Using Text Queries to Look Up Unlabeled Images: A Command-Line Search Tool Based on CLIP

Yurij Mikhalevich
Lightning AI
Dubai, United Arab Emirates
email: yurij@mikhalevi.ch

***Abstract*—This paper presents a practical, scalable implementation of an image search engine using OpenAI's Contrastive Language-Image Pre-Training (CLIP) model. The method provides a convenient Command-Line Interface (CLI) and introduces a cache layer powered by SQLite 3 relational database management system (RDBMS) that facilitates efficient repetitive image searches within extensive image databases using natural language queries. The method's effectiveness was evaluated on ImageNet-1k and CIFAR-100 datasets, yielding a 31.17% top-1 accuracy on the ImageNet-1k train set and 55.15% top-1 accuracy on the CIFAR-100 test set. The scalability study showed that indexing time scales linearly with the number of images, and image search time increases only slightly; for example, on an Apple M1 Max CPU, indexing over a million images took 26.36 times more than indexing 50,000 images, while querying the larger image set took just 2.75 times longer. This approach is particularly relevant for industries managing vast volumes of visual data, such as media and entertainment, security, and healthcare.**

***Keywords*—image search, image indexing, photo management, computer vision, natural language processing**

I. Introduction

The release of the Contrastive Language-Image Pre-Training (CLIP) model by OpenAI in 2021 has generated significant attention in the field of Natural Language Processing (NLP) and Computer Vision (CV). This model has demonstrated the ability to learn state-of-the-art image representations from a massive dataset of 400 million (image, text) pairs that were collected from the Internet. CLIP can predict the most relevant text snippet, given an image, using natural language instructions without explicitly optimizing for the task. This zero-shot ability is similar to that of GPT-2 and 3 [1]. The authors of CLIP have demonstrated that CLIP performs as well as the original ResNet50 on ImageNet in a zero-shot manner without using any of the original 1.28M labeled examples. This accomplishment overcomes significant challenges in computer vision.

In addition to these capabilities, CLIP allows researchers to perform image searches using natural language queries. This article explores this particular application of CLIP.

The exponential growth of data, particularly in the form of images, makes this research especially relevant. With the proliferation of digital devices and the Internet, more and more images are being produced and shared online every day. As a result, it is becoming increasingly challenging to find specific images manually. A robust and efficient image search solution can help users and businesses quickly find the images they need amidst this growing sea of data. Moreover, as the volume of photos being shared online continues to increase, it becomes increasingly critical to protect intellectual property and online security. An efficient image search solution can help identify and remove images that violate copyright laws or contain sensitive or inappropriate content. Therefore, with the growing amount of data and images being created, an efficient image search solution is becoming increasingly necessary and relevant.

In Section 2, the paper explores related works in the field of image search. Section 3 describes the proposed method for image search, which is using CLIP. Section 4 presents the implementation details of the proposed method. Section 5 discusses the performance of the proposed method. Section 6 presents the search quality measurement results of the proposed method. Finally, Section 7 concludes the paper and discusses future work.

II. Related Works

There exist multiple methods and solutions for image search.

Traditional image search methods rely on bag-of-features, which are sets of features that describe the contents of an image. These features can be manually specified by annotating the image with descriptive labels. For instance, Fiedler et al. [2] proposed an image tagging software that helps users enter the labels efficiently. Another way to obtain labels is to use labels injected into the photos by camera software, as explored by Tesic [3], who examined camera metadata for consumer photos.

Mobile phone software can also inject useful metadata within photos. Kim et al. [4] explored how this metadata can be leveraged to effectively manage and search photos using mobile smartphones. This metadata can include information about the location, time, or type of camera used, among others. By using such metadata, it's possible to automatically generate labels for images.

One way to automatically generate labels is by using image recognition algorithms, such as Convolutional Neural Networks (CNNs). CNNs can be specifically trained to recognize and classify images based on their contents. For example, Krizhevsky et al. [5] trained a CNN to recognize images based on their visual features. This approach can be combined

 ISBN: 978-1-68558-048-3

with other methods, such as using camera metadata or manual annotations, to generate more accurate and diverse labels for images. Lee et al. [6] proposed a scalable method for image annotation that combines manual and automatic approaches to improve the accuracy and scalability of the labeling process.

In image retrieval, search algorithms are applied to the features extracted from images to find relevant images based on a user's query. The choice of search algorithm depends on the type of labels associated with the images and the problem at hand. For instance, if the labels are GPS coordinates, a distance-based search algorithm can be used to find images related to the query location, as demonstrated by Zhang et al. [7].

On the other hand, if the labels are in text form, a text-based search algorithm can be employed to retrieve images based on textual queries. There are various techniques available for text-based search, ranging from substring matching to lemmatization-based search, as shown by Balakrishnan et al. [8], or even using word embeddings, as explored by Günther [9] and Kenter et al. [10].

