Interactive Multimodal Learning on 100 Million Images

Zahálka, J.; Rudinac, S.; Jónsson, B.Þ.; Koelma, D.C.; Worring, M.

DOI
10.1145/2911996.2912062

Publication date
2016

Document Version
Final published version

Published in
ICMR'16

License
Article 25fa Dutch Copyright Act

Citation for published version (APA):
Interactive Multimodal Learning on 100 Million Images

Jan Zahálka\textsuperscript{1}, Stevan Rudinac\textsuperscript{1}, Björn Pór Jónsson\textsuperscript{1,2}, Dennis C. Koelma\textsuperscript{1}, and Marcel Worring\textsuperscript{1}

\textsuperscript{1}Informatics Institute, University of Amsterdam, The Netherlands
\textsuperscript{2}School of Computer Science, Reykjavik University, Iceland
{\{j.zahalka, s.rudinac, d.c.koelma, m.worring\}@uva.nl, bjorn@ru.is}

ABSTRACT

This paper presents Blackthorn, an efficient interactive multimodal learning approach facilitating analysis of multimedia collections of 100 million items on a single high-end workstation. This is achieved by efficient data compression and optimizations to the interactive learning process. The compressed $\iota$-I64 data representation costs tens of bytes per item yet preserves most of the visual and textual semantic information. The optimized interactive learning model scores the $\iota$-I64-compressed data directly, greatly reducing the computational requirements. The experiments show that Blackthorn is up to 105x faster than the conventional relevance feedback baseline. Blackthorn is shown to vastly outperform the baseline with respect to recall over time. Blackthorn reaches up to 92\% of the precision achieved by the baseline, validating the efficacy of the $\iota$-I64 representation. On the YFCC100M dataset, Blackthorn performs one complete interaction round in 0.7 seconds. Blackthorn thus opens multimedia collections comprising 100 million items to learning-based analysis in fully interactive time.

Keywords

Interactive multimodal learning; multimedia analytics; data compression; YFCC100M

1. INTRODUCTION

Multimedia collections are becoming the central information resource for a growing number of domains, such as physics, forensics, and marketing. This increases the need for methods for fast and insightful analysis of the data. Often, the analysts need to progress from the dataset to insight in hours or few days at most. For example, a journalist discovering stories about the Brussels terrorist attack in the ensuing flood of social media content cannot wait weeks before publishing the story; she needs at least preliminary insights right away. An essential ingredient of insight gain in multimedia analytics in any knowledge domain is interaction [17, 28]. Currently, multimedia collections can easily reach millions of items, and even larger datasets are common. Behemoths such as the Yahoo Flickr Creative Commons 100M (YFCC100M) dataset [21] were not so long ago considered inconceivable challenges, yet to truly advance large-scale multimedia analytics, techniques for analyzing such large datasets are needed. How can interactivity in multimedia analytics be assured facing the rapid growth of collection size?

Interactive learning techniques, such as relevance feedback or active learning, are a good fit for multimedia analytics, as they elicit training data labels from the user directly and train the model based on them [9]. This paradigm aligns well with insight, the goal of multimedia analytics. Insight is iteratively built by the analyst through interaction with all or most of the data in the collection and application of the analyst’s domain knowledge [17, 28]. Since interactive learning trains its model online based on the user’s interactions, the analysis is timely. How to enable interactive and multimodal learning to support large-scale multimedia analytics using modest computational resources (standard high-end workstation with 64 GB RAM and 16-core CPU) is the research question addressed in this work.

In this paper we introduce Blackthorn, an efficient interactive multimodal learning approach for collections of 100 million multimedia items.\textsuperscript{1} The architecture of Blackthorn is depicted in Figure 1. Blackthorn brings three main contributions: a multimedia content compression method allowing to fit billions of items into 64 GB of RAM; optimizations to the interactive learning process; and opening up the possibility of analytics on a collection of 100M multimedia items in interactive time (under 1 second per interaction round). To the best of our knowledge, no work has yet addressed fully interactive-learning-based exploration of an image dataset with 100 million items on a single workstation.

