DeCP-Live: A Web-Interface for DeCP, a Distributed High-Throughput CBIR System

Gylfi Þór Guðmundsson
School of Computer Science
Reykjavik University
Reykjavík, Iceland
gylfig@ru.is

Christian Andreas Jacobsen
School of Computer Science
Reykjavik University
Reykjavík, Iceland
christianj15@ru.is

Hilmar Tryggvason
School of Computer Science
Reykjavik University
Reykjavík, Iceland
hilmartry15@ru.is

Björn Þór Jónsson
Computer Science Department
IT University of Copenhagen
Copenhagen, Denmark
bjorn@itu.dk

***Abstract*—A vast number of algorithms and methods are proposed and developed every year in the domain of indexing and searching multimedia documents. Much of this work results in published papers and some sources are made openly available, but rarely will you find a fully working end-to-end system that has been pre-installed, configured, and is ready-to-go on a virtual machine available for download. In this paper we present such a system, the DeCP-Live web interface, that is built on top of the distributed, high-throughput, content-based image retrieval algorithm DeCP. The virtual machine is ready-to-go as on it we have pre-installed services, indexed openly available datasets, binaries for DeCP and DeCP-Live as well as the source code.**

***Index Terms*—content-based image retrieval, end-to-end system, web-interface, open source, ready-to-go, virtual machine.**

I. Introduction

Great progress has be made in image and video processing in the last decade or so. Many innovative algorithms have been proposed and the datasets have been growing rapidly. To minimize variance, facilitate reproducibility, and make comparisons to prior work possible, it is common to see evaluation of multimedia algorithms being done on pre-calculated datasets such as the YLI-Corpus [7]. The downside to this approach is that often a fully working end-to-end system is not produced and thus a fully working demonstration cannot be created. This limits the usefulness of the contribution to the field of multimedia research, as an unnecessarily high barrier is created for others that would like to compare to the contribution, extend it, and build upon it. This is especially true if the code base is made publicly available, as then it is all the more tempting to extend it in some new way.

In this paper we highlight a ready-to-go virtual machine that has been made available for download with a fully functioning CBIR system that can do end-to-end search, using query images as input and producing ranked results that can be mapped into database images and displayed via a browser interface. Not only are the binaries and all necessary services pre-installed and configured but the source code has been made publicly available and is also included on the virtual machine.

The remainder of this paper is constructed as follows: In Section II-B we discuss DeCP and how it was extended into a fully functioning end-to-end CBIR system. We then, in Section III, describe DeCP-Live, the web-interface we built on top of DeCP, followed by a short Section IV on how we bundled everything in a ready-to-go virtual machine. Finally we conclude in Section V.

II. Distributed extended Cluster Pruning (DeCP)

DeCP is an distributed high-throughput content-based image retrieval algorithms that is based on vectorial quantization and approximate k-NN search. DeCP, and the non-distributed variant eCP, have been implemented multiple times, using a wide range of programming languages on multiple platforms and configurations: Non-distributed in C++ [4]; Hadoop using C++, Hadoop using Java [1]; and Spark using Scala and Java [2], [3]. Each time it has been extensively evaluated, using the largest datasets available, with the aim of ever pushing and testing the scalability of the algorithm.

In [2], [3], DeCP was evaluated using the 43 billion SIFT features of the YLI Feature Corpus [7], extracted from the 99.2 million still images of Yahoo's YFCC100M dataset [6]. Not only is this the largest publicly available dataset we know of but DeCP used and index designed for a 10x larger collection. The experiments were run on hardware from Amazon Web-Services (AWS), using a total of 50 nodes that had a combined total of 1600 virtual cores, 2.8TB of RAM and 30TB of SSD storage, all harnessed using the automatically distributed computing framework (ADCF) Apache Spark [5].

A. DeCP as an End-to-End CBIR System

In [2], DeCP is extended to be an end-to-end system that can take image files as input queries and produce ranked results of the most similar database images. Adding the SIFT feature extraction is done with the Java-based vision library, *BoofCV* (boofcv.org), that is placed as a pre-processing step in the search pipeline. Additionally, a post-processing step converts the k-NN results of each query vector from a given query image into a ranked result of the most similar images from the database. This process is in fact much like aggregating the k-NN results and using *TF-IDF* to rank the most frequently observed image identifier from the database images.

