66

Table 2.4: *NDCG@25 of aggregation on Dev dataset(%)*

2.4 MUVIS System [4]

The system contains four decision making modules, which rely on results returned by the text processing module. If the text processing module fails to return anything, they use random guess. The core of the system is a module performing PCA-assisted perceptron regression with random sub-space selection (P^2 R^2 S^2) that is assisted by the three other decision making modules: face bank, duplicate detector, and optical character recognizer. All these modules return a relevance score and a reliability score to each query-image pair. The merging module uses both scores to decide the final output.

2.4.1 Text processing

The text processing module finds the most relevant query texts from the training dataset and returns the images associated with them. This module first convert each query text into unique semantic ID, a set of word stems, in order to merge different forms of the same query into one entry. This procedure includes the following steps:

- Split query text into words and perform speech tagging.
- Lemmatize the words using WordNet engine.
- Remove meaningless words for image retrieval.

If the exact semantic ID is not found in the training dataset, they try to find queries whose semantic ID_s are supersets of semantic ID of the probing query text. If reliable queries are not found, they try to find queries whose semantic ID_s partially overlap with semantic ID of the

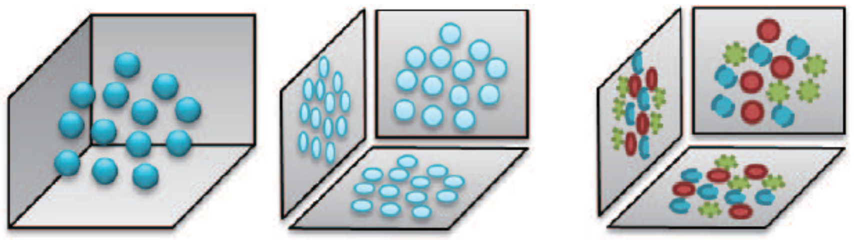


Figure 2.1: *Example of partitioning feature space and data samples. a)Original data b) Sub-feature spaces c) Sample partitioning*

probing query word set. Otherwise, they apply the Hunspell[2] text autocorrection to correct possible typos and do the query search an expansion again.

2.4.2 Features

They use OverFeat [9] convolutional network-based image features extractor and classifier for extracting features from the dataset images. 1000-dimensional output layer of the smaller network has been used as a descriptor fro the dataset images, i.e.,each image is described by its correlation with the ImageNet classes. They first extract OverFeat features from all the training set images. Then they extract OverFeat features from the test image and using $P^2R^2S^2$ module, they transform the features vectors of the test image and the set of associated training images returned by the text processing module.
Low-level features are used to detected whether the query image is a duplicate or near duplicate of a training set images. For this purpose they choose Local Binary Patterns (LBP) and Color Structure Descriptor (CSD) features.

2.4.3 PCA-assisted Perceptron Regression with Random Subspace Selection ($P^2R^2S^2$)

$P^2R^2S^2$ is the core of the system. It forms N different subspaces of the original features spaces, each consisting of D dimensions of the original space. N and D are set so that the resulting feature spaces are overlapping. Each randomly generated new features space is later investigated using principal component analysis (PCA)-assisted perceptron regression. $P^2R^2S^2$ divides also the sample set into smaller partitions to reduce the amount of samples per examination, enable parallelization and increase stability and accuracy of the applied learning method. A simplified example of partitioning feature space and data samples is shown in Figure 2.1.

For each partition, they train a regressor to represent the behaviour of the samples in the corresponding partition. Regressors are trained in a supervised manner, or in other words, for

[2]http://hunspell.sourceforge.net

each sample used in the training of a regressor, a desired output must be presented. Supervision in an unlabelled dataset is possible using PCA. After training, the mean square error is computed ad if it is lower tha a predefined value, training is assumed to converge. They store each converged regressor.
A sample vector of 1000 is first divided into N different vectors of D dimensions, then each D dimensional subvector is propagated through P regressors with outputs of V dimension. So the initial 1000 dimensional vector is transformed into a $DxVxP$ dimensional vector.

They use the L1 distance to compute the similarity of transformed feature vectors of the query image and the given training image. Any example with a click count lower than 2 is discarded, unless those are the only examples at hand. Then they calculate the weighted average of the distances of the closest three vectors, obtained by natural logarithm of the click counts of the corresponding training images. Finally, this average distance is converted to a relevance score using a negative exponential function.

2.4.4 Face Bank and OCR

The objective of face bank module is to enhance the query-image relevance evaluation, when a face is detected in the query image. They train several multi-block LBP based face detectors to obtain pose-invariant face detection. They create a face bank for 2531 well-known celebrities selected from www.pose24.com.
If a face is detected in an image, it is compared with every image in the face bank and a feature vector is formed as a histogram of relevant matches. In the challenge test evaluation, they download the precomputed face feature vectors for the associated images returned by the text processing module and compute the Euclidean distances between the query images and training images. if a match is detected high relevance scores are returned. They also create a duplicate image detector which is used to identify whether the query image is a duplicate or a near duplicate of a relevant image from the training set. They consider images to be near duplicates if the Euclidean distance over LBP and CSD features are below a given threshold Δ

$$D_{LBP}(q_{img},t_{img})\&D_{CSD}(q_{img},t_{img}) \leq \Delta \tag{2.7}$$

where $D_{LBP}(q_{img},t_{img})$ is the Euclidean distance of LBP features vectors extracted from the query image q_img and the training image t img, and similarly $D_{CSD}(q_{img},t_{img})$ is the Euclidean distance od CSD features vectors.

They also perform Optical Character Recognition (OCR) over the ranking images using Tesseract [14], which is a widely used OCR toolbox.

Algorithm Merging Algorithm

```
given: relevance and reliability scores of $P^2R^2S^2$ module, face bank, duplicate image detector and OCR
if duplicate image detector reliability score $\geq$ threshold_1
    return duplicate image detector relevance score
else if face bank reliability score $>$ threshold_2 and face bank relevance score $>$ threshold_3
    return face bank relevance score
else if $P^2R^2S^2$ reliability score $\leq$ threshold_4 and query text is found by OCR
    return OCR relevance score
else
    return $P^2R^2S^2$ relevance score
```

Table 2.5: *Merging Algorithm of MUVIS system*

2.4.5 Merging Results

The merging algorithm assembles the results of all the modules to determine the final relevance score. The relevance evaluations of duplicate image detector, face bank and OCR are exploited only when their reliability score is high, otherwise the system uses the relevance score from $P^2R^2S^2$. Details of merging algorithm are given in Table 2.5.

2.4.6 Overall Results

The DCG scores for different versions of the system over the development and test sets are shown in Table.

	Random	**Master**	**Sub2**	**Sub3**
Dev set	0.4704	0.5099	0.5361	0.5006
Test set	0.4858	0.5116	0.5463	0.5044
OverFeat	OverFeat2	$P^2R^2S^2$	$P^2R^2S^22$	PCA
0.4974	0.5287	0.5082	0.5359	0.4945
0.5037	0.5406	0.5123	0.5473	0.5042

Table 2.6: *Overall results of MUVIS system*

Random score was obtained using random guess only, Master, Sub2 and Sub3 results have been obtained using master, second and third submission. OverFeat results are obtained setting the relevance score according to the L1 distance of OverFeat features of the query image and the closest relevant training set image. For OverFeat2 results they set the relevance score according to L1 distance of OverFeat features of the query image and other test images connected to the same query text. $P^2R^2S^2$ and $P^2R^2S^22$ are similar to master and second submission, but face bank, duplicate image detection and OCR are not used. For PCA results, they replaced $P^2R^2S^2$ with principal component analysis.

2.5 Search-Based Relevance Association with Auxiliary Contextual Cues (2013) [5]

The winner of 2013 MSR-Bing Challenge observed that the provided image-query pairs (e.g., text-based image retrieval results) are usually related to their surrounding text; however, the relationship between image content seems to be ignored. They attempted to integrate the visual information for better ranking results. In addition, they found that plenty of query terms are related to people (e.g., celebrity) and user might have similar queries (click logs) in the search engine. Therefore, they investigated the effectiveness of different auxiliary contextual cues (i.e., face, click logs, visual similarity). Experimental results show that the proposed method can have 16% relative improvement compared to the original ranking results. Especially, for people-related queries, it is possible to have 45.7% relative improvement.
Given a list of image-query pairs, they integrated three different approaches to measure the relevant scores for each image-query pair. The search-based system consists of two parts, images search and tag similarity association. For each image-query pair, similar images are retrieved based on visual content (tag). Then the relevance is measured by the tag (visual) similarities.
They observed that a significant portion of the image-query pairs in Bing dataset comprise faces in images and names in their tags. Therefore, once an image-query pair contains at least a face and a name, the relevance of this query will be measured by the proposed name-to-face correspondence estimation. Another observation is that the provided image-query pairs might be the text-based search results. Relevant images might be the majority in the list corresponding to the tag. The classical PageRank algorithm can solve the problem. The graph is constructed for each query and each image in the list is a node and the weights of the edges are set by the visual similarities. Finally, they integrated the results from each model.

2.5.1 Strategies

Given a tag and a list of images related to that tag, a list a (re-ranked) relevance scores should be returned. Image-query pairs are first fed into three different scoring methods, which are: *content-based method*, *human-based method* and *query association method*. The output scores are fused together as the final relevant scores.

System flow is shown in Fig. 2.2.

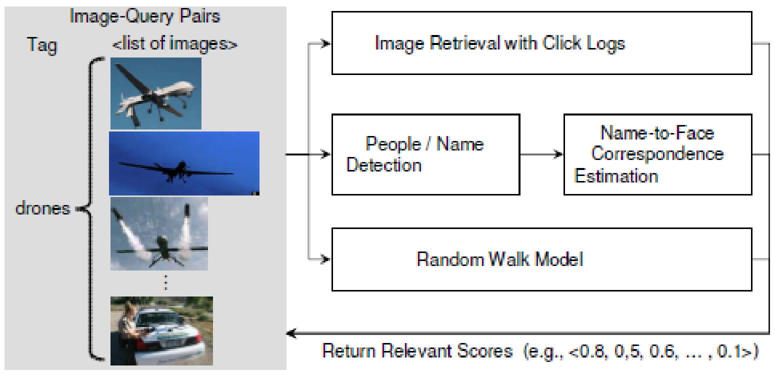


Image-Query Pairs
Tag
<list of images>
drones
Image Retrieval with Click Logs
People / Name Detection
Name-to-Face Correspondence Estimation
Random Walk Model
Return Relevant Scores (e.g., <0.8, 0,5, 0.6, ... , 0.1>)

Figure 2.2: *System diagrams of the proposed method*

2.5.2 Content-based method: Image Retrieval with Click Logs (IRCL)

To search for reference candidates, they thought that the query is relevant to the image, if other visually similar images are associated with similar queries. They retrieved visually and textually similar images from database. In particular, they first retrieved visually similar images (CBIR) and calculated tag similarity from the top ranked images. Similarly, they retrieved textually similar images (TBIR) and then calculated visual similarity from the top-ranked images. To retrieve similar images, they apply the state-of-the-art bag-of-words (BoW) model with 1M vocabulary. First, visual features was extracted for the query image, and similar database images was retrieved under the inverted index structure. They calculated the tag similarity between the query tag and click logs, using the top-ranked visually similar images. They also applied standard pre-processing before calculating the similarity.

For text-based image retrieval (TBIR), they applied tag expansion for query and tag clustering for database. For tags in database images, they applied affinity propagation. For each query tag, first similar tweets from Twitter was retrieved and then extracted top frequently appeared hash tags as the query expansion results.

Textual and visual similarity was combined as the final relevant scores.

$$relevance = \sum_{i \in C} sim(v_q, v_i) \cdot sim(t_q, t_i) \cdot click \tag{2.8}$$

Where q is the query and C are the top ranked images. Similarity are weighted by reliability of candidates (number of clicks).

2.5.3 Human-based method

People/Name detection

As said before, a significant portion of image-query pairs comprise names. They proposed two name detection methods: *name list* and *social media*. The first method consists of collecting a list of celebrity names from *http://www.posh24.com/celebrities/*, denoted by $\mathcal{L}$. For each multiple-word celebrity name in $\mathcal{L}$, the first and the last words of the name are added into the FirstNameSet and LastNameSet. For each new coming tag, if it contains a name in the name list $\mathcal{L}$, they said a name is detected. Unseen names cannot be handled by using this method only. Social media method: if a query tag contains names, it has higher probability that social media website can be found in the title of top-10 related website, such as Linkedin, Twitter and Facebook.

Name-to-Face Correspondence Estimation

Given a set of image-query pairs, face detection and name detection are used to collect training face images and their name annotations. *Identity Bank* consists of 6762 identity models trained by 35092 face images from Bing training dataset. To reduce uncertainties, they randomly sampled more models from Identity Bank for testing and used the multiple test results to confirm the confidence of correspondence measurement.
In Bing development set, around 29.96% image-query pairs are with names never shown in any training image and thus without corresponding models in Identity Bank. The proposed method is called *Face Number* (FN): they measured the consistency between the number of faces in image content and the number of names in query tag, for finding unseen personsâĂŹ name-to-face correspondence. Once a name is detected in a tag, the associated image should comprise at least one face. If more names are detected, more faces are expected to appear in image content. The names are detected via the Name List and Social Media as mentioned before.

2.5.4 Query-association method: Random Walk Model

Images expected by the users usually have similar visual content but are not directly the same as the semantic of the tag. If an image in the list is relevant to the tag, there should be other images with strong visual similarities. In other words, images having strong visual connections with others deserve more relevance scores. Thus, it is reasonable to assign the relevance score to an image based on its similarities with others. The stronger the connections are, the higher the score should be. This is inspired by the PageRank. The relevance scores (s) are formulated as:

$$s = (\alpha P + (1-\alpha)\cdot v1^T)\cdot s \tag{2.9}$$

Where 1 is a vector of one and v is encoded with prior knowledge from previous sections. The transition probability (P) is equivalent to the normalized similarity between image-query pairs:

$$P(i,j) = \frac{sim(i,j)}{\sum_i sim(i,j)} \tag{2.10}$$

The solution is the eigenvector corresponding to the largest eigenvalue.

2.5.5 Experimental setup and results

1. **Dataset and evaluation**. They used the entire dataset of 1M images with 23M click logs. They extracted different kind of visual features: bag-of-words (BoW), vector of locally aggregated descriptors (VLAD), grid color moment (GCM) and local binary pattern (LBP) for facial images. The team evaluated the relevant scores of the same query terms by DCG@25. The averaged DCG@25 of the original ranking and the ideal ranking is 0.469 and 0.684, respectively.

2. **Performance of IRCL**. They evaluated the performance of the entire queries (2.7 (A)) by content-based method that consider both visual and tag similarities. The proposed method IRCLV (0.484) is better than the initial ranking (0.469), because users might have similar types of queries in the search engine. The query might contain the name, so the number of faces in images must also be considered. They improved the accuracy to 0.489 (IRCLV + FN). They also used query tag expansion from hashtags in Twitter, and this approach improved accuracy to 0.495.

Full queries	**Initial**	**IRCLV (+FN)**	**IRCLT**	**Ideal**
DCG@25 (A)	0.469	0.484 (0.489)	0.495	0.684

Name queries	**Initial**	**IB**	**FN**	**IB+FN**	**Ideal**
DCG@25 (N)	0.481	0.496	0.508	0.516	0.702
DCG@25 (I)	0.350	0.496	0.500	0.510	0.6272

Table 2.7: *Performance of content-based method (IRCL) and name-to-face correspondence measurement*

3. **People/Name Detection Results**. With the social media keywords, the recall of detection increases. Facebook provides the largest improvement. Therefore, there are many false positives, but the accounts of Linkedin owned by real people can drop the precision.

4. **Name-to-Face Correspondence Estimation**. They conducted the experiments on the test queries with at least one name detected, totally 7.180 image-query pairs. The baseline for comparison is assigning each query with the same relevance. Relevance measurement by Identity Bank (IB) and by Face Number (FN) report 1.5% improvement and 2.7% compared with the baseline (0.481). they also conducted experiments on the test queries with names included in IB. these test query are more challenging.

5. **Random Walk Model**. First, they measured the relevance directly based on the average similarity of each query images. More similar images the one has, higher relevance score the one deserves. They improved the accuracy by random walk model to 0.543.

Chapter 3

Background

"**Information retrieval** (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)." [15]

Information Retrieval (IR) deals with the representation, storage and organization of textual information, in order to facilitate the user information need. Given a collection of documents, the aim of an IR system is finding information that could be relevant for the user. The IR techniques are not only based on textual data, but have been also applied and extended to multimedia data (images, audio, video). In the 90s, the explosion of the Web has increased the interest in IR. The Web in fact is a huge collection of documents in which users want to make searches. Therefore, Information Retrieval is becoming the principle form of information access.
IR deals with three notions: collection of documents; query (user's information need) that usually does not require formal language knowledge; notion of relevancy, because result set is ordered by rank, which gives a measure of the relevancy of each document.

3.1 Image Retrieval

Recent years have seen a rapid increase in the size of digital image collections. Everyday, both military and civilian equipment generates giga-bytes of images. We cannot access or make use of the information unless it is organized so as to allow efficient browsing, searching, and retrieval. Image retrieval has been a very active research area since the 1970s, with the thrust from two major research communities, database management and computer vision. These two research communities study image retrieval from different angles, one being text-based and the other visual-based. The text-based image retrieval can be traced back to the late 1970s. A very popular framework of image retrieval then was to first annotate the images by text and then use text-based database management systems (DBMS) to perform image retrieval. However, there exist two major difficulties, especially when the size of image collections is large (tens or hundreds of thousands). One is the vast amount of labour required in manual image annotation. The other difficulty, which is more essential, results from the rich content in the images and the subjectivity of human perception. That is, for the same image content different people may perceive it differently. The perception subjectivity and annotation impreciseness may cause unrecoverable mismatches in later retrieval processes.

To overcome these difficulties, content-based image retrieval was proposed. That is, instead of being manually annotated by text-based keywords, images would be indexed by their own visual content, such as color and texture. Since then, many techniques in this research direction have been developed and many image retrieval systems, both research and commercial, have been built. The advances in this research direction are mainly contributed by the computer vision community.

Typically, in content-based retrieval (CBR) approach the search is not performed at the level of the actual digital content, but rather using characteristic features extracted from its content, such as shape descriptors or color histograms in case of images. The feature extraction, however, is not always required: for instance for comparisons of strings or sequences, the content itself is explicitly used in the query. In CBR, an exact match has little meaning, and similarity concepts are typically much more suitable for searching. The problem of similarity search arises in many other unrelated areas such as statistics, computational geometry, artificial intelligence, computational biology, pattern recognition, data mining, etc. For example, user wants to search for London Big Bang (Fig.3.1). He submits an existing picture or his own sketch of Big Bang as query. The system will extract image features for this query and will compare them with that of other images in a database. Relevant results will be displayed to the user.