2. RELATED WORK

In response to the rapid growth of digital collections, a number of approaches facilitating more efficient search and exploration have been proposed. Interactive learning (e.g., relevance feedback) has been proven invaluable for improving multimedia information retrieval [9], and benchmarks such as the Video Browser Showdown were introduced [19]. However, the collections considered in these benchmarks are much smaller than the one considered in this work. In addition, most of current approaches employ entire computa-

\textsuperscript{1}As this paper focuses on efficiency of interactive learning, the name is inspired by the efficiency of the blackthorn shrub: as fire wood, it yields much heat with little smoke.
Figure 1: Interactive learning with Blackthorn. The components of interactive learning innovated by Blackthorn are marked orange.

sional clusters to facilitate analysis of the data in interactive time [6, 12]. The algorithmic efficiency component often boils down to reducing the number of items for which the similarity is computed, reducing the dimensionality of data representation, or simplifying the similarity computation.

Early work demonstrated that exact k-NN search techniques suffer from the curse of dimensionality [2], paving the way for approximate methods. Several recent works from the multimedia and computer vision communities rely on embedding visual features in binary space to compress the data and reduce the distance computation time [4, 8, 11, 15, 16, 24, 29]. LSH is a well-known scalar-based indexing technique [1]. The NV-tree is a scalar-based method, which has been shown to significantly outperform LSH while requiring about 6 bytes per descriptor [14]. Product quantization is a family of vector-based quantization schemes producing compact code-words by indexing low-dimensional subspaces independently [10, 13, 26].

While these approaches for scaling up image retrieval were proven effective in K-NN search, their applicability in analytic tasks remains limited for at least two reasons. First, binary hashes are not suited for classification, which is an essential component of most analytic platforms. Moreover, in many analytic tasks search and exploration are alternately performed, which requires updating, summarizing and repartitioning of the collection based on user interactions [28]. Such operations require preservation of the original vectors or a significant portion of information contained in them. Novel approaches are needed for building such representations for efficient deployment in analytic tasks.

3. DATA REPRESENTATION

In the interactive learning setting, it is desirable that the user understands the data representation. Hence, we focus on representations carrying semantic meaning. However, in case the collection comprises 100 million items, the memory requirements of such representations are prohibitive. Indeed, assuming 1000 visual concepts, 100 text topics, and 8 bytes (B) as the size of one floating-point number, the semantic representation of a 100M collection requires roughly 880 GB of RAM. We address this issue by introducing a data compression method that greatly reduces the representation size with only a modest information loss.

Semantic representations such as visual concept scores output by a convolutional neural network [29] or LDA topics in the text domain [3] are sparse, allowing the design of an efficient representation following the established sparse representation practices [25]. Thus, in each modality, most of the information contained in a feature vector with \( n_f \) features can be preserved by encoding only the top \( t_f \) features in compact bit fields, since \( t_f \ll n_f \). For the \( k \)-th feature, \( 1 \leq k \leq t_f \), we need to encode the feature ID (denoted \( i_k \)) and the respective score (denoted \( s_k \)). The bit cost for encoding the ID (denoted \( b_i \)) is \( \log_2 n_f \). The bit cost for encoding the score (denoted \( b_s \)) is \( \log_2 f \). Obtaining the bit field compressing the feature IDs of item \( I \) in each modality (denoted \( \mathcal{I} \)) is defined by Equation 1. Obtaining the bit field compressing the scores, \( \mathcal{S} \), is defined in Equation 2. Here, \( \lor \) denotes the binary OR operation, \( \ll \) the left bit shift operation, and \([\cdot]\) the rounding function.

\[
\begin{align*}
\mathcal{I} &= e_I \lor \bigvee_{k=0}^{e_I-1} i_k \ll (k \cdot b_i + b_c) \quad (1) \\
\mathcal{S} &= e_S \lor \bigvee_{k=0}^{e_s-1} \left[10^p \cdot s_k\right] \ll (k \cdot b_s + b_c) \quad (2)
\end{align*}
\]

