Implementation of image search system for event photo albums

Konstantin Sidorov*, and *Aleksandr* Koshkarov

Astrakhan State University, Project Office "Artificial Intelligence", 20a Tatischev Street, #402, 414056, Astrakhan, Russia

Abstract. The growing amount of image data (produced in a multitude of different ways – for example, like photo albums from various events) and lack of a direct search method for image information frequently create a situation in which end users of the service that publishes such data want to find themselves but cannot do it faster than looking through all entries one by one. The objective of this study is to develop and implement an approach that would be able to process this kind of data source and extract images similar to those provided by end-users in an on-demand manner. To this end, the modern methods for extract and compressing faces are introduced and applied.

1 Introduction

In the modern world, the large amount of data generated in different domains frequently aids the development of new data-driven products and (even more importantly) using them to extract valuable insights from the acquired data. This, in turn, makes the underlying business processes more effective – a pattern also known as *data-driven decision making* which has been widely studied [1,2].

One of the most widely studied and used classes of data sources is image datasets. While it is relatively easy to produce this kind of information (by, e.g., photographing), *retrieving* from such data source is much harder than, e.g., from a pool of texts. One of the practical setups – which is the key object of study in this paper – is a photo search in a photo album of an event (or, if the hosting organization is large enough, from multiple photo albums).

For instance, this problem has occurred at Astrakhan State University (ASU) – since this university hosts a wide variety of events, it also employs a full-fledged news agency, which is fittingly named *ASU Media* [3]. Since ASU Media covers all major university events and most minor events, it has accumulated a large photo archive in a form of photo albums in VK – a prime Russian social network – and a wide interest from people related to ASU. An increasingly common complaint encountered among both the university staff and the students was the lack of a convenient way to find stories related to some people. In particular, the motivation behind this complaint has frequently been a need to find professional photography of oneself – the stories are normally accompanied by the photo album made by ASU Media photographers. To tackle this problem, the following solution has been proposed:

- Collect the relevant data from the news stories.

* Corresponding author: sidorov.k@asu.edu.ru

- Develop a search engine that could find a photo (along with the story) by the face image.
- Provide a convenient user interface to the search engine.

The objectives of this study are listed as follows:

1. Describe the data source used for the other stages and outline the key problems that might have an impact on further implementation.
2. Implement the pipeline that could be used both for processing the given dataset and for real-time usage in the user interface.
3. Evaluate the performance of the pipeline and the requirements for the hardware that is used to run it.
4. Develop requirements for the user interface and implement the UI conforming to them.

The rest of the paper is structured in the following way:

- Related work on the subject in question is outlined in Section 2.
- Section 3 describes the data source and the technical challenges arising from it.
- A high-level description of the algorithmic approaches used in the final solution is given in Section 4.
- The key requirements for the user interface and the resulting UI are described in Section 5
- Details and corner cases of implementation are described in Section 6.
- Finally, Section 7 gives a summary of the contents and results of this study and highlights several fruitful directions for future work.

2 Related work

The problem of searching for similar images in an image database has been studied in many different contexts. Closest to our approach was described by Wang, Otto, and Jain in [4], where they proposed an idea to reduce the search task to the problem of finding approximate nearest neighbors – however, their approach, unlike ours, heavily relies on using deep convolutional models. Another approach to this problem has been described in [5], which, unlike other described approaches, also solves a problem of *labeling* images – i.e., assigning images from the database into meaningful (potentially not disjoint) categories – to this end, the authors reduce the labeling problem to the quadratic programming problem, which is itself relaxed by clustering of images. Bach et al. in [6] have outlined a more "manual" approach to the search problem – namely, they describe an architecture in which the feature extractors applied to the images are pluggable, hand-written (authors name them *primitives*) and have variance in their granularity (i.e., include both generic features, such as color and texture, and domain-specific features).

3 Dataset description

As described above, the main data source in this study is provided by [7]. However, since this is an image dataset, it also provides several challenges, such as:

1. *Data volume*. The provided dataset contains images from all news stories, which results in a large volume of data – at the time of development, the image directory's size had been over 8 Gb. For this reason, any viable solution should be able to run some sort of compression on the input data to enable any kind of efficient search.
2. *Data collection*. Since the data source is a social network group – and not, for instance, BLOB storage – input images should be collected in a way that does not violate Terms and Conditions [8] of API.

3. *Retrieval speed*. While any required pre-processing could be done on the raw dataset "as-is", the parts that are used in the communication with the user should be executed on some indexed and/or compressed version of the dataset.