In practice, combining multiple features and approaches, such as GPS coordinates, text labels, image capture date, camera model, focal distance, etc. can yield more accurate and relevant results. Ismail [11] investigated image annotation and retrieval based on multi-modal feature clustering and found that this approach can significantly improve the retrieval performance of the system.

Therefore, in summary, image retrieval is a complex process that involves the extraction of features from images, the selection of suitable search algorithms based on the type of labels and the integration of multiple features and approaches to improve retrieval accuracy.

The methods mentioned earlier do not facilitate searching for images using a text query without first identifying the features. Moreover, they do not permit searching for a concept that is not included in the image labels, even if the concept exists within the image. These are the problems that OpenAI's CLIP [1] enables us to solve.

III. Method

With CLIP's text transformer, it is possible to convert a text query to a n-dimensional vector (where n differs depending on the CLIP model used). With CLIP's image transformer, it is possible to convert an image to a n-dimensional vector. Then, we can calculate the dot product (Equation 1) of the normalized query vector and each of the normalized image vectors. After this, we sort the images by the decreasing dot product and take first k images; this gets us the k images that match the query the most.

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \qquad (1)$$

While this approach works reasonably fast on a few images, it may not be scalable when dealing with a large number of images. This is particularly challenging if there is no access to GPUs to run the CLIP model. In order to address this scalability issue, the solution proposed in this paper suggests caching the image vectors. By caching the image vectors, repeated queries can be executed quickly, without having to wait for the image to be processed each time that a query is made.

Caching involves storing the computed image vectors on disk after the initial processing of images so that they can be accessed later. When a new query is received, the system retrieves the cached image vectors and computes only the query vector. This approach reduces the computational overhead associated with image querying and improves the response time for users. This is particularly useful when dealing with large image databases, where repeated queries are common. By leveraging caching, the system can handle repeated queries quickly and efficiently without having to recompute the image vectors each time.

The proposed solution also allows adding new images to the cache to avoid recomputing the whole cache when the image catalog is updated.

The main advantage of the proposed method over the methods described in Section 2 is that method does not require any metadata, labels, or annotations, which enables using it with any image catalog. Moreover, if used on labeled images, the proposed method allows searching for concepts not included in the image labels. The presence of these mechanisms enables users to utilize CLIP effectively in real-world scenarios involving image search.

IV. Implementation

The method described above is implemented in the Python utility called `rclip` [12]. `rclip` provides an easy-to-use CLI interface that allows users to search images within any directory on a computer where `rclip` is installed. Search within nested directories is also supported. To use it, the user should open the terminal, navigate to the directory containing the files that they want to search through, type `rclip <search query>`, and hit "Enter."

The solution uses OpenAI's `clip` library [1] to load the model and compute the feature vectors. `rclip` uses `ViT-B/32` version of the CLIP model. The code of the feature computing methods is shown in Figure 1 and Figure 2. The vectors produced by these methods are then used to compute the dot product (Equation 1).

`rclip` features the image vector cache implemented using `SQLite 3` RDBMS [13]. The image vector is computed and added to the index only if the cache does not already contain an entry for a given image. The vectors are cached by image paths. This allows the user to execute repeated queries over the same image catalog without waiting for the image vectors to be recomputed. Figure 3 defines the structure of the table storing the cached image vectors.

The implementation introduces a cache layer of a design described above, a module to manage (add, remove, and update) entries in the cache, a module to perform image and text vector computations using the CLIP model, and a module

 ISBN: 978-1-68558-048-3

```
def compute_image_features(self, images):
  images_preprocessed = (torch
    .stack([self._preprocess(thumb) for
        thumb in images])
    .to(self._device))

  with torch.no_grad():
    image_features = self._model.
        encode_image(images_preprocessed)
    image_features /= image_features.norm
        (dim=-1, keepdim=True)

  return image_features.cpu().numpy()
```

Figure 1. Image vector computing method

```
def compute_text_features(self, text):
  with torch.no_grad():
    text_encoded = self._model.
        encode_text(clip.tokenize(text).to
        (self._device))
    text_encoded /= text_encoded.norm(dim
        =-1, keepdim=True)

  return text_encoded.cpu().numpy()
```

Figure 2. Text vector computing method

to provide the user with a convenient command-line interface to perform image search.

When the user executes the `rclip` command, the tool first recursively processes the current directory and all its subdirectories, finds all of the images in them, and, if the image is not already present in the cache, computes their feature vectors and saves them to the SQLite cache database. This step is quick on repeated queries because the images are already cached, but if the user wants to skip the process of checking that the cache is up-to-date, they can pass the `-n` argument to `rclip` to skip the indexing step completely and speed up the `rclip` command execution even more.