The high-throughput DeCP search process sacrifices low response time for high throughput by batching hundreds, or

even thousands, of query images into a single search. The primary reason for this is that DeCP was designed with the assumptions that all the feature data should be kept and that the feature collection would be so large that it must reside on disk. For a single query image, only a tiny subset of the whole dataset is relevant during search but retrieving it would result in many random I/Os that are hugely expensive. The two main advantages of batching are 1) that overlapping requests for the same cluster can be merged and 2) that the data can be read in sequential order, taking full advantage of prefetching and simplify the I/O access pattern.

B. Input to the DeCP Search Engine

The search engine is implemented in Scala and Java and runs as a job submitted to the Spark framework. When the DeCP engine is launched the initialization process loads the index, and the indexed database (RDD), as well as checking a specific folder for new images to added, see Section 4.2.3 in [3] for full details. After the initialization, the search engine runs an infinite loop on the Spark master, awaiting queries.

To keep things as flexible as possible we chose to use text-based files as the input to our engine. The format of the text file is a single *header line*, with colon separated query settings, followed by a path to the query images, one image per line. The header has three search parameters: $b : k : m$, where b is the search expansion, i.e. how many clusters to scan per query vector; k is the size of the k-NN created for each query vector and m is the maximum number of images in the ranked result list produced per query image. Further information and example input files can be found in the github repository for DeCP.

The batch is then submitted for searching by writing a *.batch* file into a specific input folder called **queries** that the search engine is monitoring.

C. Output from the DeCP Search Engine

Once the search has been completed, the *query.batch* file is deleted from the **queries** folder and the results written to a specific output folder called **results**. As there may be many query images in each query batch, a folder is created for the results of each batch. In this folder we will find a *batch.res* file that contains a *header line* as well as a list of all the query image result files. The header is a copy of the *.batch* header with two additions: $b : k : m : n : t$, where n is the number of query images in the batch and t is the total time in seconds that it took to process the batch.[1] The subsequent lines, following the header, are the names of each image result file that have also been created and written to the batch result folder. The result file for each query image has no *header line* with parameters. Instead each line is a path to an image, followed by a number that is separated by a colon. The first line is the path to the query image, followed the number of SIFT features extracted from the image. The subsequent lines are the m ranked results of database images, with the most similar image listed first. The colon separated value after the image path is the number of features that matched the query image.

[1]Note that the number of query images in the results may not match the number of lines of the input file. An example would be images where no SIFTs could be extracted or where the path to the input image was broken.

D. Interfacing with the DeCP Search Engine

In Section III we describe the web-based interface we built for DeCP that we call DeCP-Live. As DeCP is designed for running large query batches we do realize the limitations of our graphical interface however and the potential tediousness of manually selecting thousands of images for a large query batch. We will therefore highlight here the flexibility the text-based file format and demonstrate how a large batch can easily be created in Linux command line.

Let us assume that we have a large number of query images at the following path `/uploads/bigbatch` and that we would like to create a *.batch* file using no search expansion (b=1) keep 20 neighbors for each query feature (k=20) and retaining only the 10 most similar images for each query image (m=10). To create the *header line*, the simples way would be to use *echo* and redirect the output:
`$ echo "1:20:10" > bb01.batch`.
This will create an *.batch* with the search parameters but missing are still the query images with a full path. To add the missing information we use the *ls* command and do:
`$ ls /uploads/bigbatch/*.jpg >> bb01.batch`.
We need to make sure to use >> to append the image paths to the *.batch* file as well as using an absolute path with a wild-card ('*') in the *ls* command. What that does is make *ls* append the full path to each image file, just as we require. This way we have created a *query.batch* file with thousands of images in just a few seconds and all that remains is to copy the file to the **input** directory of the DeCP engine to start the search process.

III. DeCP-Live: the Web-Interface

On top of the DeCP engine we have built a web-interface, called *DeCP-Live*, that allows users to graphically create batch queries and view all existing batch-query results. The web interface is written in Angular, using Materialize, and runs on Apache HTTP Server with additional content served by a NodeJS server.

In the web-interface we have a menu at the top of the page, right hand side. In it we find three options: 1) *Admin*, for the administration menu; 2) *Query*, allows users to create a new batch query and 3) *Results*, that allows the user to browse the existing query-batch results.