4 Methodology

As mentioned in the previous section, this study involves a considerable computational load for searching faces on images in a large image pool. For this reason, the main focus of this section is on computational methods for:

- detecting a face on an image,
- compressing a face image into a low-dimensional numerical vector,
- finding an approximate nearest neighbor.

4.1 Face detection

The main concept used in this study is *histogram of oriented gradients* (HOG) [9]. HOG is an efficient feature extractor which is used as follows:

- First, for a given image its *intensity gradients* are computed – intensity gradients can be characterized as vector quantities having the direction of the largest intensity decrease. Since gradient vectors are orthogonal to the corresponding level sets [10], this can be considered as a method for edge detection. However, image is presented as a grid of discrete pixels (and not as a continuous structure), the gradient has to be approximated by convolutions – in [9] several methods are considered, but the most balanced (i.e., fast but not decreasing detection quality) of those involves a finite difference approximation (see also [11]) with masks $[-1,0,+1]$ and $[-1,0,+1]^T$.
- The image is partitioned by blocks of 8×8 pixels, and for each block gradients are grouped by their direction in 9 bins (0—20 degrees, 20—40 degrees, …, 160—180 degrees). This can be considered as the smoothing procedure which reduces differences between neighboring pixels and differences in the gradient direction.
- Next the *block normalization* is applied to the histogram values – the vector containing the histogram values is divide by some scalar to ensure that its norm is close to 1. For example, if $\mathbf{v}$ is a vector with histogram values and Euclidean norm is used, the normalization procedure is given by

$$\mathbf{v}_{norm} = \frac{\mathbf{v}}{\sqrt{||\mathbf{v}||_2^2 + \epsilon}}, \qquad (1)$$

where ϵ is a small constant used to ensure that division by zero doesn't happen.

The complete run of the HOG is also illustrated with the figure 1 from the documentation of [12].

Given a HOG representation of the image, the following classification problem is posed:

- The dataset is composed of images of faces (and only faces – if the images contain several faces, its "windows" containing faces are labeled and extracted) and images of non-faces (which could be achieved by *negative sampling* – selecting random image windows and labeling them as non-faces; most of the time it will be so).
- The input variables are HOG features of the image.
- The output is a binary variable with value "face" and "not face".

This problem turns out to be relatively simple and thus could be solved by various methods of supervised learning – a computationally efficient and rigorous way to do it is to use the support vector machine [13].

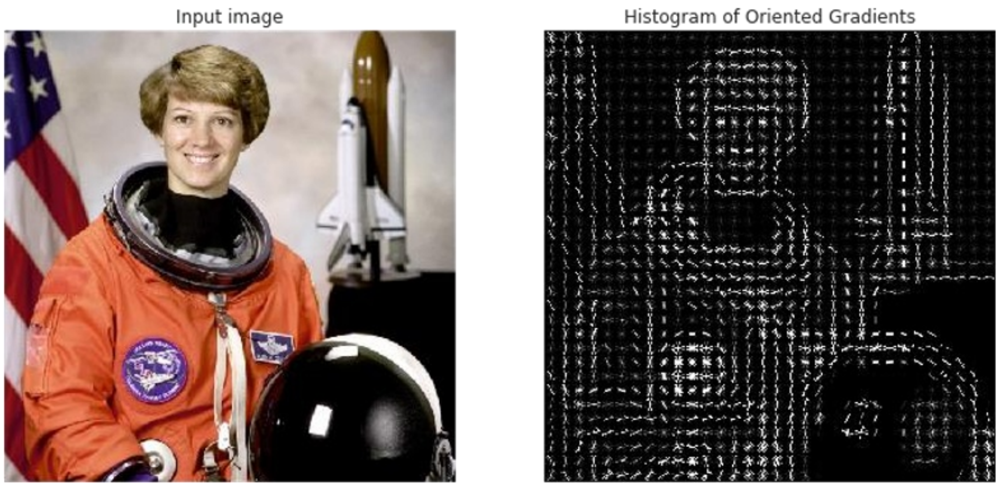

Fig. 1. A complete run of the HOG feature extractor. Left: original image. Right: output features

4.2 Face compression

While the previous section describes approaches to extracting image segments corresponding to faces, using the resulting pixel grid directly as a representation for a search engine is problematic for the following reasons:

1. This representation is not restricted by dimensionality – i.e., an image window with a face can have arbitrary height and width. As a result, it is unclear how to introduce any kind of similarity function on this representation of faces.
2. This representation is not invariant to lighting change or head rotation – photographs of the same person shot from different angles will have vastly different pixel representation.