The memory bit requirements for \( \mathcal{I}_j \) and \( \mathcal{S}_j \), \( b_I \) and \( b_S \), are then equal to \( e_I \cdot b_i + b_c \) and \( e_S \cdot b_s + b_c \), respectively. \( \mathcal{I} \) and \( \mathcal{S} \) can be decompressed to obtain the encoded features, setting the features outside the top \( t_f \) to 0. Let \( \land \) denote the bitwise AND operation and \( \gg \) the right arithmetic bit shift operation. Equation 3 describes the decompression of the number of encoded features \( (e_I) \), Equation 4 describes the decompression of the \( k \)-th feature ID, and Equation 5 describes the decompression of the respective feature score.

\[
\begin{align*}
e_I &= I \land \left(2^{b_i} - 1\right) \quad (3) \\
\delta(I, k) &= (I \gg (k \cdot b_i + b_c)) \land \left(2^{b_i} - 1\right) \quad (4) \\
\delta(S, k) &= (S \gg (k \cdot b_s + b_c)) \land \left(2^{b_s} - 1\right) \quad (5)
\end{align*}
\]

Since most current machines are 64-bit, it is desirable to choose \( t_f \) and \( p \) so that \( \mathcal{I} \) and \( \mathcal{S} \) align with the 64-bit architecture. Given a semantic representation with less than 1024 relevant visual concepts and text topics and \( p = 3 \), we can encode \( \mathcal{I} \) and \( \mathcal{S} \) each in \( t_f \) 64-bit integers, keeping the top \( 6t_f \) features. This setting enables a trade-off between minimizing memory and computational requirements (low \( \iota \)) on the one hand and information loss on the other (high \( \iota \)).

The resulting memory-efficient \( \iota \)-integer (\( \iota \)-I64) representation only requires 4\( t_f + 1 \) 64-bit integers per item: 2\( t_f \) for \( \mathcal{I} \) and \( \mathcal{S} \) per modality, as well as an additional integer for the item ID. With \( \iota = 1 \), the representation is 220x smaller than the uncompressed representation and allows fitting 1.5 billion items into 60 GB of memory. This makes \( \iota \)-I64 representation suitable for large-scale interactive multimodal learning.

4. INTERACTIVE LEARNING

Each interactive multimodal learning session consists of a number of interaction rounds, each with 3 steps. Firstly, a classifier is trained for each modality on the user-provided examples. Secondly, the unlabelled items in the collection are scored. Thirdly, the rankings per modality are aggregated and the top \( r \) results are returned to the user. A user
typically provides at most tens of examples per interaction round, regardless of the collection size. However, the time cost of steps 2 and 3 increases dramatically as the collection grows larger. We have alleviated this issue somewhat by data compression and an efficient parallel implementation in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the classifier’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.

Blackthorn utilizes a linear SVM classifier. To improve the classifier’s efficiency, we design a custom SVM scoring function that operates directly in the collection’s efficiency, we design a custom SVM scoring in C, but this does not solve the issue. In this section, we address Blackthorn’s algorithmic optimizations.
5 minutes is depicted in Figure 2. For all three evaluation measures, we compare \texttt{svm\_rf} with the best-scoring \( \nu, \iota \) configuration of \texttt{blackthorn}. We have tested \texttt{blackthorn} with \( \nu \in \{100, 200, \ldots, 1000\} \) and \( \iota \in \{1, 2, 3\} \). Varying \( \nu \) has some impact on precision, with \( \nu = 100 \) reaching roughly 75\% of the precision of \( \nu = 1000 \). Increasing \( \nu \) impacts the time per interaction round: \( \nu = 1000 \) costs roughly twice the time of the \( \nu = 100 \). Regarding recall over time, low \( \nu \) tends to outperform the higher values due to the lower time per interaction round. Regarding \( \iota \), we observe that increasing its value brings very minimal precision and recall gain, but has a negative impact on the time per interaction round. We believe that the low precision and recall gain is caused by insufficient decimal precision of the compressed values. We aim to investigate this further. Overall, we recommend choosing \( \iota = 1 \) in all cases, higher \( \nu \) for maximum precision, and lower \( \nu \) for maximum recall over time and speed.