The administration menu has only two options: 1) save the current state of the indexed database (RDD) to disk, see Section 4.2.3 in [3], and 2) halt the search engine by terminating the Spark job. Below we describe the *Query* and *Results* options.

A. Submitting a query

Figure 1 shows the *Query* interface, where users can create a new batch-search and submit it to the DeCP search engine for

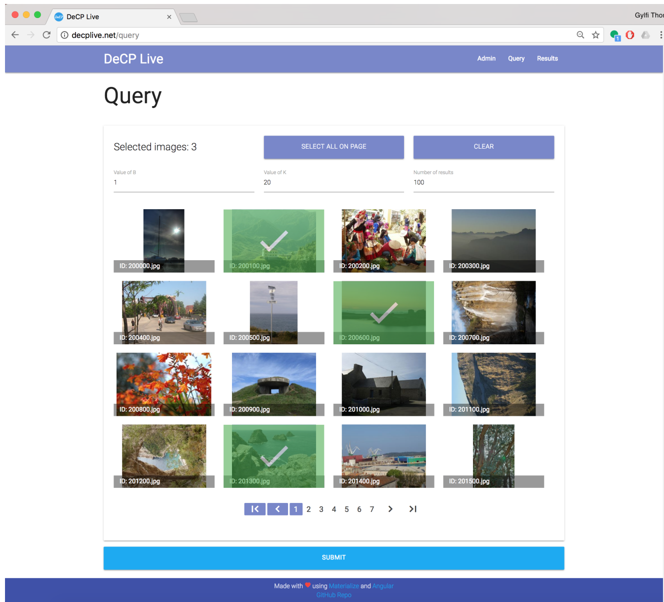

Fig. 1. The **Query** page is used to create a batch query in the *DeCP-Live* web-interface. First the search settings (b, k and m) for the batch are set. Then images to include in the batch can be selected, indicated by a green check-marker.

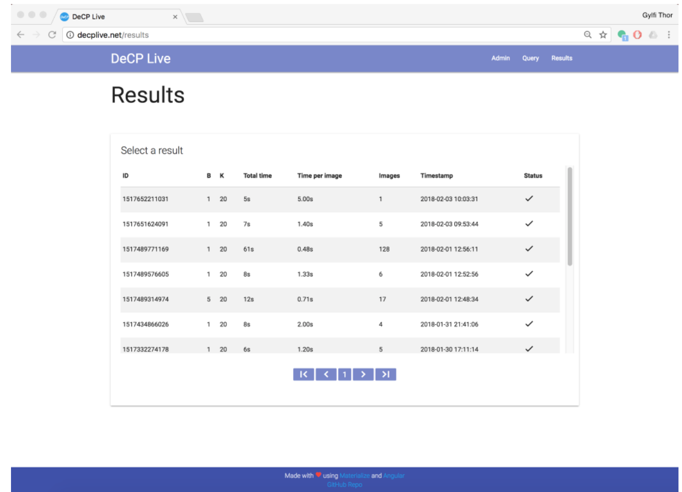

Fig. 2. The **Results** page shows all the available results in DeCP's **results** folder regardless of where the query originated.

processing. On the page we find *Selected images*, that indicates how many images have been selected for this batch, and two buttons: *Select all on page*, that will select every image currently displayed, and *Clear*, that will unselected all images and reset the search parameters to the default values. Next, we have the three parameters of the batch-search settings, b for how many clusters to scan per query vector, k the size each query vectors k-NN and m the number of similar images to return for each query image. The user can then select images to include in the search (indicated with the green check-mark overlay) from the image grid. The grid is populated from a folder on disk of available query images (*aqi*), showing 16 images at a time. A paging menu is used to allow the user to browse all the images in *aqi* and the *Select all on page* button allows the user to quickly select all the images on each page. Once the user is happy with his batch-query, the *Submit* button instructs DeCP-Live to generate and write a *.query* file to DeCP's **queries** folder.

For security reasons we decided not to allow users to upload to *aqi* via the web-interface. Adding new query images is trivial however as writing images to *aqi* and refreshing the *Query* page will suffice to make them available for selection.