For this reason, the face images have also been compressed using FaceNet architecture [14] – it is a deep learning architecture which has the following distinctive features:

- The training dataset is a set of *triplets*, which are composed of *anchor*, *positive* and *negative* images. As a result, the objective for any learning algorithm is to learn similar representations in "anchor/positive" pair and different representation in "anchor/negative" pair.
- The learning function is a *triplet loss* which is defined as:

$$\sum_i \max\left\{0, \left[f(x_i^a) - f\left(x_i^p\right)\right]^2 - [f(x_i^a) - f(x_i^n)]^2 + \alpha\right\}, \tag{2}$$

where the triplet is defined as (x^a, x^p, x^n), and f is a parametric function that maps face images to their vector representations (for example, a deep neural network).

As a result, a function $f: \mathbb{R}^{H\times W} \to \mathbb{R}^K$ has been learned – in this study we used $K = 128$ which results in representing faces as a vector of 128 numbers.

4.3 Nearest neighbor search

The final problem which has to be solved is the following: given a large source of vectors from $\mathbb{R}^K$ and a new vector (corresponding to the user input), how to find a vector (i.e., a face from one of the images) from the data source closest to the mew vector (i.e., corresponding to the most similar face).

The data structure used for running an *exact* nearest neighbor search is *K-D tree* [15] which is a binary search tree in which a leaf corresponds to the vector, and inner nodes correspond to the cutting planes.

While this data structure is a common choice for low-dimensional problems, it has the following problem when applied to K-dimensional space: if the number of vectors is comparable with 2^K, K-D tree tends to degenerate in a convoluted implementation of the linear search. For this reason, we have to resort to solving an *approximate* version of the nearest-neighbor problem.

For the approximate K-D tree several data structures based on the idea of *locality-sensitive hashing* – in particular, we used the technique implemented in the Annoy library [16] which involves the following heuristic modification of the K-D tree: instead of running the exact procedure for selecting a cutting plane, sample two vectors from the data source and use the equidistant plane as the cutting plane.

5 User interface

Since the results of this study were intended to be used by ASU students and staff – who are not necessarily proficient in IT – the key requirements to the user interface were:

- *Familiarity* – the UI should be similar to a familiar, recognizable application (or, better yet, embedded in it).
- *Simplicity* – the user should be able to run an image search in a simple, intuitive fashion.
- *Presence of a stable API* – since the resulting service is intended to be maintained by other teams of developers, the user interface should have a straightforward well-documented application programming interface.

Considering all factors described above, the user interface was implemented inside of the Telegram messenger [17], which is well known in Russia not only for its direct merits but also for its political agenda [18]. Another upside of Telegram as a platform is its simple API for programming *bots* – third-party applications that run inside Telegram and interact with Telegram users using normal messages (including image messages) and (optionally) command messages with special syntax [19].

The typical interaction with the chatbot id depicted in Figure 2.