3.2 Deep Convolutional Neural Network features

In the previous section we said that to overcome the difficulties associated with manual annotation, we can use features extracted from images content.

> *Visual Feature* [16] is a mathematical description of the image visual content that can be used to compare different images, judge their similarity, and identify common content.

Most objects recognition systems tend to use either global image features, which describe an image as a whole, or local features, which represent image patches. Global features have the ability to generalize an entire object with a single vector. Consequently, their use in standard classification techniques is straightforward. Local features, on the other hand, are computed

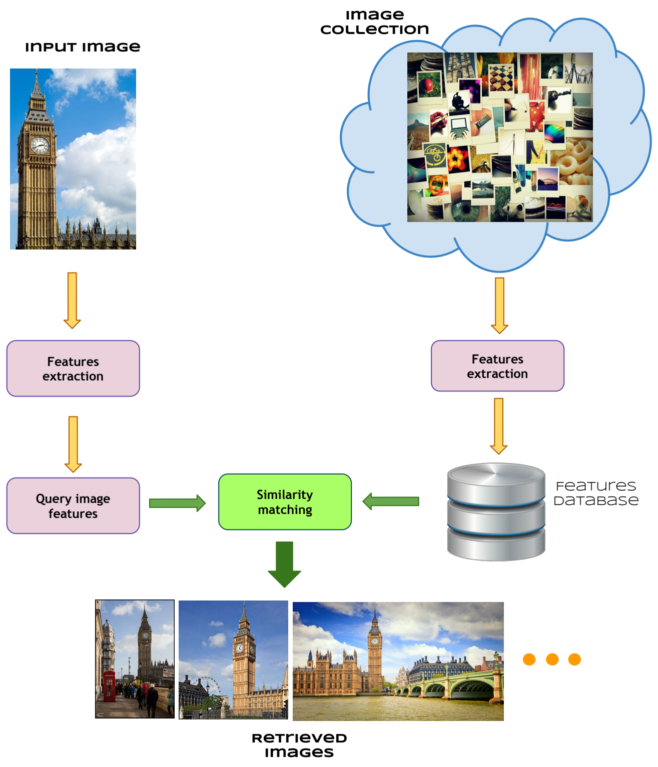


Figure 3.1: *Content Based Image Retrieval example*

at multiple points in the image and are consequently more robust to occlusion and clutter. Feature extraction is the process of generating features to be used in classification task.
In this thesis, we used Deep Convolutional Neural Network features, or simply *deep features*, which are the representation of one of the CNN internal states, when it is excited with a given input (more details are given below). We can briefly say that the "depth" of deep learning models comes from composing functions into a series of transformations from input, through intermediate representations, and on to output. The overall composition gives a deep, layered model, in which each layer encodes progress from low-level details to high-level concepts. This yields a rich, hierarchical representation of the perceptual problem. Figure 3.2 shows the kinds of visual features captured in the intermediate layers of the model between the pixels and the output.

3.2.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) [17] [18] consists of several layers, that can be of three types:

- **Convolutional**: convolution is a mathematical operation which describes a rule of how to mix two functions or pieces of information: (1) the feature map (or input data) and (2) the convolution kernel mix together to form (3) a transformed feature map. Convolution is often interpreted as a filter, where the kernel filters the feature map for information of a

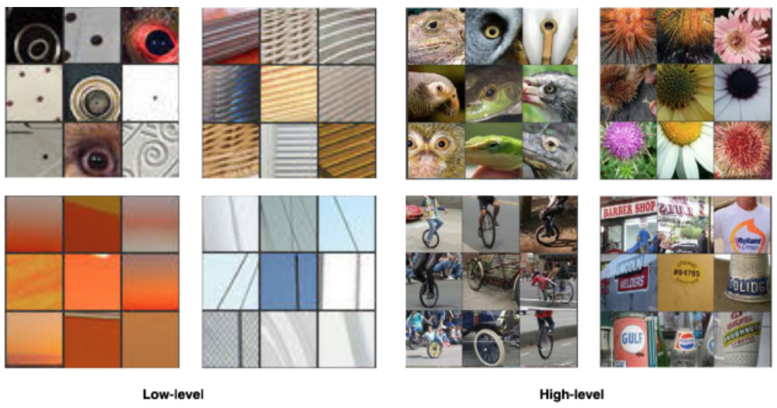


Figure 3.2: *Visualization of deep features by example. Each 3 x 3 array shows the nine image patches from a standard data set that maximize the response of a given feature from a low-level (left) and high-level (right) layer. The low-level features capture color, simple shapes, and similar textures. The high-level features respond to parts like eyes and wheels, flowers in different colors, and text in various styles.*

certain kind (for example one kernel might filter for edges and discard other information). Thus, convolutional layer filters inputs for useful information. This layer has parameters that are learned so that filters are adjusted automatically to extract the most useful information for the given task.

- **Max-pooling**: after each convolutional layer, there may be a pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum. Pooling layers subsample their input.

- **Fully-Connected**: finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. A fully connected layer takes all neurons in the previous layer and connects it to every single neuron it has. There can be no convolutional layers after a fully connected layer.

A Convolutional Neural Network (CNN) consists of one or more convolutional layers (often with a subsampling step) and followed by one or more fully connected layers, as in a standard multilayer neural network. The input of a convolutional layer is a $m \times m \times r$ image where m is the height and width of the image and r is the number of channels (e.g. an RGB image has r=3). The convolutional layer has k filters (or kernels) of size $n \times n \times q$ where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller, and may vary for each kernel. The size of the filters gives rise to the locally connected structure, which are each convolved with the image to produce k feature maps of size mâĹŠ$n+1$. Each map is then subsampled typically with mean or max pooling over $p \times p$ contiguous regions where p ranges between 2 for small images, and is usually not more than 5 for larger inputs.

To summarize, we can say that CNNs are basically just several layers of convolutions with nonlinear activation functions like *ReLU* or *tanh* applied to the results. In a traditional feed-forward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer. In CNNs we do not do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform.

3.2.2 Deep features

In deep learning, *convolutional layers* are exceptionally good at finding good features in images to the next layer to form a hierarchy of nonlinear features that grow in complexity (e.g. blobs, edges -> noses, eyes, cheeks -> faces). Each feature can be thought of as a filter, which filters the input image (for example to find a nose). If the feature is found, the responsible unit or units generate large activations. A unit often refers to the *activation function* in a layer, by which the inputs are transformed via a nonlinear activation function. Usually, a unit has several incoming connections and several outgoing connections. In deep learning, using non-linear activation functions creates increasingly complex features with every layer.

It is possible to extract multiple layers of features. It can be shown mathematically that for images, the best features for a single layer are edges and blobs. To generate features that contain more information we need to transform the first features (edges and blobs) again to get more complex features. In other words, the first layers learn "low-level" features, whereas the last layers learn semantic or "high-level" features. The human brain does exactly the same thing: the first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges and blobs while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.

3.2.3 Deep Learning

Deep Learning [19] [20] is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence (AI). It has become the most popular approach to develop AI machines that perceive and understand the world.

Most machine learning and signal processing techniques have exploited shallow-structured architectures. These architectures typically contain at most one or two layers of non-linear feature transformations. Examples of the shallow architectures are Gaussian mixture models (GMMs), linear or non-linear dynamical systems, support vector machines (SVMs), multi-layer perceptron (MLPs) with a single hidden layer including extreme learning machines (ELMs). Shallow architectures have been shown effective in solving many simple or well-constrained problems, but their limited modelling and representational power can cause difficulties when dealing with more complicated real-world applications involving natural signals such as human speech, natural sound and language, and natural image and visual scenes. Human information processing mechanisms (e.g., vision and audition), however, suggest the need of deep architectures for extracting complex structure and building internal representation from rich sensory inputs. Feed-forward neural networks or multilayer perceptron (MLP) with many hidden layers, which are often referred to as deep neural networks (DNNs), are good examples of the models with a deep architecture (Fig. 3.3).

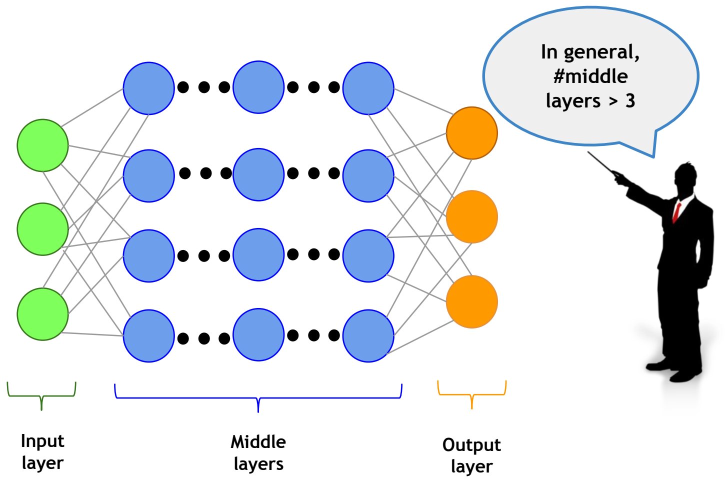


Figure 3.3: *Deep Neural Network*

Deep learning allows the computer to learn a multi-step computer program. Each layer of the representation can be thought as the state of the computer's memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence. Being able to execute instructions sequentially offers great power because later instructions can refer back to the results of earlier instructions.

It is difficult for a computer to understand the meaning of raw sensory input data, such an image represented as a collection of pixel values. The function mapping from a set of pixels to an object identity is very complicated. Deep learning resolves this difficulty by breaking the complicated mapping into a series of nested simple mappings, each described by a different layer of the model. The input is presented at the visible layer, then a series of hidden layers extracts increasingly abstract features from the image (i.g. Fig. 3.4). These layers are called *hidden* because their values are not given in the data; instead the model must determine which

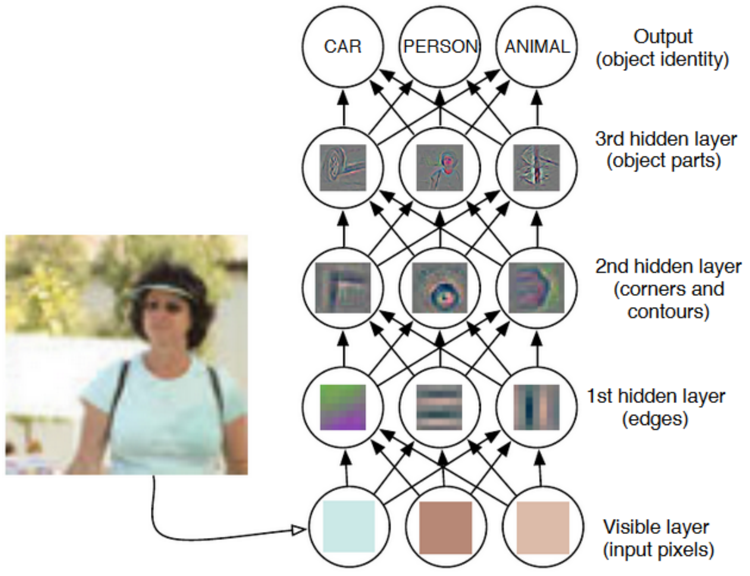


Figure 3.4: *Deep Learning layers example*

concepts are useful for explaining the relationships in the observed data. The images here are visualizations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighbouring pixels. Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image.

Deep learning is in the intersections among the research areas of neural networks, artificial intelligence, graphical modelling, optimization, pattern recognition, and signal processing. Three important reasons for the popularity of deep learning today are:

- the drastically increased chip processing abilities (e.g., general-purpose graphical processing units or GPGPUs);
- the significantly increased size of data used for training;
- the recent advances in machine learning and signal/information processing research.

These advances have enabled the deep learning methods to effectively exploit complex, compositional non-linear functions, to learn distributed and hierarchical feature representations, and to make effective use of both labelled and unlabelled data.
Deep learning uses neural networks (DNNs) many layers deep and large datasets to teach

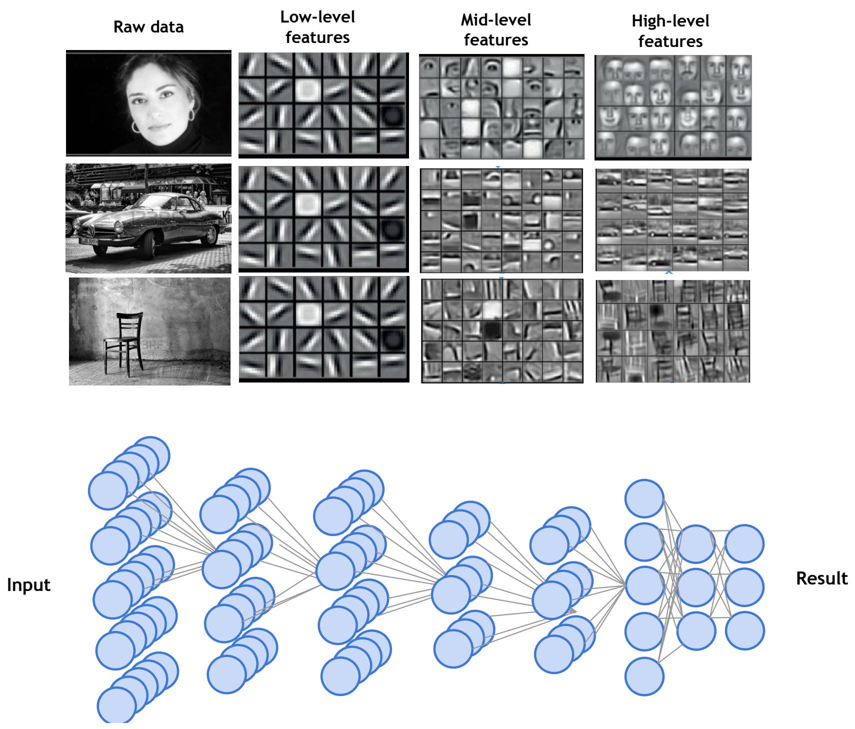


Figure 3.5: *Face and object recognition with DNN*

computers how to solve perceptual problems, such as detecting recognizable concepts in data, translating or understanding natural languages, interpreting information from input data, and more. Deep learning is used in the research community and in industry to help solve many big data problems. Practical examples include vehicle, pedestrian and landmark identification for driver assistance; image recognition; speech recognition; natural language processing; neural machine translation and cancer detection. Deep learning enables a machine to build a hierarchical representation (i.g. Fig. 3.5). The first layer might look for simple edges. The next might look for collections of edges that form simple shapes like rectangles, or circles. The third might identify features like eyes and noses. After five or six layers, the neural network can put these features together. The result: a machine that can recognize faces and objects.

In recent years, it has seen tremendous growth in its popularity and usefulness, due in large part to more powerful computers, larger datasets and techniques to train deeper networks, the availability of faster CPUs, the advent of general purpose GPUs, faster network connectivity and better software infrastructure for distributed computing. The years ahead are full of challenges and opportunities to improve deep learning even further and bring it to new frontiers.

To summarize, deep learning is a type of machine learning, a technique that allows computer systems to improve with experience and data. It achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts and representations, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

3.2.4 How to write application using Deep Learning

There are different framework that can be used to write application using DL (i.g., Caffe, Torch, Theano). In our work, we used Caffe.

Caffe

Caffe[1] (**Convolutional Architecture for Fast Feature Embedding**) [10] is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Centre (BVLC) and by community contributors. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures. The code is written in C++, with CUDA used for GPU computation. The same models can be run in CPU or GPU mode on a variety of hardware: Caffe separates the representation from the actual implementation, and seamless switching between heterogeneous platforms furthers development and deployment. It can even be run in the cloud. Caffe provides a complete tool-kit for training, testing, finetuning, and deploying models. Model definitions are written as config files using the *Protocol Buffer* language. Caffe supports network architectures in the form of arbitrary directed acyclic graphs, and provides reference models for visual tasks, including the landmark "AlexNet" ImageNet [7] model with variations.

Deep networks are compositional models that are naturally represented as a collection of inter-connected layers that work on chunks of data. Caffe defines a net layer-by-layer in its own model schema. The network defines the entire model bottom-to-top from input data to loss. Caffe stores and communicates data in 4-dimensional arrays called *blobs*. Blobs provide a unified memory interface, holding batches of images (or other data), parameters, or parameter updates. The details of blob describe how information is stored and communicated in and across layers and nets. A Blob stores two chunks of memories, data and diff. The former is the normal data that we pass along, and the latter is the gradient computed by the network.

The layer comes next as the foundation of both model and computation. The net follows as the collection and connection of layers. A *layer* is the essence of a neural network layer: it takes one or more blobs as input, and yields one or more blobs as output. In other words, it takes input through bottom connections and makes output through top connections. Layers

[1]http://caffe.berkeleyvision.org/

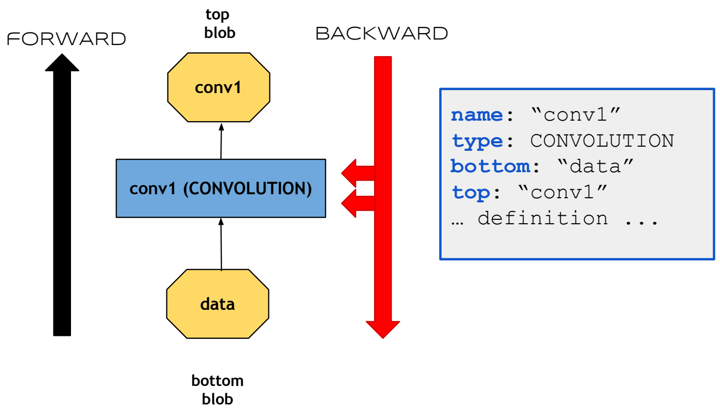


Figure 3.6: *Simple example of Caffe network*

have two key responsibilities for the operation of the network as a whole: a forward pass that takes the inputs and produces the outputs, and a backward pass that takes the gradient with respect to the output, and computes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers. Caffe provides a complete set of layer types including: convolution, pooling, inner products, non-linearities like rectified linear and logistic, local response normalization, element-wise operations, and losses like softmax and hinge. These are all the types needed for state-of-the-art visual tasks. Coding custom layers requires minimal effort due to the compositional construction of networks.

The net is a set of layers connected in a computation graph - a directed acyclic graph (DAG) to be exact. A typical network begins with a data layer that loads from disk and ends with a loss layer that computes the objective for a task such as classification or reconstruction. Fig. 3.6 shows a simplified example of Caffe network, where blue box represents a layer and yellow octagons represent blobs.
Blobs and layers hide implementation details from the model definition. After construction, the network is run on either CPU or GPU.