B. Displaying results

As was discussed in section II-C, the results of a query batch will be written to a specific **results** folder by DeCP. The web-interface monitors the result folder and notifies the user every time a new result is available. In Figure 2 we see the layout of the **Results** page on a system that has several query batch results available. The primary content of this page is the results list, populated by reading the *header* of each *batch.res* file available. As we can see, DeCP-Live uses a timestamp value to identify each query batch. In the result list we find a quick overview of the statistics for each batch, such as the search parameters b, k and m, the processing time, both total and per image, the number of images in the batch, and when the search was issued. The final marker, *Status*, indicates whether the search has been completed or is still executing on the search engine. The layout gives a clear overview of the statistics for the various batch results available. The list is clickable and each line of the list is a link to the **Result - BatchID** page, see Figure 3. This page is populated from all of the content of the *batch.res* file. The list has two modes: *Rows* and *Grid*.

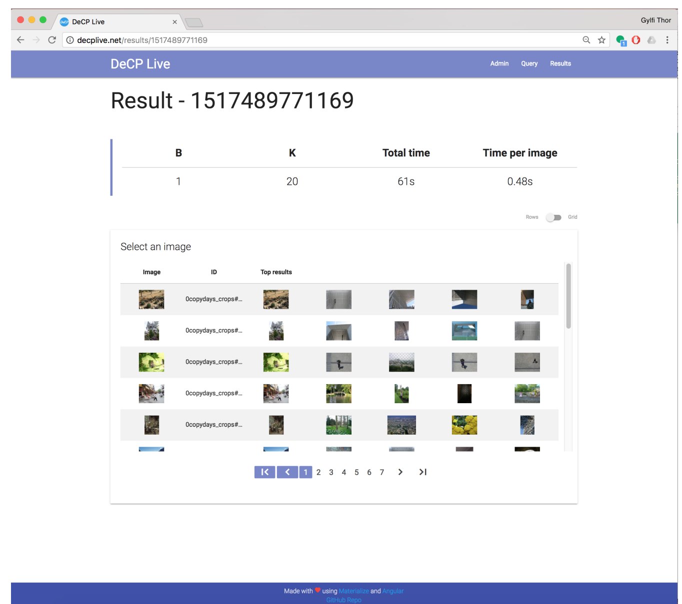

Fig. 3. Batch results in *Rows*-view where each row is a query image in the batch. Notice the sneak-peak preview of the top five results for each image. This allows a visual overview of the image-level results of the whole batch.

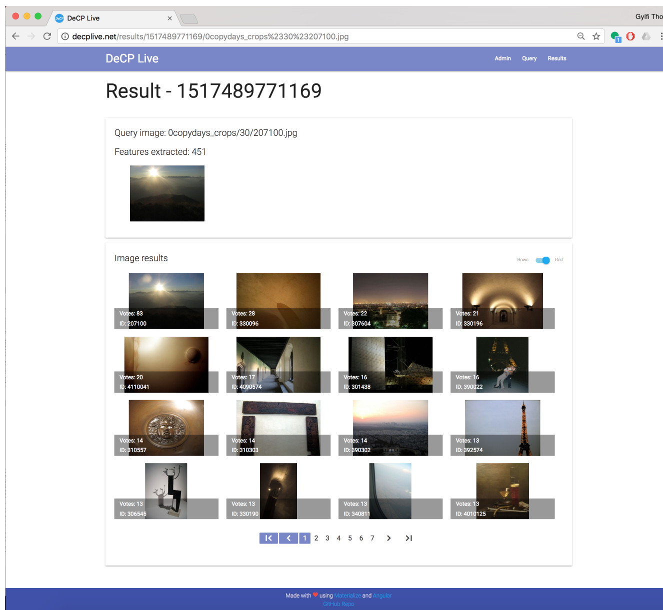

Fig. 4. Query image result in the default *Grid*-view. Displayed at the top is the query image as well as the number of features extracted and searched. Overlaid on each result image is both the number of *Votes* as well as the ID of the database image.

The *Grid* mode is much like that of the **Query** pages, with large thumbnails of the images while the *Rows* mode is the one displayed in Figure 3. In this mode we display a row in a list for each query image. The list starts by showing a small thumbnail of the query images followed by it's ID. In addition we find a sneak-peak preview of the top five most similar database images (populated by reading the result file for this query image). This gives us the ability to quickly preview visually the image-level results without having to access each image-level result file individually.