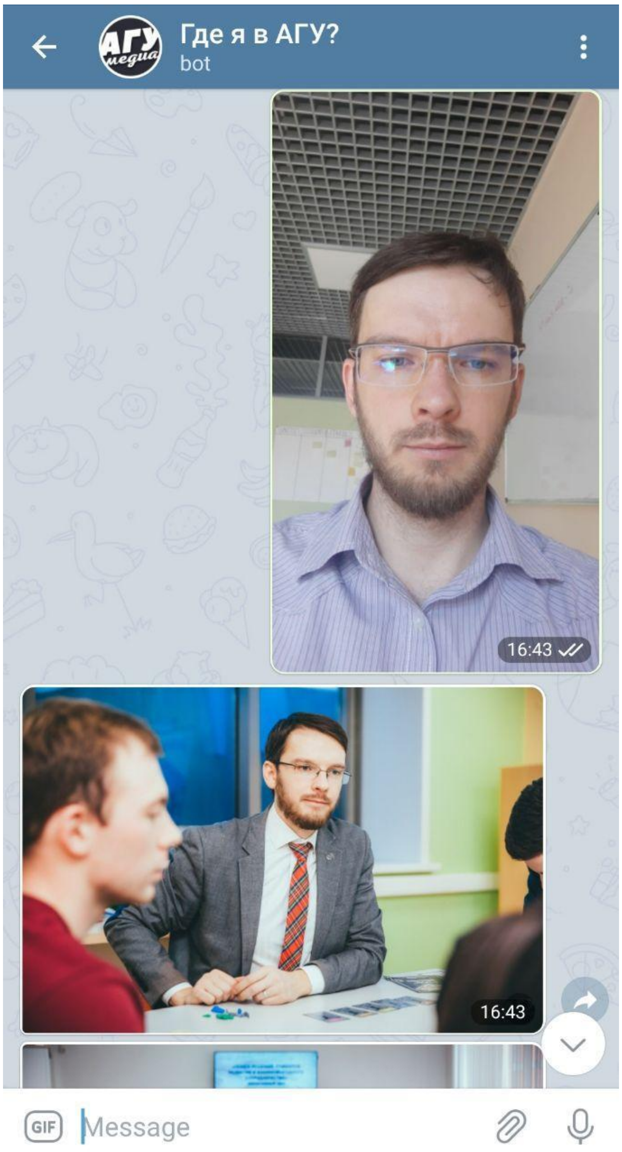

Fig. 2. An example of the typical interaction with the Telegram bot. Left: several photos found in the archives of ASU Media. Right: a user input.

6 Implementation details

The final pipeline consists of two major parts – *warm-up* stage and *interaction* stage. When the service is started, the *warm-up* stage is execute – namely:

1. If there are any images in the dataset directory for which compressed vectors (see Section 4.2) haven't been stored, then for every such image:
    a. Detect all faces and store their bounding boxes as described in Section 4.1.
    b. For each found face, run the compression procedure from Section 4.2 and store the resulting vectors.

2. Load the stored face vectors into the data structure for the nearest neighbor search.

This stage is run only once per bot lifetime.

After the warm-up stage, the *interaction* stage is initiated. After a new message with an image arrives, it executes the following steps:

1. Extract face positions and compress them in the same way it is done during the warm-up.
2. Find K closest faces from the dataset (the word "closest" here means "having the smallest Euclidean distance in the space of compressed vectors") and return them with any required metadata attached to them.

Figure 3 outlines the pipeline used in the resulting solution.

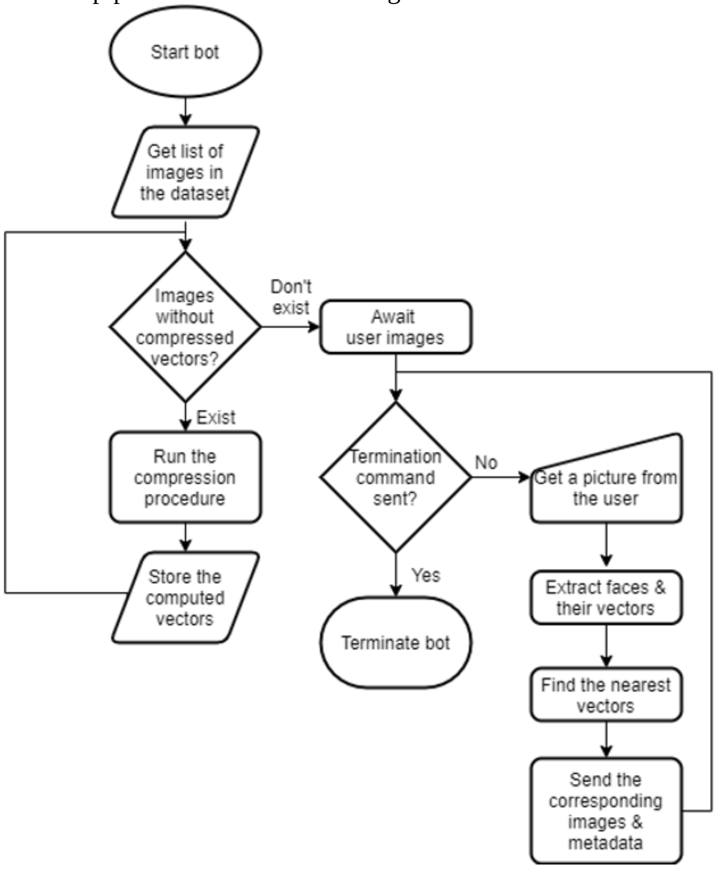
Start bot
Get list of images in the dataset
Images without compressed vectors?
Don't exist
Await user images
Exist
Run the compression procedure
Store the computed vectors
Termination command sent?
No
Get a picture from the user
Yes
Terminate bot
Extract faces & their vectors
Find the nearest vectors
Send the corresponding images & metadata

Fig. 3. A flowchart that visualizes the major stages in the lifetime of the chat bot.