The models are defined in plaintext protocol buffer schema (*prototxt*) while the learned models are serialized as binary protocol buffer (*binaryproto*) *.caffemodel* files. Models are saved to disk as Google Protocol Buffers because has the following important features: minimal-size binary strings when serialized, efficient serialization, a human-readable text format compatible with the binary version, and efficient interface implementations in multiple languages, most notably C++ and Python. Large-scale data is stored in LevelDB or LMDB databases. LevelDB is an open source on-disk key-value store written by Google. It is not a NoSQL database, so it does not have a relational data model and it does not support SQL queries. Also, it has no support for indexes. Applications use LevelDB as a library, as it does not provide a server or command-line interface.

Caffe can be used for several applications, such as object classification, features extraction from images using a pre-trained network, object detection, scene recognition and so on. Caffe has an online demo [2] showing state-of-the-art object classification on images provided by the users, including via mobile phone. The demo takes the image and tries to categorize it into one of the 1.000 ImageNet categories. The framework can also be used to extract semantic features from images using a pre-trained network. Figure 3.7 shows a two-dimensional embedding of all the ImageNet validation images, colored by a coarse category that they come from. The nice separation testifies to a successful semantic embedding.

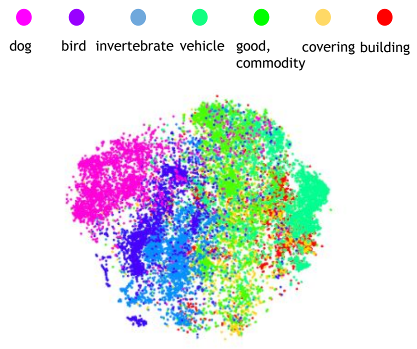


Figure 3.7: *Features extracted from a deep network, visualized in a 2-dimensional space*

3.2.5 ImageNet Classification with Deep CNN

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. The immense complexity of the object recognition task means that this problem cannot be specified even by a dataset very large, so the model should also have lots of prior knowledge to compensate for all the data we do not have. Convolutional neural networks (CNNs) constitute one such class of models. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images. Compared to standard feed-forward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.
In a CNN, the key computation is the convolution of a feature detector with an input signal. Convolution with a collection of filters, like the learned filters in Figure 3.8 , enriches the representation: at the first layer of a CNN the features go from individual pixels to simple primitives like horizontal and vertical lines, circles, and patches of color. In contrast to conventional

[2]http://demo.caffe.berkeleyvision.org/

single-channel image processing filters, these CNN filters are computed across all of the input channels. Convolutional filters are translation-invariant so they yield a high response wherever a feature is detected.

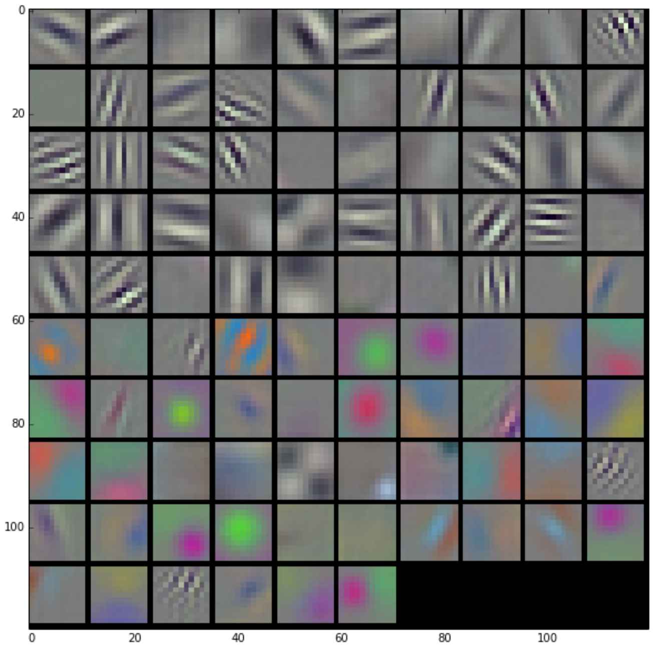


Figure 3.8: *The first layer of learned convolutional filters in CaffeNet, the Caffe reference ImageNet model based on AlexNet by Krizhevsky et al. These filters are tuned to edges of different orientations, frequency, and phase and colors. The filter outputs expand the dimensionality of the visual representation from the three color channels of the image to these 96 primitives. Deeper layers further enrich the representation.*

The first work that popularized Convolutional Networks in Computer Vision was the *AlexNet*, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a similar architecture basic as LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer immediately followed by a POOL layer).

ImageNet

ImageNet is a dataset of over 15 million labelled high-resolution images belonging to roughly 22.000 categories. The images were collected from the web and labelled by human labellers using Amazon's Mechanical Turk crowd-sourcing tool. Starting in 2010, an annual competition called the *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) has been held. ILSVRC uses

a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50.000 validation images and 150.000 testing images.
The images are down-sampled to a fixed resolution of 256x256. Given a rectangular image, it is first rescaled such that the shorter side was of length 256, and then cropped out the central 256x256 patch from the resulting image. The images are not pre-processed in any other way, except for subtracting the mean activity over the training set from each pixel. The network is trained on the (centred) raw RGB values of the pixels.

Network Architecture

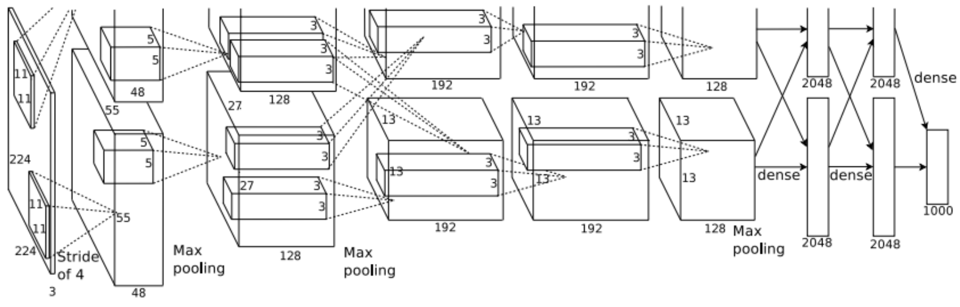


Figure 3.9: *An illustration of the AlexNet architecture, explicitly showing the delineation of responsibilities between the two GPU_s. One GPU runs the layer-parts at the top of the figure while the othen runs the layer-parts at the bottom. The GPU_s communicate only at certain layers.*

The net (Fig.3.9) contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. There are different types of layers:

- **Input** that holds the raw pixel values of the image.
- **CONV** that computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and the region they are connected to in the input volume.
- **RELU (Rectified Linear Unit)** layer applies a non-saturating non-linearity activation function, such as the $f(x) = max(0,x)$ thresholding at zero. Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units.
- **POOL** layer summarizes the outputs of neighbouring groups of neurons in the same kernel map. It can be thought of as consisting of a grid of pooling units spaced s pixels apart,

each summarizing a neighbourhood of size $s \times s$ centred at the location of the pooling unit.

- **FC (Fully-Connected)** layer computes the class scores. As with ordinary Neural Networks and as the name implies, the neurons in the fully-connected layers are connected to all neurons in the previous layer.

The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU. The kernels of the third convolutional layer are connected to all kernel maps in the second layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers, follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.
The most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations. Two distinct forms of data augmentation are used, both of which allow transformed images to be produced from the original images with very little computation. The first form of data augmentation consists of generating image translations and horizontal reflections. This is done by extracting random 224x224 patches (and their horizontal reflections) from the 256x256 images and training the network on these extracted patches. The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, they perform PCA on the set of RGB pixel values throughout the ImageNet training set.

Combining the predictions of many different models is a very successful way to reduce test errors but it is too expensive for big neural networks. There is a new technique called "dropout" that consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. Dropout is used in the first two fully-connected layers. Without dropout, the network exhibits substantial overfitting. Dropout roughly doubles the number of iterations required to converge.

3.3 Text-Based Retrieval

One of the most important tasks of an information retrieval system is to return, in response to a user query generally expressed as a set of terms, the subset of documents which best matches the information need expressed by the user. Text matching is, beyond doubt, the most common technique used to accomplish this document retrieval task; the search is performed by looking for (and returning to user) the documents which contain the set, or a subset, of the terms

included in the query. The way to avoid linearly scanning the texts for each query is to index the documents in advance.
Terms are the indexed units, that are usually words. By *documents* we mean whatever units we have decided to build a retrieval system over. An *information need* is the topic about which the user desires to know more, and is different from a *query*, which is what the user conveys to the computer in an attempt to communicate the information need. A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need.

Text-document retrieval is based on a similarity match between the terms that make up a document and those used in an information request - the query - to describe the information need of the user. Most Internet search engines support queries that range from the single keyword to combinations of some limited number of them.

Most IR systems support a variety of query forms, including: *Boolean queries*, in which the search terms are connected by the Boolean operators AND, OR, NOT; *term proximity*, specified by surrounding several terms by quotes, or a statement of nearness; *natural language*, where the query is formed as 'normal' text. In this case, there is no length limitation, so that a source document could be used as a query.
Documents that match the query search terms are sorted or ranked according to their assumed relevance to the query. This requires an assumed/calculated weighting of terms in both the query and the document result set. Commonly, terms are weighted according to their frequency of occurrence in the document.

The first major concept in IR is the **inverted index**, also referred to as postings file or inverted file. It is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. We keep a dictionary of terms (sometimes also referred to as a vocabulary or lexicon). To gain the speed benefits of indexing at retrieval time, we have to build the index in advance. This inverted index structure is essentially without rivals as the most efficient structure for supporting ad-hoc text search.

3.3.1 Ranking in Retrieval System

Early IR systems were boolean systems which allowed users to specify their information need using a complex combination of boolean ANDs, ORs and NOTs. Boolean systems have several shortcomings, e.g., there is no inherent notion of document ranking, and it is very hard for a user to form a good search request. In fact users have to provide sufficient syntactical restrictions in their query to limit the number of documents retrieved, and those retrieved documents are

not ranked in order of any relationship to the user's query. These systems are good for expert users with precise understanding of their needs and the collection. However, most everyday users expect IR systems to do ranked retrieval. IR systems rank documents by their estimation of the usefulness of a document for a user query.
The ranking approach to retrieval seems to be more oriented toward end-users, because it allows them to input a simple query such as a sentence or a phrase and retrieve a list of documents ranked in order of likely relevance. This method eliminates the often-wrong Boolean syntax used by end-users, and provides some results even if a query term is incorrect, that is, it is not the term used in the data, it is misspelled, and so on. The ranking methodology also works well for the complex queries that may be difficult for end-users to express in Boolean logic.
Most IR systems assign a numeric score to every document and rank documents by this score. Several models have been proposed for this process. The three most used models in IR research are the vector space model, the probabilistic models, and the inference network model.

tf-idf weighting

We need a way of assigning a score to a query/document pair, that measures how well document and query match. A document that mentions a query term more often has more to do with that query and therefore should receive a higher score. A plausible scoring mechanism then is to compute a score that is the sum, over the query terms, of the match scores between each query term and the document.
We assign to each term in a document a weight, that depends on the number of occurrences of the term in the document. We would like to compute a score between a query term t and a document d, based on the weight of t in d. The simplest approach is to assign the weight to be equal to the number of occurrences of term t in document d. This weighting scheme is referred to as *term frequency* and is denoted $\mathrm{tf}_{t,d}$, with the subscripts denoting the term and the document in order.
The exact ordering of the terms in a document is ignored but the number of occurrences of each term is material (in contrast to Boolean retrieval). We only retain information on the number of occurrences of each term. This model is known in the literature as *bag of words*.
We have to understand that relevance does not increase proportionally with term frequency, because words in a document are not equally important. Rare terms are more informative than frequent terms, whereas frequent terms (with respect to the collection) are less informative than rare terms. If we consider a query term that is frequent in the collection, then a document containing such a term is more likely to be relevant than a document that does not contain it. But it is not a sure indicator of relevance. We define the *document frequency* df_t as the number of documents in the collection that contain a term t and it is an inverse measure of the

informativeness of t. To estimate the rarity of a term in the whole document collection, we use the *inverse document frequency* defined as:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t} \tag{3.1}$$

where N is the total number of documents in the collection. Thus the *idf* of a rare term is high, whereas the *idf* of a frequent term is likely to be low.
Now, we can combine the definition of term frequency and inverse document frequency, to produce a composite weight for each term in each document. The *tf-idf* is the best known weighting scheme in IR, and assigns to term t a weight in document d given by

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{3.2}$$

We introduce the *score measure*: the score of a document d is the sum, over all query terms, of the tf-idf weight of each term in d:

$$\mathrm{Score}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d} \tag{3.3}$$

Vector Space Model

The tf-idf values can now be used to create vector representations of documents. Each document is now represented as a real-valued vector of tf-idf weights $\in R^{|V|}$, so we have a $|V|$-dimensional real-valued vector space. Terms are axes of the space and documents are points or vectors in this space. Dictionary terms that do not occur in a document are weighted zero. We might think to do the same for queries: represent them as vectors in the high-dimensional space and then rank documents according to their proximity to the query (or similarity of vectors).
To assign a numeric score to a document for a query, the model measures the similarity between the query vector and the document vector. To quantify the similarity between two documents in vector space, we might consider the Euclidean distance, or in other words the magnitude of the vector difference between two document vectors. Since documents are usually not of equal length, simply computing the difference between two vectors has the disadvantage that documents of similar content but different length are not regarded as similar in the vector space. To compensate for the effect of different document lengths, a common way to compute the similarity of two documents is using the *cosine similarity* measure. Typically, the angle between two vectors is used as a measure of divergence between the vectors, and cosine of the angle is used as the numeric similarity (since cosine has the nice property that it is 1 for identical vectors and 0 for orthogonal vectors). Rank documents according to the angle between query and document in decreasing order is equivalent to rank documents according to cosine(query,document) in increasing order.
If all the vectors are forced to be unit length, then the cosine of the angle between two vectors

is same as their dot-product. We can (length-) normalized a vector by dividing each of its components by its length. We use the $L2$ norm:

$$\parallel x \parallel_2 = \sqrt{\sum_i x_i^2} \tag{3.4}$$

As a result, longer documents and shorter documents have weights of the same order of magnitude. However, not all words are created equally. Some are more important than others when computing similarity. Rather than use the count or the presence/absence of each term, we can use a weight. For example, we can give a lower weight to common words. What would make a suitable weighting? tf-idf of course. Putting this altogether, we can define the *cosine similarity* between query and documents:

$$\cos(\vec{q},\vec{d}) = sim(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{\mid \vec{q} \mid \cdot \mid \vec{d} \mid} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sum_{i=1}^{|V|} q_i^2 \sum_{i=1}^{|V|} d_i^2} \tag{3.5}$$

where: $\mathrm{q_i}$ is the tf-idf weight of term i in the query; $\mathrm{d_i}$ is the tf-idf weight of term i in the document; $\mid \vec{q} \mid$ and $\mid \vec{d} \mid$ are the lengths of $\vec{q}$ and $\vec{d}$. The numerator represents the dot product (also known as the inner product) of the two vectors, while the denominator is the product of their Euclidean lengths. For length-normalized vectors, cosine similarity is simply the *dot product* (or scalar product):

$$\cos(\vec{q},\vec{d}) = sim(\vec{q},\vec{d}) = \sum_{i=1}^{|V|} q_i d_i \tag{3.6}$$

The resulting scores can then be used to select the top-scoring documents for a query.

For the purpose of ranking the documents, the query normalization does not change the relative order of the documents. Query normalization factor is constant for the query, so we can ignore it when we calculate the score.

Chapter 4

Proposed Approach to MSR-Image Retrieval Challenge

In this chapter, the system built to solve the MSR-Bing Image Retrieval Challenge is presented and described. The two strategies, we proposed to retrieve candidates, are reported. The architecture of the system is shown. Our Web-based prototype is described.

4.1 Overview

The main goal of our system is to measure the relevance of web images and the query given in text form, as shown in figure 4.1. That is, given a query and a list of related images, we need to measure the relevance of the images according to the textual query.

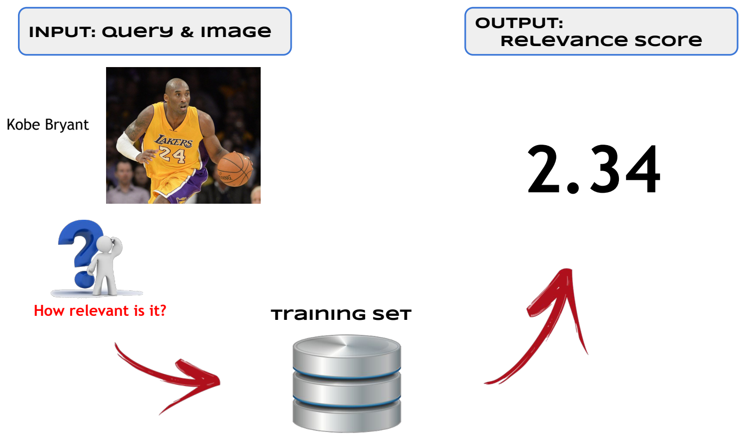


Figure 4.1: *The main goal of the system*

During the official execution of the competition, a Test set is provided by the organization, to test the developed algorithms. Since the Test set it is not available (it was made available to users only when the challenge started), our work has been done using only the *Development set*, which though may differ in size, is created to have consistent query distribution, judgement guidelines and quality as the Test Dataset. This dataset allows us to verify and fine-tune our algorithm, and we need to assume we do not know the queries and images contained in it. The Development test is just used to validate the developed algorithms, and it is not used for training.

The Development set also contains a ground truth, that can be used to evaluate the quality of the developed algorithms. In this ground truth, every pair (query text, image) is associated with one of the possible relevance levels: *Excellent*, *Good*, and *Bad*. For each image-query pair, similar images are retrieved based on visual and textual content. In other words, the query is relevant to the image, if other visually similar images are associated with similar queries.
The challenge aims to predict scores for each image-query pair individually. The most intuitive way to measure the relevance score of each pair is to retrieve visually and textually similar images from the database. We proposed two strategies: intersection and union of candidates results.

4.1.1 Two strategies for sorting images according to relevance to a text query

Given an image-query pair, the text query is used to perform text-based image retrieval, while the image is used to perform content-based image retrieval. These procedures, allowed us to create two different sets of triplets, *textual* and *visual* candidates sets. Each triplet was retrieved from the Training set and, as explained in Section 4.5, has the following structure: $< image,\ text\ query,\ click\ count >$. To combine them, we proposed two strategies: intersection and union.