The experiments clearly show that computational speed and efficiency are \texttt{blackthorn}'s forte. One interaction round of \texttt{blackthorn} on YFCC100M-California takes 0.05 ± 0.01 seconds on average across all evaluated \( \nu \) values, and the fastest configuration is 105x faster than \texttt{svm\_rf}. The information loss incurred by the \( \iota\text{-I64} \) representation turns out to be affordable. The highest-precision configuration of \texttt{blackthorn} keeps 92\% of the average precision with a speed-up of 45. The recall over time plot (Figure 2) clearly shows that \texttt{blackthorn} dramatically surpasses \texttt{svm\_rf}.

7. RESULTS ON YFCC100M

Our approach succeeded in the test of interactivity on the large dataset: on average across all \( \nu \) values, \texttt{blackthorn} takes 0.69 ± 0.01 seconds per interaction round. Figure 3 shows the development of recall over time on YFCC100M. Given the dataset’s prohibitive size, \texttt{svm\_rf} is infeasible. In order to compare the approaches, we extrapolate \texttt{svm\_rf}'s results on YFCC100M-California. Since \texttt{svm\_rf}'s computational cost is \( O(N) \), the 3.16 second performance on 1.2M becomes roughly 4.3 minutes. Hence, only the first interaction round needs to be considered. The base extrapolation is twice the recall value of \texttt{blackthorn}'s recall in the first interaction round, following the trend observed on YFCC100M-California. In addition, we report two \( \lambda \)-extrapolated \texttt{svm\_rf} baselines, where \( \lambda \in \{10, 100\} \) further multiplies the base extrapolation. We believe this more than compensates for any potential unverifiable factors influencing the true recall scalability. The results clearly show \texttt{blackthorn}'s efficiency with respect to recall gain over time: it is able to surpass all three interpolations in the first 5 minutes of the analysis.

The precision per interaction round on the YFCC100M dataset, averaged over all \( \nu \) values, is 0.003 ± 0.0001. Note that the low values are influenced by two important factors. Firstly, the size of the dataset brings an increase in noise and lower discriminative power of feature vectors. Secondly, the evaluation task is extremely difficult. Indeed, there are cases where the algorithm’s suggestions are evaluated as irrelevant, despite their semantic similarity to the provided positives. This is demonstrated in Figure 4. In a less strict setting, these suggestions could be easily deemed relevant. Despite the low values of precision, \texttt{blackthorn} does provide relevant suggestions even on the large dataset.

8. CONCLUSION

In this paper, we have presented Blackthorn, an efficient interactive multimodal learning framework which enables full interactive-learning-based analysis of a collection with 100M multimedia items. The data compression method of Blackthorn, \( \iota\text{-I64} \), is shown to reduce the size of multimodal features by a factor of 220 and preserve most of the information contained in the original features. Blackthorn yields a massive 105x speed-up in comparison to the classic relevance feedback paradigm. The experiments further show that Blackthorn is suitable for the analysis of the entire YFCC100M dataset. It is able to learn on the fly from the user-provided training samples, and one interaction round on the entire 100M collection takes less than a second. Its high efficiency and low resource cost would also support multi-category exploration with proactive suggestions by the system. The analysis can be performed on a single standard high-end workstation with 64 GB RAM and 16 CPU cores. In order to foster further research on YFCC100M, the Blackthorn software package has been made available as an open-source tool [27]. In conclusion, Blackthorn is a step forward towards fully harnessing the wealth of information contained in large-scale multimedia collections.
9. REFERENCES