To see the certainty level of the search engine, i.e. the margin by witch the ranking of similar images was established, we must click the query image in the list and open the **Result for Query image** page, see Figure 4. The **Result for Query image** page is populated by reading the full *.res* file for that given query image. As we can see in Figure 4, the page defaults to the *Grid*-view. We prefer this view as it shows the larger image thumbnails, and retain the important information as it can be overlaid onto the images.

The database images in the result are ranked by the number of *Votes*, calculated by counting how frequently this database image identifier was observed in the k-NNs of the query features. The more *Votes* the more similar the images are. We should keep in mind however that the number of features extracted from each image varies greatly and thus very good results may frequently be far from a 100% match. The relative scale of the *Votes* between the top-ranked images is a much better indicator of similarity.

IV. Ready-to-Go Virtual Machine

To make our work as accessible as possible to others, we have made the full code base to both DeCP and DeCP-Live open source.[2] In the code repository you will also find a link to a ready-to-to virtual machine (Oracle's VirtualBox) that has everything already installed and configured. After launching the VM, the DeCP-Live interface can be accessed on the host machine through `http://localhost:9080`, as port 80 has been forwarded to the host as port 9080. To be able to use DeCP however the search engine must be started by running a script on the VM, see instructions on the git repository.

Included on the VM are also two datasets: 1) HolyDays that consists of 1491 images and 2) CopyDays that has 158 original images and 3075 augmented images.[3] The indexed database uses a 200k wide, 3-level index, that has been used to index the 1491 images of Holydays + the 158 originals of the CopyDays. The 3075 augmented Copyday images are used as the query images. Adding more images to either the database or the query set is easy; for detailed instruction please see the git repository for DeCP.

V. Conclusions

In this paper we have presented an end-to-end distributed search engine for content-based image retrieval, a highly flexible text-based communication protocol and a web-based graphical interface. Not only is all of the code open source but we have also been bundled it all into a demo and made a ready-to-go virtual machine. We hope that this will make our work accessible to the wider community and make it easier for others to build upon it.

References

[1] D. Moise, and D. Shestakov, and G. Þ. Guðmundsson, and L. Amsaleg, "Indexing and Searching 100M Images with Map-Reduce," Proc. of the ACM Int. Conf. on Multimedia Retrieval (ICMR), pp. 17–24, 2013.

[2] G. Þ. Guðmundsson, and B. Þ. Jórnsson, and L. Amsaleg, M. J. Franklin, "Prototyping a Web-Scale Multimedia Retrieval Service Using Spark", Accepted for publ. in ACM Trans. on Multimedia Computing Communications and Applications, vol. 14, 2018.

[3] G. Þ. Guðmundsson, and B. Þ. Jórnsson, and L. Amsaleg, M. J. Franklin, "Towards Engineering a Web-Scale Multimedia Service: A Case Study Using Spark", Proc. of ACM Multimedia Systems Conf. (MMSys), pp. 1–12, 2017.

[4] G. Þ. Guðmundsson, and B. Þ. Jórnsson, and L. Amsaleg, "A large-scale performance study of cluster-based high-dimensional indexing", Proc. of the Int. workshop on very-large-scale multimedia corpus, mining and retrieval (VLS-MCMR), pp. 31–36, 2010.

[5] Zaharia, M. and Chowdhury, M. and Franklin, M. J. and Shenker, S. and Stoica, I., "Spark: Cluster computing with working sets", Proc. of the USENIX Conf. on Hot Topics in Cloud Computing, pp. 10-10, 2010.

[6] Thomee, B. and Shamma, D. A. and Friedland, G. and Elizalde, B. and Ni, K. and Poland, D. and Borth, D. and Li, L.-J., "YFCC100M: The New Data in Multimedia Research", Communications of the ACM, vol. 59, 2016.

[7] J. Bernd, D. Borth, C. Carrano, J. Choi, B. Elizalde, G. Friedland, L. Gottlieb, K. Ni, R. Pearce,D. Poland, K. Ashraf, D. A. Shamma, B. Thomee, "Kickstarting the Commons: The YFCC100M and the YLI Corpora", Proc. of the Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions (MMCommons), pp 1–6, 2015.

[2]Available on github repositories https://github.com/elgerpus/DeCP and https://github.com/elgerpus/DeCP-Live

[3]Available for download at http://lear.inrialpes.fr/~jegou/data.php