The *intersection* strategy was implemented in the following way: we intersected the two retrieved sets, we provided the relevance scores based on the visual and textual similarities and we ranked the results.
The *union* strategy can be divided in two parts, text and visual search. Given the text query, we first retrieved textually similar images and then we calculated visual similarity from the top-ranked images. Similarly, given the image, we retrieved visually similar images and calculated textual similarity from the top-ranked images. Finally, we combined the results of these two parts making their union, we provided a relevance score for each image in the final set and we ranked the results.

Since the Training set contains 1 million images and about 23 million queries, searching efficiently through all data is quite challenging. Scanning the whole dataset in a sequential way, for each image-query pair of the Development set, to search for similar images or queries is not a good idea. It is a very time-consuming operation. Thus, we have to find an efficient way to perform text-based and content-based image retrieval. To this aim, we can use an index. In particular, we used Apache Lucene, an open-source full-text search library.

The full-text search based on Lucene index has different advantages: is fast, flexible, reliable and scales very well; it makes possible to quickly retrieve data; it allows to search in a very precise way and ranking results based on a relevance score; it is an open source software. Since Lucene, as any information retrieval system, supports similarity-based retrieval, we would like to extend it to support image similarity without building a new information retrieval system from scratch. In order to employ Lucene, we must: 1) creating a Lucene index and 2) parsing the query and looking up the pre-built index to answer the query. The biggest challenge was to find a strategy that allows us to transform the image feature in textual form in order to exploit the invert index of Lucene. In this way, we are able to set up a robust retrieval system that combines full-text search with content-based image retrieval capabilities. Therefore, we define a textual representation for each visual feature array.

4.2 Measure of similarity

When we perform a content-based query, we retrieve the most similar images and queries to the given image-query pair, based on similarity function. This function measures how well indexed documents and query match. A variety of similarity or distance measures have been proposed and widely applied. We compared two of them: *euclidean distance* and *cosine similarity.*

Euclidean distance

Euclidean distance is a standard metric and can be easily measured. Given two documents d_a and d_b represented by their term vectors $\overrightarrow{t_a}$ and $\overrightarrow{t_b}$ respectively, the Euclidean distance of the two documents is defined as:

$$Euclidean\ distance = E(\overrightarrow{t_a}, \overrightarrow{t_b}) = \sqrt{\sum_{i=1}^{m} \mid w_{t,a} - w_{t,b} \mid^2} \tag{4.1}$$

where the term set is $T = \{t_1,\ ...,\ t_m\}$. We use the $tf \cdot idf$ value as term weights, that is $w_{t,a} = tf_t \cdot idf_{t,d_a}$.

In other words, euclidean distance is the square root of the sum of squared differences between corresponding elements of the two vectors, that in out case are the tf-idf weights.

Cosine similarity

Cosine similarity is one of the most popular similarity measure applied to text documents, in many information retrieval applications and clustering. The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude. It can be seen as a comparison between documents on a normalized space because we are not taking into the consideration the magnitude of the vectors, but only the angle between the vectors. Given two vectors $\overrightarrow{t_a}$ and $\overrightarrow{t_b}$, the cosine similarity between them is:

$$cosine\ similarity = cos(\overrightarrow{t_a}, \overrightarrow{t_b}) = \frac{\overrightarrow{w_{t,a}} \cdot \overrightarrow{w_{t,b}}}{\parallel \overrightarrow{w_{t,a}} \parallel \cdot \parallel \overrightarrow{w_{t,b}} \parallel} \tag{4.2}$$

where $\overrightarrow{w_{t,a}}$ is the entry of the vectors, which, in case of a textual information retrieval system, represents the tf-idf weight of term t in the document that is represented as a vector t_a; similarly for $\overrightarrow{w_{t,b}}$.

4.2.1 Euclidean distance vc Cosine similarity

To quantify the similarity between two documents in Vector Space, we might consider the Euclidean distance (eq. 4.1), or in other words the magnitude of the vector difference between two document vectors. Since documents are usually not of equal length, simply computing the difference between two vectors has the disadvantage that document of similar content but different length are not regarded as similar in the vector space, as shown in figure 4.2. Contrariwise, an important property of the cosine similarity is its independence of document length. For example, combining two identical copies of a document d to get a new pseudo document d', the cosine similarity between d and d' is 1, which means that these two documents are regarded to be identical. In other words, documents with the same composition but different totals will be treated identically.
However, we have to note that this measure tends to ignore the higher term count on documents. Suppose we have a document with the word "sky" appearing 200 times and another document with the word "sky" appearing 50 times. The Euclidean distance between them will be higher but the angle will still be small because they are pointing to the same direction, which is what matters when we are comparing documents.

In our work, to measure the similarity of two images, we compared two methods: cosine similarity applied to the textual representation of images features and Euclidean distance applied to sequential search. In the first method, we converted the visual features vectors into string text, that could be indexed by means of the Lucene search engine library, as described in section 4.3. Using this approach, given an image query, we could easily find similar images through

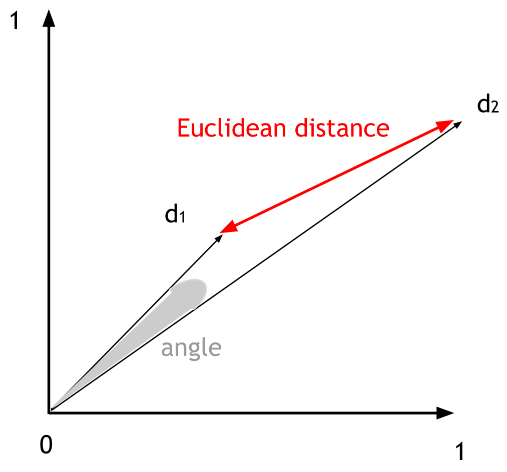


Figure 4.2: *The euclidean distance between d_1 and d_2 is large even if the distributions of terms in the two documents are very similar.*

the cosine similarity between features textual representation. Also in the second method, we used the visual features extracted with Caffe. For each image, we considered a descriptor that contained the visual features and an id. Then, the descriptors were loaded in memory. Thus, given a query image, we computed the Euclidean distance between its visual features and the features contained in all descriptors. We applied *K-nearest Neighbours* (kNN) search and we obtained the K most similar images to the query image.

To decide which of the two measures was better, we used the *Mean Average Precision*. We know that the two most frequent and basic measures for information retrieval effectiveness are *Precision* and *Recall*. The former is the fraction of retrieved documents that are relevant to the user's information need, and the latter is the fraction of relevant documents to the query that are retrieved. However, precision and recall are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable also to consider the order in which the returned documents are presented. Thus, by computing a precision and recall at every position in the ranked sequence of documents, one can plot a precision-recall curve, plotting precision P(r) as a function of recall r. The area under the precision-recall curve can be used as an evaluation measure, called *Average Precision*, that is the average of the precision value obtained for the set of top k documents existing after each relevant document is retrieved:

$$Average\ Precision = AvgP = \frac{\sum_{k=1}^{n} P(k) \cdot rel(k)}{Number\ of\ relevant\ documents} \tag{4.3}$$

where n is the number of retrieved documents, $P(k)$ is the precision at k in the list, $rel(k)$ is an indicator function equals to 1 if the item at rank k is a relevant document, 0 otherwise. The average is over all relevant documents and the relevant documents not retrieved get a precision score of zero.

The *Mean Average Precision* for a set of queries is the mean of the average precision scores for each query:

$$Mean\ Average\ Precision = MAP = \frac{\sum_{q=1}^{Q} AvgP(q)}{Q} \tag{4.4}$$

where Q is the number of queries.

Thus, we can say that MAP summarize rankings from multiple queries by averaging the average precision, and it is the most commonly used measure in research papers.

4.3 Using text retrieval engine to index visual deep features

As mentioned in Chapter 3, features extraction is the basis of CBIR. In this work, we used *deep learning features*, that have more compact representation than the traditional visual features, and are efficient with very good performance. Deep learning has achieved state-of-the-art performance on many different applications in computer vision and multimedia, so we adapted the Convolutional Neural Network from ImageNet to the MSR-Bing challenge. It is a deep CNN architecture proposed by Krizhevsky et al. [7], which won the ImageNet Large Scale Visual Recognition Challenge 2012 (LSVRC) (section 3.2.5) . To extract visual features from training images, contained in *TrainClickLog.tsv*, we used Caffe framework as said in Chapter 3.

To create a Caffe model we need to define the model architecture in a protocol buffer definition file (*prototxt*), while the learned models are serialized as binary protocol buffer (*binaryproto.caffemodel*) files. In our work, we used a Caffe pre-trained model:

models/ bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel

Data enters Caffe through data layers: they lie at the bottom of nets. Common input preprocessing such as mean subtraction, scaling, random cropping and mirroring, are available.
We subtracted the mean image of the ILSVRC dataset from our dataset, because it significantly improves classification accuracies. We used *ImageDataLayer*, which load and resize images for us. In particular, firstly it ignores the image's original aspect ratio and warps it to 256×256, rather than resizing and cropping to preserve the proportions. Secondly, the features are taken on the center 224×224 crop of the 256×256 resized images. This layer requires a source and a batch_size. The source is the name of a text file which contains a list of files to process. In particular, each line contains an image filename and a label, that is 0 in our case. So we created

a file containing all Clickture training set images, that are 1 million. The batch size represents the number of images that the net processes in each run. We set this value to 1.
We had also to define the number of data mini-batches and the name of features blob that we extracted. The former represents the number of iterations or in other words the number of images read by the data layer, and we set this value to 1 million. The latter was set to fc6. We have chosen to extract features from this layer, that indicates the first fully connected layer (the 6-th layer in the whole Alex's convnet Fig. 3.9), because features fc7, the second fully connected layer, has a bit lower performance than the previous one. Even if features fc6 and pool5 (5-th convolutional layer) have similar performance, we noticed that the fc6 layer has more compact representation. In fact, the output of fc6 layer has only 4096 dimensions, while pool5 has 9216 dimensions.

The extracted features were stored to LevelDB database, but Caffe serialized them using Google Protobuf. To get the values, we had to deserialize them. We had to use protobuf compiler to compile the file *Caffe.proto*, that returned *Caffe.java* and *Caffe.cpp* files. Since we used Java code, we considered only the first file. Thus, iterating on the LevelDB database, we could keep float arrays of 4096 elements, that represent our features. We observed that about 80% of them was equal to zero. We applied L2 normalization or euclidean distance to each array.

$$x_i = \frac{x_i}{\sqrt{\sum_{j=1}^{n} x_j^2}}$$

where $n = 4096$.

4.3.1 Representing deep features as text

In order to perform the CBIR process, in principle, we should compare the features extracted from the query with all the features extracted from the images of the dataset and take the nearest images according to L2 distance. However, if the dataset is large, as in our case (one million images), this procedure can become time-consuming. For this purpose, as explained above, we have created an index to efficiently access the features. The basic idea is to exploit the sparsity of the deep feature vectors, which contain mostly zeros (about 90%). Moreover, it is easy to see that if two vectors x and y have length equal to 1 (such as in our case), the following relationship between the Euclidean distance $d_2(x,y)$ and the dot product $x * y$ exists:

$$d_2(x,y)^2 = 2(1 - x * y)$$

The advantage of formulating the L2 distance in terms of dot product is that allows us to efficiently exploit the sparsity of the vectors by accumulating the product of non-zeroes entries

in the vector x and their corresponding non-zeros entries in the vector y. To this end, we have exploited the inverted index already present in a text search engine.
Most text search engine, including Lucene, use the Vector Space Model to represent text, in which a text document is represented as a vector of terms each associated with the number of occurrences of the term in the document. These search engines are suitable for applications that require full-text search abilities. Therefore, we converted image features into a textual form. This conversion allowed us to employ the Lucene's off-the-shelf indexing and searching abilities with a little implementation effort. In this way, we were able to set up a robust retrieval system that combines full-text search with content-based image retrieval capabilities.

Lucene, as other search engines, computes the similarity between documents using the cosine similarity, which can be seen as dot product of the two vectors divided by their lengths product. Therefore, in our case, cosine similarity and dot product are the same. The idea now is to force, in some way, Lucene to fill the vector of its internal inverted index with the entries of the deep feature vectors. If a query feature transformed in this way is submitted to Lucene engine, we receive as result a ranked list of documents (i.e., the images) sorted by decreasing values of the cosine similarity, which means increasing values of the L2 and that is exactly what we wanted. For space-saving reasons, however, text search engines do not store float numbers in the posting entries of the inverted index representing documents, but they store the term frequencies, which are represented as integers. Therefore, we must guarantee that posting entries will contain integer values proportional to the float values of the deep feature entries.
To employ this idea, we had to define a textual representation of each features array.

Firstly, we associated each element of the array, say e_i, with a unique alphanumeric keyword τ_i. We chose as keyword the letter f and the index i of the array. We returned a text representation of features such that, if an element e in the array in position k (say e_k) is different from zero, then the term τ_k, is repeated e_k times in the text. For example, if the vector that is proportional to the feature vector is [0 0 4 0 2 ... 0 0 3], its textual representation will be:

$$f2\ f2\ f2\ f2\ f4\ f4\ \ldots\ f4095\ f4095\ f4095 \tag{4.5}$$

because the first two elements are equal to 0, so we do not consider them. Instead, the third element whose index position is 2, is equal to 4, and we have to repeat *f2* four times.

As explained, the number of repetitions $rep(fi)$ of the term f_i (corresponding to its term frequency) must be proportional to the value of the i-th entry of the deep feature, i.e.:

$$rep(fi) = \lfloor Qx_i \rfloor$$

Where $\lfloor\rfloor$ denotes the floor function and Q is a multiplication factor > 1 that works as a *quan-*

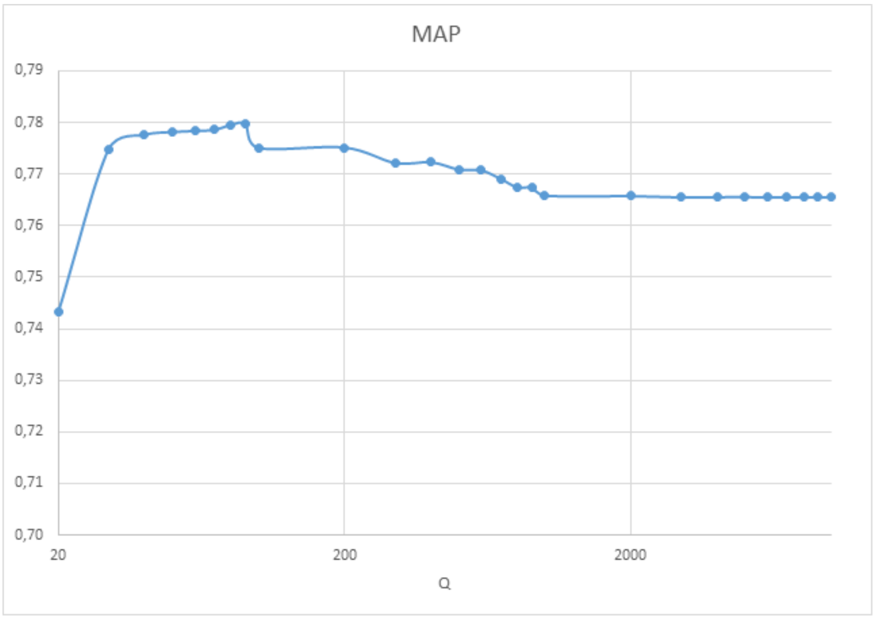


Figure 4.3: *The mean average precision varying values of the factor Q.*

tization factor. For instance, if we fix $Q = 2$, for $x_i < 0.5$, $rep(fi) = 0$, while for $x_i \geq 0.5$, $rep(fi) = 1$. This approach, introduces an approximation since the floor function causes a truncation error. Therefore, the accuracy of this approximation depends on the factor Q, we used to transform the vectors entries. In contrast, the smaller we set Q the smaller the inverted index will be. This is because, the truncation will set to zero more entries of the posting lists. Hence, we have to find a good compromise between the effectiveness of the retrieval system and its space occupation.

In order to decide the optimal value of Q, we have tested this approach on the benchmark dataset INRIA Holidays. INRIA Holidays [21] is a collection of 1,491 holiday images comprising 1,491 high resolution personal photos of different locations and objects, 500 of them being used as queries. The search quality is measured by MAP (mean average precision). In figure 4.3, we report the mean average precision for varying values of the factor Q.
As it is possible to see from the graph a good choice of Q is 50, which exhibits a satisfying value of MAP and small value of Q. It is worth noting that if we use the standard L2 distance on the deep features, we obtain a MAP of about 0.76, which, surprising, is lower than our approach with $Q = 50$ for which we obtain a MAP of about 0.78. The results of this experiment validate our approach.

4.4 Prototype

As mentioned before, in this work we made extensive use of indexes. They are the core of our system. In this section a briefly introduction to Lucene and index process, the system architecture and the Web-based prototype we implemented, are reported.

4.4.1 Architecture

An index is a data structure that improves the speed of data retrieval operations and that is used to quickly locate data without having to access every item every time a query is submitted. In our work, we created our indexes, both visual and textual, using Apache Lucene.
Apache Lucene[1] [22] is a high-performance, full-featured text search engine library written entirely in Java that is suitable for nearly any application requiring full-text search abilities. At the core of Lucene's logical architecture is the idea of a document containing fields of text. This flexibility allows Lucene's API to be independent of the file format. In nutshell, Lucene works as a heart of any search application and provides the vital operations pertaining to indexing and searching. Lucene allows us index any data available in textual format and can be used with almost any data source as long as textual information can be extracted from it. The first step in indexing data is to make it available in simple text format.

The indexing process

Indexing is a process of converting text data into a format that facilitates rapid searching. A simple analogy is an index you would find at the end of a book: that index points you to the location of topics that appear in the book.