The entire service has been deployed as a Python service using the following third-party libraries:

- `python-telegram-bot` — Python API for interacting with Telegram Bot API. [20]
- `dlib` – a library (and an API for it) containing implementations of machine learning algorithms in C++. [21]
- `face-recognition` – a Python interface for `dlib`. [22]
- `annoy` – a library for running an efficient heuristic approximation for the nearest neighbor search. [16]
- `requests` – a well-known Python library for running HTTP queries from Python environment. [23]

- Finally, relational database `SQLite` [24] and Python module `sqlite3` for interaction with it are used to implement the persistency layer.

The compressed dataset at the time of writing was stored in a single SQLite file with a size of 40 Mb (cf. original 8 Gb dataset).

7 Conclusion and future work

In this study, we considered an approach for searching images in the collection of events' photo albums based on face recognition technologies. It should be noted that the described approach allows not only to automate the image search process but also to implement it in a fast and concise manner.

One of the possible directions for future work is integrating modern deep learning methods not only in the face compression stage (which is a relatively simple operation since it involves transforming a small image into an even smaller vector) but also in the face detection stage (which is much more computationally intensive part of the pipeline). However, this direction of work also involves balancing between two opposite objectives – higher precision of detection and better run time. It should be also noted that one of the major factors in this decision is the environment in which the service is executed – for instance, a single-machine CPU-only (i.e., without GPU) environment mostly precludes usage of deep learning methods in this stage.

Other lucrative directions for future work include:

- Development of Web-based user interface – while Telegram is a well-known platform, it is still a platform having its policies; Web-based UI, on the other hand, can be self-hosted. One of the possible approaches to this is implementing the Progressive Web Application [25].
- Resource-adaptive implementation of the pipeline – as mentioned above, different environments pose different constraints on the technical solutions, and for more constrained environments many deep learning approaches quickly become infeasible. For instance, a class of approaches known as *knowledge distillation* [26] may be considered for this task.

References

1. E. Brynjolfsson, L. M. Hitt, and H. H. Kim, *Strength in Numbers: How Does Data-Driven Decisionmaking Affect Firm Performance?* (2011)
2. J. A. Marsh, J. F. Pane, and L. S. Hamilton, *Making sense of data-driven decision making in education: Evidence from recent RAND research* (2006)
3. *ASU Media*
4. D. Wang, C. Otto, and A. K. Jain, IEEE Transactions on Pattern Analysis and Machine Intelligence **39**, 1122 (2017)
5. D. Wang, S. C. H. Hoi, Y. He, and J. Zhu, IEEE Transactions on Knowledge and Data Engineering **26**, 166 (2014)
6. J. R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R. C. Jain, and C.-F. Shu, *Virage Image Search Engine: An Open Framework for Image Management*, in *Storage and Retrieval for Still Image and Video Databases IV*, edited by I. K. Sethi and R. C. Jain (SPIE, 1996)
7. *ASU Media VK group*
8. *VK Platform Terms & Conditions*

9. N. Dalal and B. Triggs, IEEE (2005)
10. V. A. Zorich, *Mathematical Analysis i* (Springer Berlin Heidelberg, 2015)
11. R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations* (Society for Industrial; Applied Mathematics, 2007)
12. S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, *Scikit-Image: Image Processing in Python*, PeerJ **2**, e453 (2014)
13. B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A Training Algorithm for Optimal Margin Classifiers*, in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory - COLT 92* (ACM Press, 1992)
14. F. Schroff, D. Kalenichenko, and J. Philbin, IEEE (2015)
15. J. Goodman, *Handbook of Discrete and Computational Geometry* (Chapman & Hall/CRC, Boca Raton, 2004)
16. E. Bernhardsson, *Annoy: Approximate Nearest Neighbors in c++/Python* (2018)
17. *Telegram Messenger*
18. A. Akbari and R. Gabdulhakov, Surveillance & Society **17**, 223 (2019)
19. *Telegram Bot API* (2020)
20. *python-telegram-bot/python-telegram-bot: We have made you a wrapper you can't refuse*
21. D. E. King, *Dlib-Ml: A Machine Learning Toolkit*, Journal of Machine Learning Research **10**, 1755 (2009)
22. A. Geitgey, *ageitgey/facerecognition: The world's simplest facial recognition api for Python and the command line*
23. K. Reitz, *Requests: HTTP for Humans*, https://requests.readthedocs.io/en/master/
24. R. D. Hipp, *SQLite* (2020)
25. D. A. Hume, *Progressive Web Apps*, 1st ed. (Manning Publications Co., USA, 2017)
26. G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, Neural Information Processing Systems, 742 (2017)