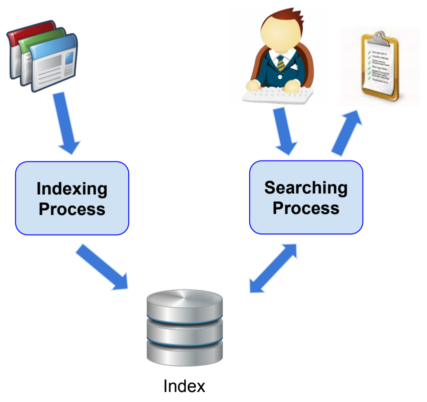


Figure 4.4: *Example of indexing and searching processes.*

As explained above, Lucene stores the input data in a data structure called *inverted index*,

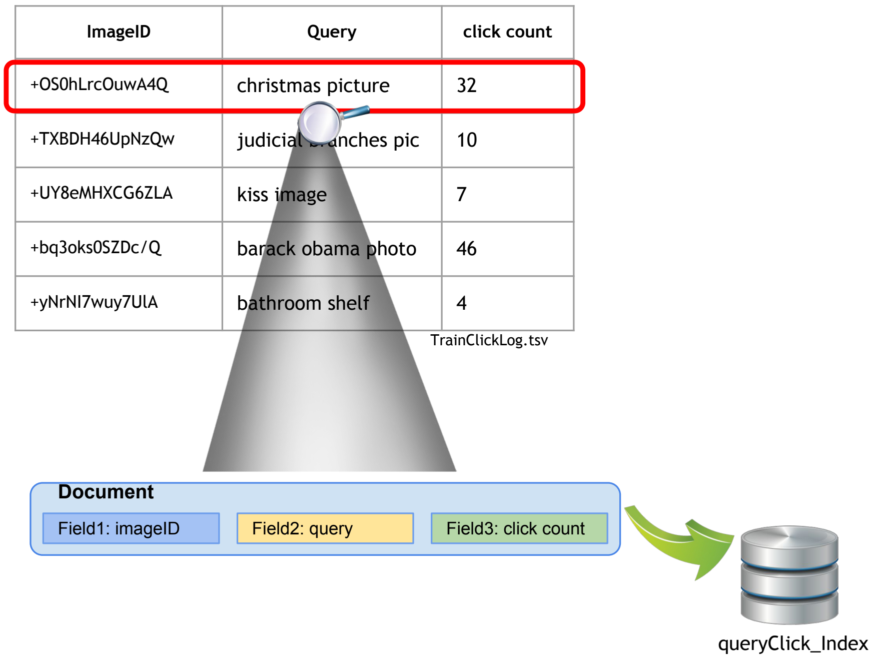
[1]http://lucene.apache.org

which is stored on the file system or memory as a set of index files. It lets users perform fast keyword look-ups and finds the documents that match a given query. Before the text data is added to the index, it is processed by an analyzer (using an analysis process).
Analysis is converting the text data into a fundamental unit of searching or *term.* During analysis, the text data goes through multiple operations: extracting the words, removing common words, ignoring punctuation, reducing words to root form, changing words to lowercase, etc. Analysis happens just before indexing and query parsing. Analysis converts text data into tokens, and these tokens are added as terms in the Lucene index. Lucene comes with various built-in analyzers, such as *SimpleAnalyzer, StandardAnalyzer, StopAnalyzer, SnowballAnalyzer*, and more. These differ in the way they tokenize the text and apply filters. As analysis removes words before indexing, it decreases index size, but it can have a negative effect on precision query processing. We utilized *EnglishAnalyzer* for our textual queries and *WhiteSpaceAnalyzer* for images identifier. The former also includes the EnglishPossesiveFilter (for stripping's from words) and the PorterStemFilter (for chopping off common word suffixes, as removing ming from stemming, etc). The latter splits text into tokens on whitespace characters and makes no other effort to normalize the tokens. It does not lowercase each token.

Indexes construction

To create our indexes, we used the Lucene *Document* class, to which we added the fields we want to index. A Lucene Document is basically a container for a set of indexed fields. *Field* represents a piece of data queried or retrieved in a search. The Field class encapsulates a field name and its value. Lucene provides options to specify if a field needs to be indexed or analyzed and if its value needs to be stored. These options can be passed while creating a field instance. In nutshell, indexing a text file involves wrapping the text data in fields, creating a document, populating it with fields, and adding the document to the index using *IndexWriter.*

We have indexed the whole Clickture-Lite Training set. Since we had textual and visual information, we had to index both of them in a way that was possible to search in the query field or visual features field. We could overcome the typos problem in queries though Lucene. It makes a pre-processing before the indexing process starts. Our first idea was to create a single index in which each Document contained four field: *<imageID, query, click count, text-rep visual features>*, (where "text-rep visual features" is the text representation of the visual features as described in Section 4.3), so that we could get all necessary information with a single access. However, this was not a good idea in terms of space, because an image can be associated with multiple queries, therefore we should repeat the same features for each of these, and the index became huge. Thus, we have created two indexes (Fig. 4.6), one for textual information (*queryClick_index*) and another one for visual information (*features_index*). The



ImageID	Query	click count
+OS0hLrcOuwA4Q	christmas picture	32
+TXBDH46UpNzQw	judicial branches pic	10
+UY8eMHXCG6ZLA	kiss image	7
+bq3oks0SZDc/Q	barack obama photo	46
+yNrNI7wuy7UlA	bathroom shelf	4

Figure 4.5: *Construction of queryClick_index.*

former was structured as follows: for each row of the TrainClickLog.tsv file <*image, query, click count*>, we created three Lucene fields: the first field contains the unique identifier of the Clickture images, the second field maintains the textual information taken from the query inserted by users and the third field contains the click count. The first two fields can be queried separately or in combination. (The index construction is shown in Fig. 4.5). The latter was created in order to support content based search. It consists of three Lucene fields: <*image, visual features in binary format, text-rep visual features*>. The first field is equal to first one of previous index, the second field contains visual features extracted using Caffe framework converted in binary format, and the last field contains features converted to text. We indexed the visual features in binary format, because in future works they can be useful, and we have not to deserialize them again. Instead, the textual representation of features is stored so that we can perform content-based image retrieval using a full-text search engine. In this way, we leave the possibility to the user to search for visual and textual information independently or to search for both information together.

In the Web-based prototype, in addition to these two indexes, we also built another one. Until now we have considered only the *TrainClickLog.tsv* file, but the Training dataset contains also the *TrainImageSet.tsv* file. It has the following format: imageID<*tab*>base$_{64}$ encoded jpeg thumbnail. Thus, we created a third index (Fig. 4.7) in which we considered each row of the last file as Document with two fields, one for the unique identifier of the Clickture training images and the other one for base$_{64}$ encoded thumbnail. This is used by our Web-service to

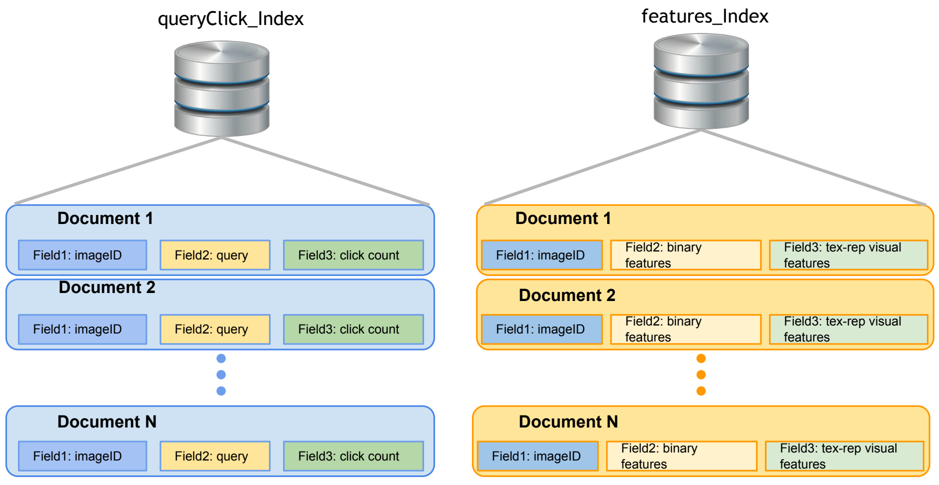


Figure 4.6: *Training indexes for textual and visual information*

decode images faster instead of the sequential scan of the entire file.

4.4.2 Web-based prototype

To allow interactive usage of the system, we implemented a web based image search engine. The URL is as follows:

`http://virserv04.isti.cnr.it:8090/IndexWebApp/pages/searchIndex.jsp`

The user can insert a textual query and the number of results he wants to search for (Fig. 4.9). The IR engine returns a results list of textually similar images. The result is divided in 3 columns: 1) a *text query* similar to the query inserted by the user, 2) *click count* that is the number of times the image associated to the text has been clicked by the users, 3) *image* related to the text.
More precisely, the IR engine processes queries submitted by users looking for words in the index (*queryClick_index* Fig.4.6) and finding the documents that contain these words. We used *IndexSearcher*, a commonly used subclass that allows searching indexes stored in a given directory. The most fundamental unit for searching is Term. It is composed of two elements: the text of the word and the name of the field in which the text occurs. Term objects are created by Lucene internals. Searching for a specific word or phase involves wrapping them in a term, adding the terms to a query object, and passing this query object to IndexSearcher's search method. Lucene comes with various types of concrete query implementations. We wanted to search by phrase, so we used *PhraseQuery* that matches documents containing a particular sequence of terms. It uses positional information of the term that is stored in an index. We also used a *QueryParse* for parsing user-entered query string and for escaping special characters such as /, *, ? with a backslash (\ ). Besides, it is important to use the same analyzer in the query parser as is used in the creation of the index. If they do not match, queries that

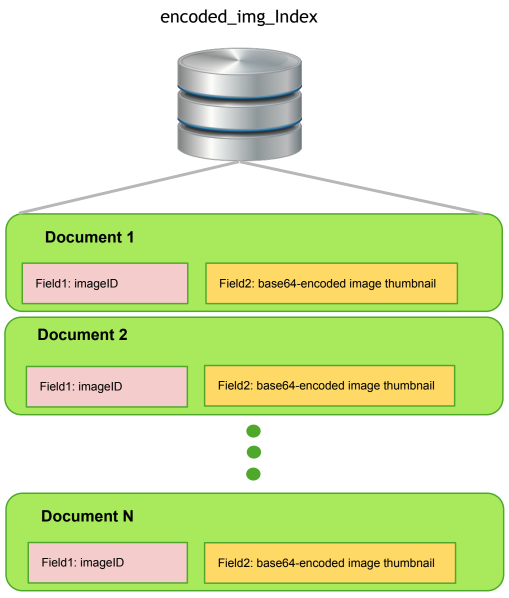

encoded_img_Index
Document 1
Field1: imageID
Field2: base64-encoded image thumbnail
Document 2
Field1: imageID
Field2: base64-encoded image thumbnail
Document N
Field1: imageID
Field2: base64-encoded image thumbnail

Figure 4.7: *Index containing encoded images thumbnails*

should succeed will fail because the tokens will not match. Thus, also in this case we used EnglishAnalyzer for queries.

Lucene returned an array of searched results sorted in decreasing order of score, with higher scores representing better matches. Lucene calculated a score for each of the documents that match the given query. With the Lucene document indentifier, we were able to retrieve the document from the searcher (which just delegates this operation to its index reader internally). Finally, with the document in hand, we could retrieve the values we need. We returned all the three fields values: query, image id and click.

The steps we followed to build our application using Lucene are shown in Fig. 4.8.

In the Web-based prototype it is also possible to perform visual search, by clicking on the images returned by previous search (Fig. 4.10). We considered the clicked image identifier and using the *features_index* we retrieved the correspondent textual features. We searched for similar features with a procedure same as that described previously for queries. Since in our dataset we had base$_{64}$ encoded jpeg image thumbnail, we implemented also a RESTful Web Service to decode and return them. Finally, the decoded images that represent those similar to the clicked image by the user, are shown.

RESTful Web Service

Representational State Transfer (REST) is an architectural style for networked hypermedia applications and it is primarily used to build Web services that are lightweight, maintainable, and scalable. A service based on REST is called a *RESTful service*. REST is not dependent on

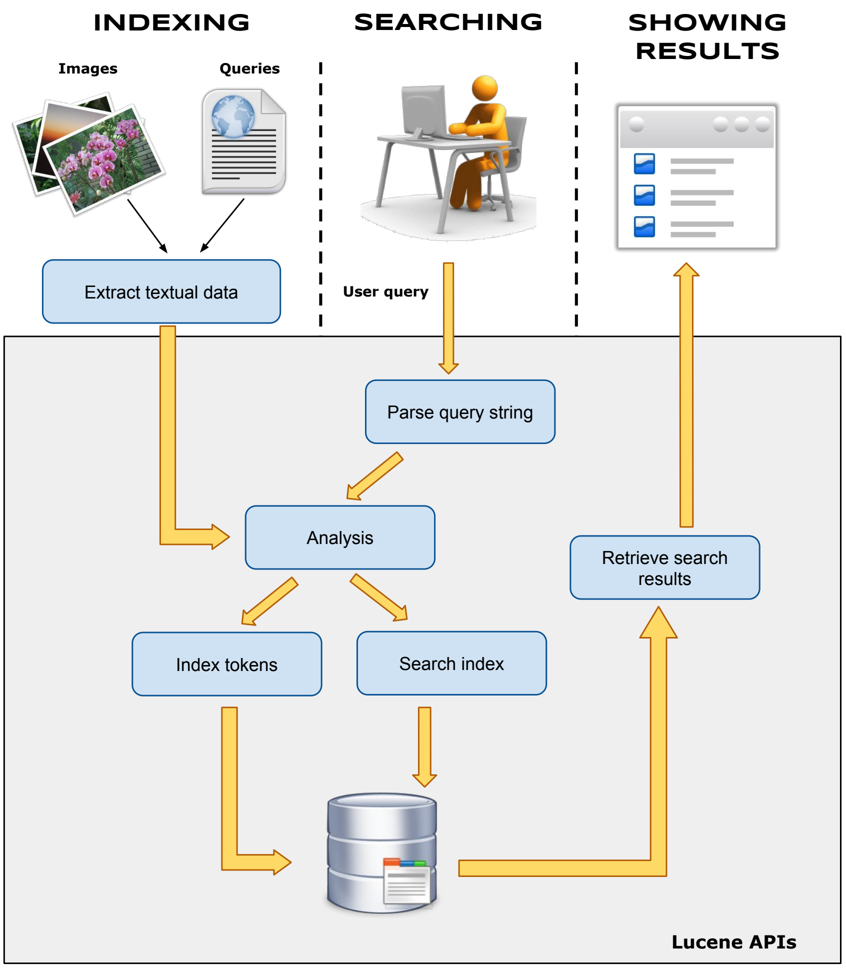
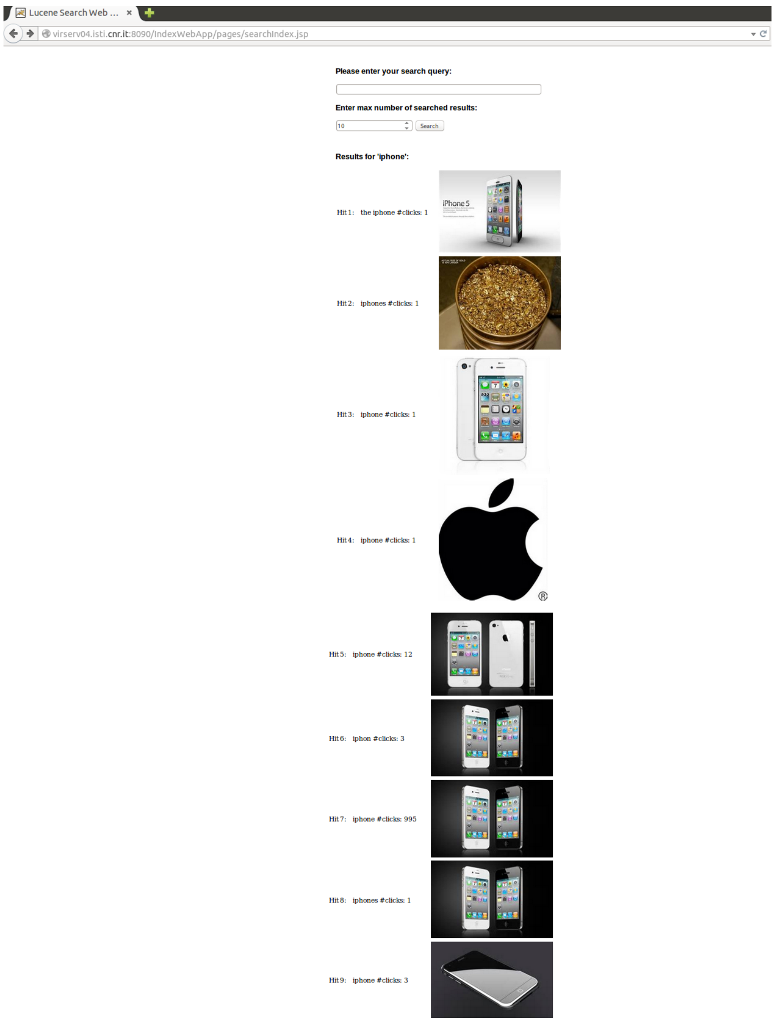


Figure 4.8: *Steps in building our application using Lucene*

any protocol, but almost every RESTful service uses HTTP as its underlying protocol. REST was first introduced by Roy Fielding in 2000.

The focus of a RESTful service is on resources and how to provide access to them. A resource is everything that can be reached, accessed and transferred on the Web from server to client. It can easily be thought as an object accessed by a common interface using HTTP standard methods. While designing a system, the first thing to do is identify the resources and determine how they are related to each other. In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents them. Here each resource is identified by *Uniform Resource Identifiers* URIs, global IDs. REST uses various representations to represent a resource like text, JSON and XML. Nowadays JSON is the most popular format being used in web services.
REST asks developers to use HTTP methods explicitly and in a way that's consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping


Lucene Search Web ...
virserv04.isti.cnr.it:8090/IndexWebApp/pages/searchIndex.jsp
Search
iPhone 5

Please enter your search query:

Enter max number of searched results:

10

Results for 'iphone':

Hit 1: the iphone #clicks: 1

Hit 2: iphones #clicks: 1

Hit 3: iphone #clicks: 1

Hit 4: iphone #clicks: 1

Hit 5: iphone #clicks: 12

Hit 6: iphon #clicks: 3

Hit 7: iphone #clicks: 995

Hit 8: iphones #clicks: 1

Hit 9: iphone #clicks: 3

Figure 4.9: *Example of results that we obtain with our IR engine, when we search for the text query "iphone"*

between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping:

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.

In order to simplify development of RESTful Web service and their clients in Java, a standard and portable JAX-RS API has been designed. We used Jersey[2], an open source framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs. It allows to create resources using specific annotation for methods and classes.

[2]https://jersey.java.net

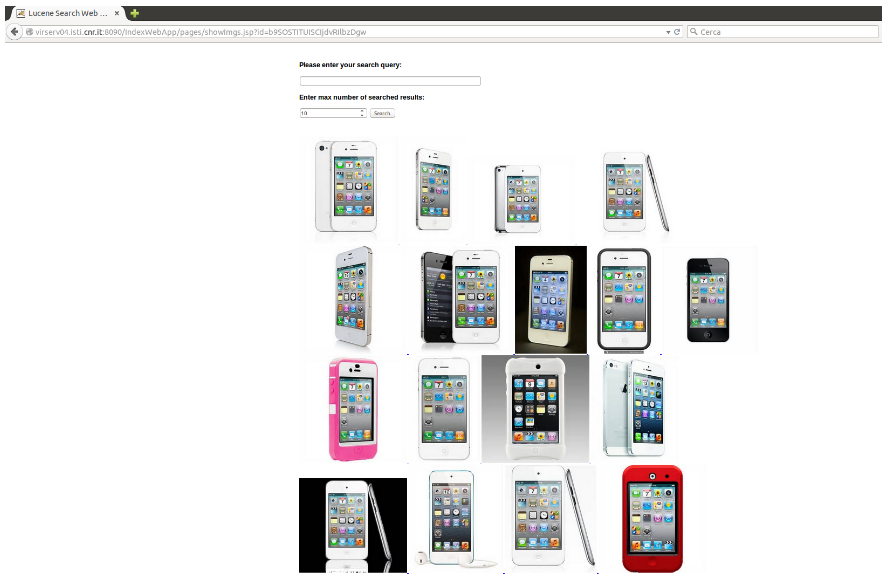

Lucene Search Web ...
virserv04.isti.cnr.it:8090/IndexWebApp/pages/showImgs.jsp?id=b9SOSTITUISCIjdvRIlbzDgw
Cerca
Please enter your search query:
Enter max number of searched results:
10
Search

Figure 4.10: *Example of results that we obtain with our IR engine, when we perform visual search. These images have been obtained clicking on an image returned from the text search with text query "iphone"*

We developed a RESTful Web service that uses Jersey framework and considers our images as resources. It is used within our IR engine. After finding images similar to the one clicked by the user, using their ID we made a request to the RESTful Web Service. In nutshell, we asked for an image through its ID and the web service searched for the image in the index (*encoded_img_index* Fig.4.7), decoded it, because as mentioned before images were base$_{64}$ encoded, and returned it. To retrieve the requested resource our web service used GET method. We can make a request in this way:

http://virserv04.isti.cnr.it:8090/MyIndexWebService/rest/service/getImage/imageID

where *imageID* represents the id of the image to decode.

4.5 Sorting images according to relevance to a text query

As mentioned before, given a query and a list of images related to that query (contained in the Test set), we need to measure the relevance of the images according to the query. In other words, we have to return this list of images re-ranked based on relevance score.
The most intuitive way to measure the relevance score of an image-query pair is to retrieve visually and textually similar images from the training dataset. Our assumption is that a text query is relevant to the image, if other visually similar images are associated with similar text queries. To search for reference candidates, we can not consider only the images returned from textual search or only those returned from visual search. We have to take in consideration both of them.

Given an image-query pair read from *DevSetLabel.tsv*, we considered separately the text query T_Q and the query image I_Q to create two set of candidates, that we can call C_T and C_V, as shown in figure 4.11. These sets consist of triplets of the Training set: image, text query, click count. Finally, to retrieve relevance candidates from the two sets we built, two different strategies, intersection and union, have been proposed.

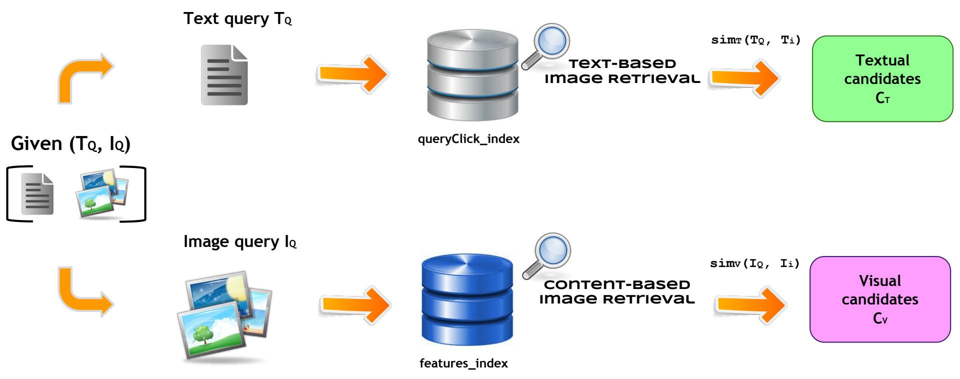


Figure 4.11: *Search procedures to retrieve relevance candidates*

Since the Test set is not available in advance, we considered each image-query pair of the Development set, that has the same structure as the Test set. It contains 1000 text queries, and for each of them there are a different number of associated images.

4.5.1 Text search

Firstly, using the text query T_Q, we performed *text-based image retrieval* from the knowledge base, through Lucene queryClick_index. This procedure found similar queries in the index and its corresponding image and click count. The retrieved textually similar images are denoted as *textual candidates* and were added to C_T. In this search procedure, we used the default text similarity in Lucene to measure the similarity of two queries. Lucene uses a modified version of Vector Space Model (VSM) to calculate the similarity of two documents. In VSM, documents and queries are represented as weighted vectors in a multi-dimensional space, where each distinct index term is a dimension, and weights are Tf-idf values, as described in section 3.3.1. Instead, the scoring function used by Lucene is:

$$sim_T(Q,D) = Score(Q,D) = queryNorm(D) \cdot coord(Q,D) \cdot norm(D) \cdot \sum_{t \,\in\, Q} tf(t,D) \cdot .idf^2(t) \tag{4.6}$$

where:

- Q is the given query;

- D is the document retrieved by Lucene;
- $queryNorm(D) = \frac{1}{\sum_{t \in D} idf^2(t)}$ is the query normalization factor used to make scores between queries comparable, and does not affect document ranking;
- *coord(Q, D)* is a score factor based on how many of the query terms are found in the specified document, and it may affect the ranking. The coordination factor multiplies the score by the number of matching terms in the document, and divides it by the total number of terms in the query. Typically, a document that contains more of the query's terms will receive a higher score than another document with fewer query terms.
- $norm(D) = \frac{1}{\sqrt{|D|}}$ is the document length normalization and is computed during indexing and stored in the index;
- $tf(t, D) = \sqrt{tf_{t \in D}}$ is the term frequency, defined as the number of times term t appears in the currently scored document D. Documents that have more occurrences of a given term receive a higher score.
- $idf(t) = 1 + \ln \frac{N}{1+df(t)}$ is the inverse document frequency, where N is the total number of documents and df(t) is the document frequency, or in other words the number of document in which the term t appears. This means rare terms give higher contribution to the total score. idf(t) appears for t in both the query Q and the document D, hence it is squared in the equation.

Thus, these resulting scores can be used to select the K highest-scoring documents for the given query T_Q. The top K candidates were added to C_T.

4.5.2 Visual search

Secondly, we performed *content-based image retrieval* using the deep features extracted by Caffe framework. We needed to extract deep features also from query image I_Q. The procedure to extract features "on the fly" using Caffe is quite complex, because as said before, we have to specify various parameters (model architecture, batch size, mini batch size, name of features blob), create a file containing a list of files to process and take the features from the database created by Caffe itself using protobuf compiler. To overcome these problems, we decided to extract the deep features from all Development set images and we inserted them in an index similar to *features_index* (Fig. 4.6). They differ in the last field: it contains the label (Excellent, good, bad) in place of the click count. In this way, instead of extracting features each time an image-query pair is provided, we accessed to the index and we took related features, through imageID. Once we had the visual features of I_Q, we could retrieve visually similar images under the inverted index *features_index.* In this case, we did not use the default Lucene similarity. We noted that it is better to use the cosine similarity, because we can obtain a greater MAP (Mean Average Precision) value than what we would get using euclidean distance on deep vector (for details, see section 4.2). To be more precise, we did not apply the standard cosine similarity between image query and document, that is:

$$\cos(\vec{q}, \vec{d}) = sim(\vec{q}, \vec{d}) = \frac{\overrightarrow{w_Q} \cdot \overrightarrow{w_D}}{\parallel \overrightarrow{w_Q} \parallel \cdot \parallel \overrightarrow{w_D} \parallel} = \frac{\sum_{i=1}^{|V|} w_{Q,i} \cdot w_{D,i}}{\sum_{i=1}^{|V|} w_{Q,i}^2 \cdot \sum_{i=1}^{|V|} w_{D,i}^2} \tag{4.7}$$

where $\mathrm{w_{Q,i}}$ is the tf-idf weight of term i in the query Q; $\mathrm{w_{D,i}}$ is the tf-idf weight of term i in the document D; $| \vec{w_Q} |$ and $| \vec{w_D} |$ are the lengths of $\vec{q}$ and $\vec{d}$. The numerator represents the dot product of the two vectors, while the denominator is the product of their Euclidean lengths.

We used a modified version in which idf(i) is set to 1 and we did not consider the denominator (length-normalized vectors). Therefore, our visual similarity is:

$$sim_V(\vec{Q}, \vec{D}) = \overrightarrow{w_Q} \cdot \overrightarrow{w_D} = tf_{i,Q} \cdot tf_{i,Q} \tag{4.8}$$

where $tf_{i,Q}$ is the term frequency of term i in query Q, and $tf_{i,D}$ is the term frequency of term i in document D. As for the text query, at this point we could select the K highest-scoring documents for the given image I_Q. The top K visual candidates were added to C_V.

4.5.3 Intersection strategy

Given a query-image pair ($\mathrm{T_Q}$, $\mathrm{I_Q}$), we have applied text-based and content-based image retrieval from the Training set, we have calculated the textual and visual similarity, and we have built two set of candidates, C_T that contains triplets with associated text similar to the text query $\mathrm{T_Q}$, and C_V that contains triplets in which images have visual content similar to the image query $\mathrm{I_Q}$.

Now, we have to combine the textual and visual similarity to calculate the final relevant score (Fig. 4.12). If an image is not in the set C_V, it means that it has not a visual content similar to $\mathrm{I_Q}$ or the visual similarity is too low. Thus, we can not consider this image, because in any case it would produce a low score. This also applies to a text query. If it is not in the set C_T, we can not consider it, because the textual similarity is too low, and it does not affect the score. Therefore, we can intersect the two set, and obtain the relevance candidates:

$$Relevance\ candidates = C_R = C_T \cap C_V \tag{4.9}$$

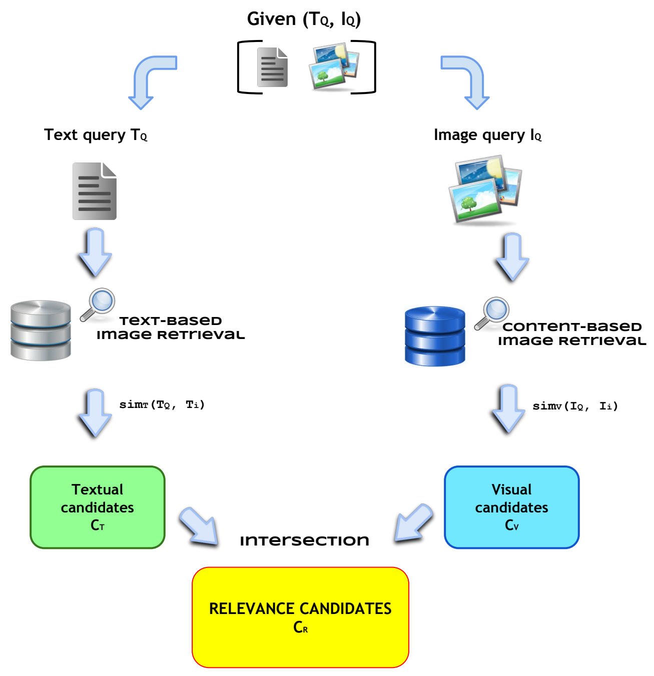


Figure 4.12: *Intersection strategy*

At this point, given (T$_Q$, I$_Q$), we have to provide the relevance of the image according to the query. Hence, for each element of new set C_R, we have to calculate a relevance score. Then, we sum up all these values and we assign the final score to the given image-query pair, as follows:

$$Relevance\ score = \sum_{i \in C_R} sim_V(I_Q, I_i) \cdot sim_T(T_Q, T_i) \cdot click\_count \tag{4.10}$$

where $sim_V(I_Q, I_i)$ is the visual similarity between the query image I$_Q$ and the i-th image of the C$_R$ set; $sim_T(T_Q, T_i)$ is the textual similarity between the text query T$_Q$ and the text associated to the i-th image of the C$_R$ set. Similarities are weighted by reliability of candidates, that is the *click_count*, or the number of times the image was clicked in the search results of the query. The bigger the click is, the more people clicked image i.

This procedure has been repeated for all the images of the list associated to a given query. Once each of these images had a relevance score, we could re-rank them based on the prediction value we had associated. At this point, the evaluation metric *Discounted Cumulated Gain* (DCG) was applied to measure ranking results against the manually labelled Development set (or Test set, when it is available). DCG, as MAP, is a measure of effectiveness used to evaluate ranked lists of recommendations. To be more precise, given the scores returned by the system for the whole list, scores were sorted and the top 25 images were used for measurement. If the images number associated to a given query was less than 25, the current number of images was used.

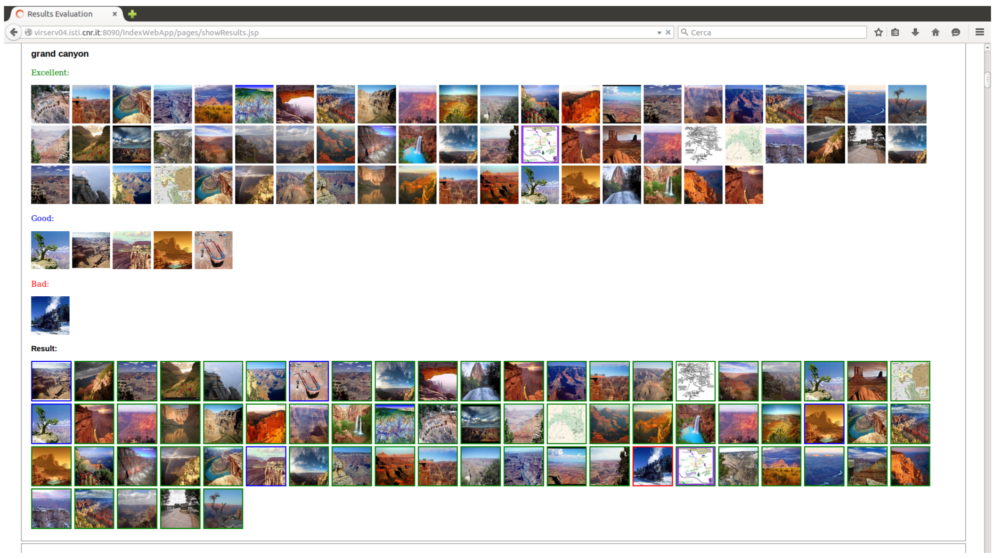


Figure 4.13: *Example of our HTML page with ranked images related to the text query "grand canyon"*

The calculation of DCG is as follows:

$$DCG_N = \alpha_N \cdot \sum_{k=0}^{N} \frac{2^{rel_k} - 1}{\log_2(k+1)} \tag{4.11}$$

where $N = min(25,\ number\ of\ images)$, $\alpha_N = 0.01757$ is a normalizer to make the score for 25 Excellent result equal to 1; rel_k is graded relevance score of the result in position k in the labelled dataset and $rel_k = Excellent = 3$, $Good = 2$, $Bad = 0$. It is the manually judged relevance for each image with respect to the query. DCG@25 appreciates predicting the *Excellent* labelled images as top-ranked, and gives penalty when the *Bad* labelled images as top-ranked. With $\log_2(k+1)$ as denominator, the higher positions have more influence on the final DCG@25 than the lower positions. In the Development set, many queries have less than 25 Excellent images responding to the query, and in many cases, the number of images is less than 25.

The average of the DCG_S on all the queries is the final evaluation result:

$$Avg\_DCG_S = \frac{\sum_{i=1}^{N} DCG_i}{N} \tag{4.12}$$

where N is the number of queries, that is equal to 1000 in the Development set.

To verify and show our results, we generate an HTML page in which it is possible to see images ranked according to the scores (Fig. 4.13). We know that in the Development set there are 1,000 queries with a different number of associated images and their level labels (excellent, good, bad). In the HTML page, for each query of this set, three images categories are shown. We divided the images related to each query, based on their level labels. Each category has a color: *Excellent* in green, *Good* in blue and *Bad* in red. After these three groups, images sorted

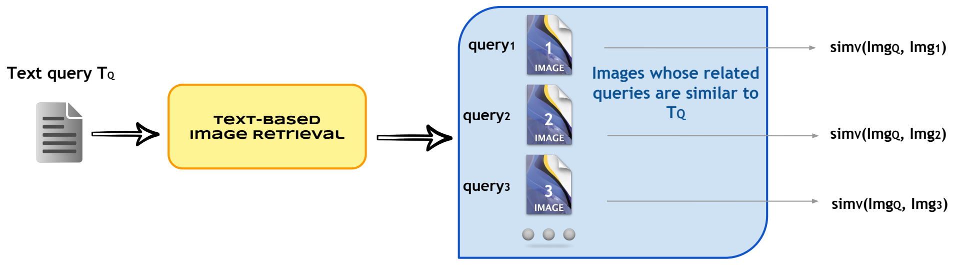


Figure 4.14: *First part of union strategy*

according to the score are shown. The images are surrounded by a colored square. The color depends on the category to which the images belong.
This method allows us to see if the strategy with which we assign scores is correct. If fact, if all the images with the red square were at beginning of the ordering, it would mean that there is a problem. The URL of the HTML page is as follows:

`http://virserv04.isti.cnr.it:8090/IndexWebApp/pages/showResults.jsp`

4.5.4 Union strategy

We have said that our search-based system consists of two parts: text-based and content-based image retrieval. As for the intersection strategy, we used the results of these search methods for each image-query pair ($\mathrm{I_Q}$, $\mathrm{T_Q}$), then we calculated the textual and visual similarity, and we built two set of candidates, C_T and C_V. Unlike the previous case, now we have to do more computations. To be more precise, we can first retrieve textually similar images and then calculate visual similarity from the top-ranked images. Similarly, we can retrieve visually similar images and calculate textual similarity from the top-ranked images. We can divide this procedure in two parts: *text search* and *visual search*.

We started from the *text search* (Fig. 4.14). Using the text query T_Q, we have performed text-based image retrieval, that found similar queries in the index and its corresponding image and click count. Then, the top textually similar images, *textual candidates*, are added to C_T. For each image in $\mathrm{C_T}$, that we can call $I_{C_{Ti}}$ we calculated visual similarity with image query $\mathrm{I_Q}$:

$$sim_V(I_Q, I_{C_{Ti}}) \tag{4.13}$$

For the textual similarity, we have used the default text similarity in Lucene, as said before. Instead, we had to write a function to calculate the visual similarity (Eq. 4.13). As mentioned before, for the images, we used a modified version of the standard cosine similarity, in which we considered only the product of term frequencies (eq. 4.8). We divided in terms the features textual representation of the image query $\mathrm{I_Q}$, and the features textual representation of each image $I_{C_{Ti}}$. We also calculated the term frequency (tf) of each term. Then, we intersected this two sets of terms and we obtained the set T, that maintained common terms to $\mathrm{I_Q}$ and $\mathrm{I_{C_{Ti}}}$ textual representations. Thus, we computed the visual similarity as follows:

$$sim_V(I_Q, I_{C_{Vi}}) = sim_V(Q, D) = \sum_{i \in T} tf_{i,Q} \cdot tf_{i,Q} \tag{4.14}$$

where Q is our image query $\mathrm{I_Q}$, the document D represents an image of the textual candidates $I_{C_{Ti}}$, T is the set that contains the common terms to the query $\mathrm{I_Q}$ and the document $I_{C_{Ti}}$, $tf_{i,Q}$ is the term frequency of the term i in query Q, and $tf_{i,Q}$ is the term frequency of the term i in document D.

Therefore, we had all the necessary elements to calculate the relevance score: textual similarity, visual similarity and click count (Eq. 4.10).

The second part has been implemented using the query image $\mathrm{I_Q}$. The procedure is shown in Figure 4.15. We performed visual search to retrieve similar images from the index, we calculated visual similarity and we added the top-ranked images, *visual candidates*, to C_V. As in the previous case, we had only one similarity, visual similarity, but we did not have textual similarity.
For each image in C_V, we had to retrieve the corresponding text query and click count, from the *queryClick_index*. An image can be associated with multiple queries, so we had to consider all of them when we calculated the textual similarity. We can call this set of retrieved queries, R_Q. Thus, the final textual similarity between text query $\mathrm{T_Q}$ and the queries in R_Q, is equal to the sum of each textual similarity between $\mathrm{T_Q}$ and each query in R_Q:

$$sim_T = \sum_{k \in R_Q} sim_{T_k}(T_Q, T_{R_{Q_k}}) \tag{4.15}$$

where T_Q is our text query and $T_{R_{Qi}}$ is a text query in R_Q.

For the visual similarity, we have used a modified version of the standard cosine similarity, as said for the intersection strategy (Eq. 4.8). Instead, for the textual similarity we have adopted another strategy. Since the textual queries are very short, we can neglect the inverse document frequency of terms. Thus, we decided to use the same similarity used for images. The idea was to divide in terms the given textual query T_Q and the textual queries associated to the retrieved images, to calculate each term frequency, to intersect the two set of terms, and to multiply the term frequencies of the intersection terms:

$$sim_{T_k}(T_Q, T_{R_{Q_k}}) = sim_T(Q, D) = \sum_{i \in T} tf_{i,Q} \cdot tf_{i,Q} \tag{4.16}$$

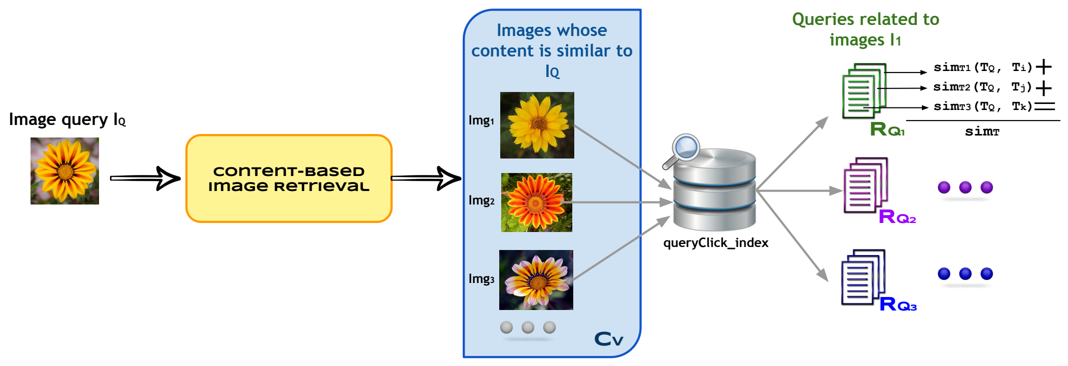


Figure 4.15: *Second part of union strategy*

where T_Q is our textual query, $T_{R_{Qk}}$ is a query of the set $\mathrm{R_Q}$, T is the set containing the common terms to the query $\mathrm{T_Q}$ and of each query in $\mathrm{R_Q}$.

We had all the elements to calculate the relevance score (Eq. 4.10) also for this second part. Then, we made the union of the two parts and we expected to measure ranking results through the DCG (eq. 4.11). This idea works only conceptually, but in practice the scores were not correct. The problem was due to the way we estimated the textual similarity (eq. 4.16). Lucene makes some internal optimization during the indexing process and when computes the similarity. Therefore, the values we calculated are different, and the scores are not reliable.

Exhaustive search

Since the union strategy described before was not a reliable solution due to the way we calculated textual similarity, we looked for other solutions.

Given the text query $\mathrm{T_Q}$, through text search, we find all possible similar queries to $\mathrm{T_Q}$, that are K. We call S_K the set of images associated to this queries. So, we have found the maximum number of results that we can obtain when performing textual search is about 45,000. More precisely, we get K = 45,070 similar queries when we search by textual query within *queryClick_index*.

As said in the first part of union strategy, after text search, we had to compute the visual similarity between query image I_Q and each image in S_K. So, we had textual similarity, visual similarity and click count and it was easy to calculate the relevance score. In this new procedure, we slightly modified the second part of union strategy. Also in this case, we performed visual search, we computed visual similarity and we considered the top-ranked similar images that we inserted in the visual candidates set C_V. But, instead of estimate the textual similarity between queries associated to these images and the text query $\mathrm{T_Q}$, we used another strategy. If the images contained in C_V were not present in the set S_K, it meant that the textual similarity was equal to 0. Otherwise, we could use the textual similarity between the query related to the image we were considering in S_K, and the given text query, that we had already calculated. Although this strategy works in theory, in practice it is difficult to apply. Many data structures are required, for example HashMap to maintain the results of text-based retrieval, other similar structures to maintain the set C_V considering all the different K values. We should have kept this information in memory and for each visual candidate in the set C_V, we should have checked if it was present in S_K. The cost in terms of space could be expensive.

Therefore, we tried another solution, *exhaustive search*: also for the visual search we found all possible similar images to $\mathrm{I_Q}$. The number of hits was equal to the entire Training set, that was $\mathrm{N} = 1$ million. We can call S_N the resulting set. Thus, instead of modifying our code, for each image-query pair ($\mathrm{I_Q}$, $\mathrm{T_Q}$), we retrieved all possible results of visual search and we inserted them in S_N and all possible results of text search, that were inserted in S_K. Then, we intersected S_N and S_K to find relevance candidates. But unfortunately, we were not able to complete the computation, because of the cost in terms of time. In fact, to retrieve the relevance candidates of only one query, it takes about 22 hours. Considered that the total number of Development set queries is 1000, it is not a viable solution.

To implement the union strategy, we had to extend the visual candidates set, $\mathrm{C_V}$, in order to evaluate the textual similarity between the queries associated to the images of the set and the given text query $\mathrm{T_Q}$. Similarly, we had to extend the textual candidates set, $\mathrm{C_T}$, in order to evaluate the visual similarity between the images within the set and the given image query $\mathrm{I_Q}$. However, from the intersection results, we noted that using low values for the candidates sets size, we get already high DCG values. Therefore, it is not convenient to extend the sets, because the cost to calculate the similarities is high, since we no longer use the indexes. More details are given in Chapter 5.

Chapter 5

Experiments and Results

In this chapter, we report the results of the experimental evaluation of our proposed approach. The objectives are to understand if the proposed approach is efficient and if the goal of Microsoft Challenge has been reached.
We will go into details and discuss the following aspects:

- what is the best measure of similarity to compare similar images and queries;
- evaluate the dependency on the size of the candidates set;
- which of the two proposed strategies, intersection or union, is better;
- comparison with winners of previous editions.

5.1 Dataset

As explained in Section 1.2.1, we used the Clickture-Lite dataset that was sampled from one-year click logs of Microsoft Bing Image search engine, which is composed of three parts, (1) *Training set*, (2) *Development set* and (3) *Test set*. However, since the Test set is made available when the competition starts, we evaluated our system using the Development set, that has the same structure as the Test set. It is composed of two files: (1) DevSetLabel.tsv and (2) DevSetImage.tsv.
The first file contains about 80K triads with 1000 distinct text queries and 80K images. Each triad is as follows:

$$\text{query}\langle tab\rangle \text{ image ID } \langle tab\rangle \text{ judgement}$$

where judgement is a label with three different levels: *Excellent*, *Good*, *Bad*. The second file consists of about 80K entries, one for each image of the Development set, in which there are base_{64} encoded JPEG images thumbnails.

We used the entire Training set provided by the MSR-Bing Image Retrieval Challenge, which contains 1 million images with 23 million click logs.

5.1.1 Evaluation metric

In the evaluation stage, we estimated a relevance score for each query-image pair contained in DevSetLabel.tsv file of the *Development set.* For each query, images were ordered based on the prediction scores. Then, *Discounted Cumulated Gain* (DCG) (Eq. 4.11) was applied to measure ranking results against the manually labelled Development set. Given the scores returned by the system for the whole images list, scores were sorted and the top 25 images were used for measurement. DCG@25 increases when the *Excellent* labelled images appear as top-ranked, and gives penalty when the *Bad* labelled images are top-ranked. In the Development set, many queries had less that 25 Excellent images responding to the query. So the optimal DCG@25 over 1,000 queries in Development set was not 1, but it was about 0.68. However, we noted that many queries had less than 25 associated images. For this reason, we decided to calculate also the DCG considering the actual number of images associated to a given query. For example, if the number of images was N=10, we multiplied by α_{10}, instead of multiplying by 0.01757, that is α_{25}.
The average of the DCG_S on all the queries, has been used as final evaluation result:

$$Avg\_DCG_S = \frac{\sum_{i=1}^{N} DCG_i}{N} \tag{5.1}$$

where N is the number of queries, that is 1,000 in the Development set.

5.2 System setting

For the experiments that we performed, we used a server running Ubuntu Linux x64 v 12.04. The toolkit we used to extract visual features is Caffe, an efficient implementation of deep Convolutional Neural Network. Moreover, the server ran a LevelDB database instance in order to save all the features we extracted from images.
On the current implementation, the visual features extraction from Development set images was made on the server before the evaluation phase started and we put them within an index. Unfortunately, Caffe does not provide an interface to extract deep features "on the fly", therefore we extracted them off-line.

5.2.1 Caffe parameters setting

As mentioned in Section 4.3, we used a Caffe pre-trained model, then we subtracted the mean image of the ILSVRC dataset from all images of our dataset and we used the *ImageDataLayer*, which loaded and resized images for us. In particular, firstly it ignored the image's original

aspect ratio and warps it to 256×256, rather then resizing and cropping to preserve the proportion. Secondly, the features are taken on the centre 224×224 crop of the 256×256 resized images. This layer required two parameters: source and batch size. *Source* was set to a text file path containing the list of files to process, that are all Clickture training set images. In particular, each line contained an image filename and a label, that was 0 in our case. The entire file maintained 1 million entries. The *batch size* was the number of images processed by the net in each run. We decided to set this value to 1. Before the extraction process started, we had to set two other values: number of data *mini-batches* and the name of *features blob* that we extracted. The former was the number of iterations or in other words, the number of images read by the data layer. We set this value to 1 million, that was the number of training images. The latter was set to fc6, that is the first fully connected layer in the whole Alex's convNet (Fig. 3.9).
Moreover, we had to select a database to store the extracted features. We chose the LevelDb database. After construction, the network could be run on either CPU or GPU. $\mathrm{GPU_s}$ have become the most suitable platforms for deep learning. They produce faster results, use lower power and reduce overall cost with respect to $\mathrm{CPU_s}$. But we did not have $\mathrm{GPU_s}$ available, so we ran the network on $\mathrm{CPU_s}$, and we specified it before the extraction started.
The overall parameters are summarized in Table 5.1. For the features extraction of Development images, we used the same parameters as in the previous case, except for the number of images and mini batches that are 79665.

Number of images	1M
Model	bvlc_reference_caffenet.caffemodel
Layer	ImageDataLayer
Batch size	1
Mini batches	1M
Features blob	fc6
Database	LevelDB
GPU/CPU	CPU

Table 5.1: *Caffe parameters used for the extraction of Training images visual features*

5.3 Measure of similarity: Euclidean distance vs Cosine similarity

In our work, to measure the similarity of two text queries, we used the default text similarity in Lucene, that is a modified version of VSM (Eq. 4.6). We compared two methods to compute the similarity between two images: cosine similarity applied to the textual representation of images features and Euclidean distance applied to the original vectors. In the first method, we converted the visual features vectors into text strings, that could be indexed by means of the Lucene search engine library, as described in Section 4.3. Using this approach, given an image query we could easily find similar images through the cosine similarity between features textual representation. Also in the second method, we used the visual features extracted with Caffe. For each image, we considered a descriptor that contained the visual features and an id. Then, the descriptors were loaded in memory. Thus, given a query image, we computed the Euclidean distance between its visual features and the features contained in all descriptors. We applied *K-nearest Neighbours* (kNN) search and we retrieved the K most similar images to the query image.

We compared these two procedures using $K = 1,000$, and we evaluated the results of the intersection strategy through the DCG metric, using α_{25}. Since for the second method we used the Euclidean distance, we had to convert it into a similarity, otherwise we could not compare the two methods. We made the conversion as follows:

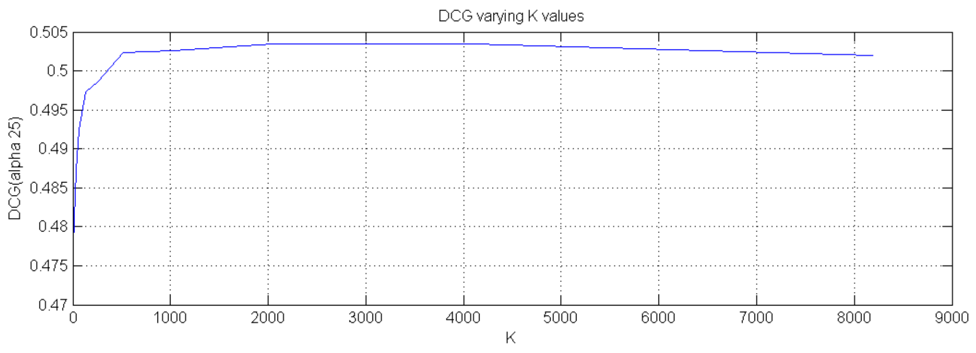
$$similarity = 1 - \frac{Normalized\ Euclidean\ distance}{Max\ distance} \tag{5.2}$$

where normalized Euclidean distance means that the computed distances have been normalized through L_2 norm. This means that it was not necessary to compute the maximum distance between them, because they were all less than $\sqrt{2}$. So, we set $\sqrt{2}$ as maximum distance. We noted that the results were very similar, as shown in Table 5.2.

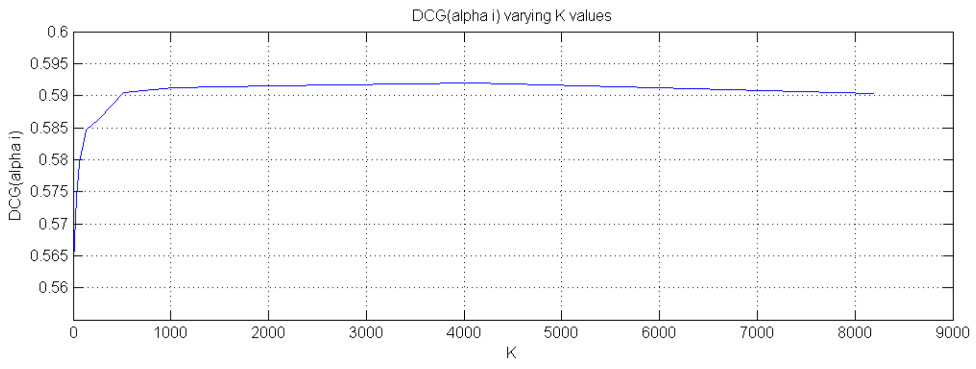
Avg DCG Sequential Search + Euclidean distance	0.5021
Avg DCG Lucene + cosine similarity	0.5017

Table 5.2: *Results obtained using different measures of similarity.*

To decide which of the two measures was better, we also used the bechmark dataset INRIA Holidays, and we evaluated the *Mean Average Precision*, as explained in Section 4.3.1. Using the standard L2 distance on the deep features, we obtained a MAP lower than the MAP obtained using the cosine similarity. Moreover, using Lucene indexes the results have been obtained faster than the sequential search method. For these reasons, the cosine similarity applied to the textual representation of images features was better than the L2 distance. In this way, we could set up a robust retrieval system that combines full-text search with content-based image



(a) AvgDCG results using α_{25}



(b) AvgDCG results using α_i

Figure 5.1: *AvgDCG obtained varying K values*

retrieval capabilities.

5.4 Candidates sets size

When performing text-based image retrieval, the retrieved textually similar images were denoted as *textual candidates*, while when performing content-based image retrieval, the retrieved visually similar images were denoted as *visual candidates*. Thus, through the cosine similarity function, we could select the top K visual candidates, that were inserted in the set C_V. In the same way, using the default text similarity in Lucene, we could select the top K textual candidates, that were inserted in the set C_T.
We tested our system with different K values, in order to select a good size for the sets C_V and C_T. We varied the value of K from 2 to 10,000. We tested the system using the intersection strategy, and for each run we calculated the DCG values (Eq. 4.11). At the end, we compared these values to understand which K was better to use.

In the Table 5.3 and in the Figure 5.1 the results we obtained are shown. $DCG_{\alpha_{25}}$ indicates that we multiplied by α_{25}, and DCG_{α_i} indicates that we multiplied by the α related to the

K	$\mathbf{DCG}_{\alpha_{25}}$	$\mathbf{DCG}_{\alpha_i}$
2	0.4705	0.557
4	0.474	0.5606
8	0.4776	0.5641
16	0.481	0.5674
32	0.4872	0.5741
64	0.4925	0.5795
128	0.4973	0.5847
256	0.4987	0.586
512	0.5023	0.5905
1024	0.5026	0.5912
2048	0.5035	0.5916
4096	0.5035	0.592
8192	0.502	0.5903

Table 5.3: *AvgDCG$_s$ obtained with different K values*

number of images associated to the given query.
As we can see, after a few values, starting at 512, we already have a high value of DCG. The best results have been obtained with K in the range [1024-4096]. The same value of K has been used for both the sets C_T and C_V. We also tested with K=10,000, but there was no improvement. We did not test values of K, because we noted that after 4096, the values tend to stabilize. We think it may be due to the fact that when K increases, there is a kind of noise in results. In other words, there may be no images very similar in the results of CBIR. This also applies to the textual-based image retrieval.

5.4.1 Union strategy sets size

As described in Section 4.5.4, for each query of the Development set, that were 1,000, the number of results that could be obtained by performing textual search has been evaluated. We refer to this number as R_{TS}. We found that the maximum number of results was 45,070. That is, we could get at most 45,070 similar images when we searched by textual queries. This value seems to suggest that we had to use K = $\mathrm{R_{TS}}$ as size of textual candidates set, C_T. Moreover, we noted that the average number of results obtained through textual search was $\mu = 9,933.7$. In the previous stage, we had stored the textual search results obtained with K=10,000. To decide if we could use this values as size of textual candidate set without repeating all searches, or if we had to use K=45,070, the distribution of the textual search results around the mean μ has been calculated (Fig. 5.2). The values on the R_{TS} axis represent the results obtained by

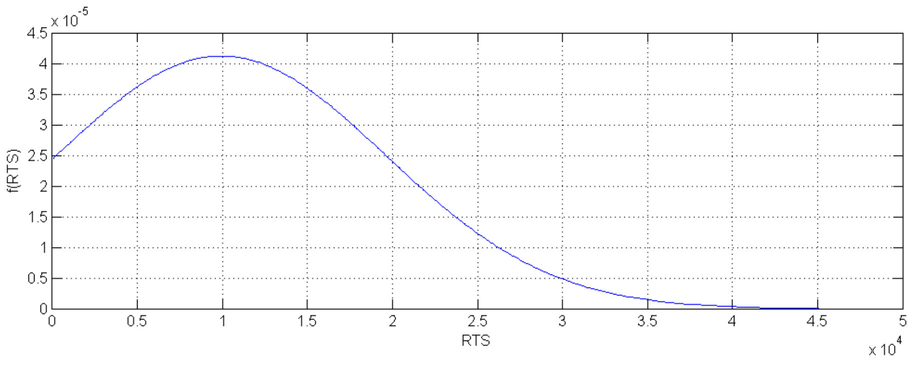


Figure 5.2: *Distribution of the results number obtained when performing textual search around the mean*

means of textual search, whereas *f(R_{TS})* represents the related probability. Since about half of the queries had a number of text search results greater than μ, we would had to use K=45,070 as C_T size.

Since the probability to obtain a number of results less than 10,000 is quite high, we concluded that it was not necessary to repeat the textual search using K = 45,070, but we could use the results previously stored.

The union strategy can be thought as a theoretical approach to our search problem, because of its intrinsic cost in terms of space and time. In fact, to retrieve the relevance candidates of only one query, it takes about 22 hours. Considered that the total number of Development set queries is 1000, it is not a viable solution.
To implement the union, we had to extend the visual and textual candidates sets size, as described in previous chapter. However, as we can see from Figure 5.2, using low values of K, we get already high DCG values. We concluded that it was not convenient to extend the sets. Therefore, the intersection strategy represent a good approximation of the union approach as we have shown. It was easier to implement, we had no problems calculating the similarity functions, we had no problems of space for data structures, we were able to complete his execution and to calculate the DCG, and the results are quite satisfactory.

5.5 Comparison with previous Challenge edition winners

In this section, we present a comparison between our system and the systems of previous Challenge edition winners, in particular the winners of 2013 and 2014. We will compare: the type of visual features, the strategies adopted and the results obtained.

5.5.1 Visual features

In our work, we used deep convolutional neural network features, extracted by means of Caffe framework. We proposed a method to convert image features into a textual form, to index them into an inverted index.

The winner team of 2013 edition extracted different kinds of visual features, state-of-the-art bag-of-words (BoW), the vector of locally aggregated descriptors (VLAD), grid color moment (GCM) and local binary pattern (LBP) for facial images, but they do not used deep features.

The winner team of 2014 edition, made two different experiments: under *offline condition* and *online condition*. In previous editions, the participants had to build an online system which received query-image pairs from servers of Microsoft, and then respond to each query-image pairs with a float number to indicate the relevance in less than 12 seconds. Then, the rule was changed into building an offline system, which means that participants can see the whole image list when processing one testing images. In the first sets of experiments, the team used the deep learning features and tested with two public pre-trained models: *DeCaf* [8], which has exactly the same structures as Alex's convnet, and *OverFeat* [9]. They compared these two networks obtaining similar performance on the MSR-Bing dataset. They decided to fuse results from Decaf and Overfeat. However, for the experiments they conducted under *offline condition*, they used Caffe to extract visual features.

5.5.2 Strategies adopted

We implemented two strategies to sort images according to relevance to a text query and we used Lucene indexes to speed up searching operations. Thus, to measure the relevance score of an image-query pair we retrieved visually and textually similar images from our inverted index structures by applying content-based (CBIR) and text-based image retrieval (TBIR). Intersection and union strategies of textual and visual candidates have been prosed. For the intersection strategy, we implemented the following procedure. We extracted deep learning visual features from the query image and we used the features stored previously, to retrieve similar images under the inverted index structure. We calculated the similarity of the top-ranked visually similar images. Then, we considered the query text, we retrieved textually similar images and we calculated the similarity of the top-ranked images. At the end, we had two sets of candidates: *textual candidates* and *visual candidates*. Their sizes have been described in the previous section (5.4). We made the intersection of these two sets. Finally, we calculated the DCG on the scores of the result set. For the union strategy, the procedure adopted is a little bit different from the previous case, and can be divided in two parts. In the first part, after retrieving textually similar images, we also calculated the visual similarity between these images and the image query. Then in the second part, after retrieving visually similar images, we retrieved the related queries and click count, from the index, and we calculated the textual similarity. Finally, we made the union of the two parts and we measured ranking results through the DCG. We also tried another solution, *exhaustive search*, in which we retrieved all possible similar queries to the text query and all possible similar images to the query image, in which the number of hits was equal to the entire Training set, that was N = 1 million. We tried to intersect the results obtained through these two searches, to find relevance candidates. Unfortunately, we were not able to complete the computation, because of the cost in terms of time, as described in Section 4.5.4.

The winners of 2013 edition investigated the effectiveness of different auxiliary contextual cues: face, click logs, visual similarity. They adopted three strategies: 1) image retrieval with click logs, 2) people/name detection and 3) random walk model. First, the search-based system consisted of two parts: content-based image retrieval with *textual probe*, that is textual similarity association; text-based image retrieval with *visual probe*, that is visual similarity association. This is similar to our union strategy. Second, since a significant portion of the image-query pairs comprised faces and names, they adopted two name detection methods: name list and social media (other details are reported in section 2.5.3). For the name-to-face correspondence estimation, they trained the visual-based face identity models, *Identity Bank*. Third, they thought that if an image in the list associated to a query was relevant to the query, there should be some other images with strong visual similarities. So, those having strong visual connections with others deserved more relevance scores. Therefore, they assigned the relevance score to an image based on its connections (similarities) with others, inspired by the PageRank. In the end, they combined the results of these three methods.

Unfortunately, we do not know many details about the strategies adopted by the winners of 2014 edition, since they were not published. We only know that they applied a sort of clustering between similar images. They relied on an assumption of [6], in which they used text clustering results to form sets image clusters. The sets image clusters served as experts to score query-image pairs. When an expert scores a query-image pair, it calculates the similarity between the test image and the top 5 images inside the image cluster. They used some images to represent the query, thus the relevance problem between text query and testing image was transformed into the similarity assessment problem between testing image and some training images. Instead of utilizing the testing images which came before the current one, as done by this team, the winners of 2014 used also other testing images. They utilized the retrieved images in the training data to represent the text query, so that they built up the relation between text query and testing image through the retrieved images. They utilized Page Rank method to learn the relevance score.

5.5.3 Results

As described before, the evaluation metric used to measure ranking results is Discounted Cumulated Gain (DCG). In the Table 5.4 the results obtained, the related strategies and the kind of visual features used, are reported.

Team	**Visual features**	**Method**	**DCG$_{\alpha_{25}}$**
Winner of 2013	BoW	Random Walk + People Detection	0.544
Winner of 2014	Deep features (fc6)	PageRank + clustering	0.5608
Our solution	Deep features (fc6)	Inverted index + CBIR & TBIR + Intersection	0.5035

Table 5.4: *Comparison between our results and those of previous edition winners*

As we can see from the Table, previous winners results are better than our, but they combined different approaches and methods, they applied also name and face detection, and they used different datasets.

5.6 Comparison with 2015 Challenge edition winners

We also compared our approach with those proposed by participants to the edition of 2015.

The winner team [23], proposed a cross-media relevance fusion. Four base cross-media relevance functions were investigated and combined. In order to generate relevance scores, they adopted three different types of methods: 1) *image2text*, 2) *text2image*, and 3) *semantic embedding*. In the first method, given an image, they retrieved its K-nearest visual neighbour from the dataset, and they calculated the relevance. For the visual similarity, they extracted CNN features and they used the cosine similarity. They employed two pre-trained Caffe models, using fc7 layer as features. In the second method, for a given query, they retrieved the top k most similar queries, and calculated the relevance. They also implemented another version of this method, called *text2image as Parzen window*. In the third method, semantic embedding, they utilized ConSE [24], a deep learning based semantic embedding method. Given the above four methods and two CNN models, they computed eight scores per image-query pair. Linear fusion was applied for its effectiveness and efficiency.

We do not have details on the approach used by the team who finished in second place. Instead, we described the approach applied by the team that achieved the third place [25]. This team integrated a series of methods with visual features obtained by CNN models. Their CNN model was pre-trained on a collection of clean datasets and fine-tuned on the Bing datasets. A multiscale training strategy was adopted by simply resizing the input images into different scales and then merging the soft-max posteriors. To extract features, they utilized CNNs trained on two different large datasets, namely object-centric ImageNet dataset and scene-centric Places dataset [26]. They used Caffe to extract features from 7-th layer. To measure the similarities between queries, they used the Jaccard index. They obtained 450 completely matched queries and 550 partially matched queries. The click count of images were sorted in decreasing order and the top 5 images in the training dataset were selected as the templates. As for each query in the partly matched list, they obtained top 5 similar queries in training dataset and pick images with highest click count in these 5 queires as the templates. A K-means clustering has been implemented as ranking strategy. They also implemented a modified PageRank method to obtain the ranking of images.

	Visual features	**Method**	**DCG$_{\alpha_{25}}$**
First place	Deep CNN features(fc7)	image2text + text2image + semantic embedding	0.5200
Second place	-	-	0.4868
Third place	Deep CNN features (fc7)	K_means clustering + modified version of PageRank	0.4715
Our solution	Deep CNN features (fc6)	Inverted index + CBIR & TBIR + Intersection	0.5035

Table 5.5: *Comparison between our results and those of 2015 edition winners*

In the Table 5.5, a comparison between our results and methods and those used by other teams are shown. As we can see, our results do not differ much from others. Since a significant portion of image-query pairs in the Clickture-Lite dataset comprises faces in images, we think that our final result can be improved by applying face detection and recognition. For now, we can be quite satisfied with the values obtained. However, we have also to consider the fact that we do not have the Test set. This means that using that set, our results could be slightly different.

To summarize, comparing our approach with those used by other teams, we can say that the use of the indexes is a good solution. Apache Lucene indexes allowed us not only to compare images faster than the sequential scan of the whole dataset, but they also provided the similarity values that we used to calculate the scores. To solve efficiency issues, we also proposed an approach that index visual features using Lucene. The other teams did not use Lucene. We do not know which strategy has been used by the winner team to retrieve the k nearest visual neighbours and the top k most similar queries. Instead, the team who finished in third place applied K-means clustering to retrieve similar images and utilized the Page Rank method to learn the relevance scores.

Chapter 6

Conclusions

In this work, a multimedia information retrieval engine for the MSR-Bing Retrieval Challenge provided by Microsoft, has been implemented and presented. The Clickture-Lite dataset generated from one-year click logs of Microsoft Bing image search engine, has been used.
To predict the relevance of images with respect to the text queries, we combined textual and visual information, by performing text-based and content-based image retrieval. An efficient implementation of deep Convolutional Neural Network, Caffe framework, was used to extract visual features. In particular, we extracted features from the output of the 6-th layer in the network, that is the first fully connected layer.
Decision has been taken using a Training set that consisted of triplets as follows: *<text query Q, image I, click count C>*, which means the image I was clicked C times in the search results of query Q in one year. Two strategies have been proposed: intersection and union. In the first case, using the text query, we retrieved textual similar triplets to the given query, through text-base image retrieval. Given the image, we retrieved visual similar triplets through content-based image retrieval. These procedures, allowed us to create two different sets of triplets, textual and visual candidates sets. We implemented the intersection among the triplets elements of the two sets. The union strategy has been split in two parts. Firstly, given the text query, textually similar triplets are retrieved and then we calculated visual similarity from the top-ranked elements. Secondly, given the image, we retrieved visually similar triplets and calculated textual similarity from the top-ranked elements. Finally, we combined the results of these two parts making their union.
To solve efficiency issues, we proposed an approach that index visual features using Apache Lucene, that is a text search engine library written entirely in Java, suitable for nearly any application requiring full-text search abilities. To this aim, a textual representation to convert image features into a textual form has been proposed. In this way, we were able to index them into an inverted index by means of Lucene, and to set up a robust retrieval system that com-

bined full-text search with content-based image retrieval capabilities.
To allow interactive use of the entire system, a multimedia information retrieval engine has been implemented. When the user inserts a textual query, the IR engine returns a list of textually similar images. It is also possible to perform visual searching by clicking on the images returned by previous search. Since the dataset contained base$_{64}$ encoded jpeg image thumbnail, a RESTful Web Service that decodes and returns them, has been implemented.
Different versions of our system over the Development set provided by Microsoft, have been evaluated. We compared the cosine similarity and the Euclidean distance, to calculate the visual similarity between images. Based on the evaluations on the benchmark dataset Inria and using the mean average precision (MAP) measure, we noted that it was better to use the cosine similarity. We also tried to make an exhaustive search, but it was too expensive in terms of time. For the intersection strategy, in order to select a good size, called K, for the visual and textual candidates sets, we tested our system with different values of this K parameter. The best results were obtained with K in the range [1024-4096]. We did not test the system with values larger than K=10000, because we noted that after 4096, the values tended to stabilize, because when K increased there was a kind of noise in the results. In other words, there may be no very similar images in the results of CBIR. This also applied to the textual-based image retrieval.

6.1 Future work

The results obtained by this thesis can be further improved.

Since a significant portion of image-query pairs in the Clickture dataset provided by Microsoft contains faces in the images and names in the queries, a face detection and recognition system could be introduced.

The power of Caffe framework could be exploit. Currently, we can not extract features from the query image "on the fly", because we have to specify various parameters (model architecture, batch size, mini batch size, name of features blob), create a file containing a list of files to process and take the features from the database created by Caffe itself using protobuf compiler. To overcome these problems, we decided to extract the deep features from all Development set images and we inserted them in an index. However, modifying the functions used by Caffe, we could extract features immediately, without using the index.

The union strategy could be improved, in particular the function used to calculate the textual similarity between the triplets within the visual candidates set and the given text query, could be modified and improved.

Bibliography

[1] Z. Xu, Y. Yang, A. Kassim, and S. Yan, "Cross-media relevance mining for evaluating text-based image search engine," in *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*, pp. 1–4, IEEE, 2014.

[2] S. Cen, L. Wang, Y. Feng, H. Bai, and Y. Dong, "Efficient image reranking by leveraging click data," in *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*, pp. 1–4, IEEE, 2014.

[3] Y. Hua, J. Shao, H. Tian, Z. Zhao, F. Su, and A. Cai, "An output aggregation system for large scale cross-modal retrieval," in *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*, pp. 1–6, IEEE, 2014.

[4] J. Raitoharju, H. Zhang, E. Ozan, M. Waris, M. Faisal, G.-y. Cao, M. Roininen, I. Ahmad, R. Shetty, S. Uhlmann, *et al.*, "Tut muvis image retrieval system proposal for msr-bing challenge 2014," in *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*, pp. 1–6, IEEE, 2014.

[5] C.-C. Wu, K.-Y. Chu, Y.-H. Kuo, Y.-Y. Chen, W.-Y. Lee, and W. H. Hsu, "Search-based relevance association with auxiliary contextual cues," in *Proceedings of the 21st ACM international conference on Multimedia*, pp. 393–396, ACM, 2013.

[6] L. Wang, S. Cen, H. Bai, C. Huang, N. Zhao, B. Liu, Y. Feng, and Y. Dong, "France telecom orange labs (beijing) at msr-bing challenge on image retrieval 2013," 2014.

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

[8] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," *arXiv preprint arXiv:1310.1531*, 2013.

[9] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, ACM, 2014.

[11] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8, IEEE, 2007.

[12] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *The Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[13] D. Grangier and S. Bengio, "A discriminative kernel-based approach to rank images from text queries," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 8, pp. 1371–1384, 2008.

[14] R. Smith, "An overview of the tesseract ocr engine," in *icdar*, pp. 629–633, IEEE, 2007.

[15] D. M. Christopher, R. Prabhakar, and S. Hinrich, "Introduction to information retrieval," *An Introduction To Information Retrieval*, pp. 151–177, 2008.

[16] P. Bolettieri, V. Casarosa, F. Falchi, L. Vadicamo, P. Martineau, S. Orlandi, and R. Santucci, "Searching the eagle epigraphic material through image recognition via a mobile device," in *Similarity Search and Applications*, pp. 351–354, Springer, 2015.

[17] `https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/`.

[18] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer vision–ECCV 2014*, pp. 818–833, Springer, 2014.

[19] `https://developer.nvidia.com/deep-learning-courses`.

[20] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[21] H. Jégou, M. Douze, and C. Schmid, "Improving bag-of-features for large scale image search," *International Journal of Computer Vision*, vol. 87, pp. 316–336, May 2010.

[22] A. Lucene, "A high-performance, full-featured text search engine library," *URL: http://lucene. apache. org*, 2005.

[23] J. Dong, X. Li, S. Liao, J. Xu, D. Xu, and X. Du, "Image retrieval by cross-media relevance fusion," in *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pp. 173–176, ACM, 2015.

[24] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. S. Corrado, and J. Dean, "Zero-shot learning by convex combination of semantic embeddings," *arXiv preprint arXiv:1312.5650*, 2013.

[25] Q. Song, S. Yu, C. Leng, J. Wu, Q. Hu, and J. Cheng, "Learning deep features for msr-bing information retrieval challenge," in *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pp. 169–172, ACM, 2015.

[26] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in neural information processing systems*, pp. 487–495, 2014.