Then, `rclip` computes the query vector, fetches from the cache database image vectors for all of the images located in the current directory, computes the similarity score between the query vector and each of the image vectors, sorts the scores by the decreasing order, and gets the first k images with the highest similarity scores.

Finally, `rclip` prints the paths to the images that match the query to the terminal. Users can then open the images in their favorite image viewer or editor.

The `rclip` source code is published on GitHub under the MIT license [12].

```
self._con.execute('''
  CREATE TABLE IF NOT EXISTS images (
    id INTEGER PRIMARY KEY,
    deleted BOOLEAN,
    filepath TEXT NOT NULL UNIQUE,
    modified_at DATETIME NOT NULL,
    size INTEGER NOT NULL,
    vector BLOB NOT NULL
  )
''')
```

Figure 3. Structure of the "images" table storing the cached image vectors

V. Performance

`rclip` was benchmarked using two different CLIP models, `ViT-B/32` (smaller CLIP model) and `ViT-L/14@336px` (larger CLIP model), on a NAS running `Intel(R) Celeron(R) CPU J3455 @ 1.50GHz`. Table I shows how `rclip` performs when indexing and searching through 269 photos when running on this CPU.

As Table I shows, the `ViT-L/14@336px` performance will not scale well, which makes `rclip` unusable in practical scenarios when running CLIP on low-level and mid-level consumer CPUs. This is why `rclip` uses `ViT-B/32`.

Running `rclip` indexing with `ViT-B/32` on 72,769 photos on the same NAS powered by `Intel(R) Celeron(R) CPU J3455 @ 1.50GHz` took 23 hours. Performing a query over 72,769 photos takes 56 seconds.

To give a better understanding of how `rclip` performance scales, Table II shows how `rclip` performs when indexing and searching through 50k images and 1.28m images on the `Apple M1 Max` CPU. As the Table II shows, the indexing time scales linearly with the number of images when the search time increases only slightly even when going from searching through 50 thousand images to searching through 1.28 million images.

It should be noted that real-life `rclip` application possibilities go far beyond results demonstrated in these benchmarks

TABLE I
INDEXING AND SEARCH PERFORMANCE ON `Intel(R) Celeron(R) CPU J3455 @ 1.50GHz` USING DIFFERENT CLIP MODELS

Model	Indexing Time	Search Time
`ViT-B/32`	3m56.626s	0m18.064s
`ViT-L/14@336px`	125m0.507s	3m19.742s
Difference	x31.70	x11.06

TABLE II
INDEXING AND SEARCH PERFORMANCE ON `Apple M1 Max` CPU USING `ViT-B/32`

Dataset	# of images	Indexing Time	Search Time
`ImagNet-1k` validation set	50k	19m24.750s	0m4.04s
`ImagNet-1k` train set	1.28m	8h31m26.680s	0m11.49s
Difference	x25.62	x26.35	x2.84

because `rclip` can be applied be used to execute any queries and not just the ones present in the dataset labels.

VI. Search Quality

As Table III shows, `rclip` achieves 31.17% top-1 accuracy and 44.80% top-5 accuracy rate on the `ImageNet-1k` [14] 1.28 million images train set and 55.15% top-1 and 81.34% top-5 accuracy on the `CIFAR-100` [15] 10 thousand images test set.

To get a better understanding of `rclip`'s performance, see Figure 5, Figure 6, and Figure 7 showing search results for a search performed on an unlabeled demo set of 142 images. Figure 4 gives a glimpse into the set and shows that there is a variety of different images present there.

VII. Conclusion

The development of the command-line tool, `rclip`, which employs OpenAI's CLIP model, has resulted in an efficient and user-friendly utility for image search. However, there are still possibilities for further optimizing its performance. Future plans include:

- to create a separate model for the CLIP text transformer and to load only it when users initiate a search that does not require indexing, thereby avoiding the loading of the CLIP vision transformer;
- to push the tool's scaling limits even further by introducing the sharded cache;
- to improve the tool's performance by preventing it from re-indexing files when they are renamed;
- to enable `rclip` write tags to the image file metadata for third-party software to utilize them;
- to enrich `rclip`'s search capabilities by utilizing metadata, which already exists within the images, like GPS coordinates, image capture date, camera model, tags, etc.;
- to do an in-depth comparison of `rclip` with other existing image search tools;
- to explore how well CLIP handles distorted and corrupted images.

Even with its current performance and capabilities, `rclip` is an incredibly valuable tool.

In summary, this paper introduces a practical and scalable method of searching images using natural language queries based on the CLIP model. The approach has demonstrated impressive results on the ImageNet-1k and CIFAR-100 datasets, indicating its potential applicability to a wide range of industries reliant on visual data.

TABLE III
RCLIP SEARCH QUALITY

Model	Top-1 accuracy	Top-5 accuracy
`ImageNet-1k` 1.28m	31.17%	44.80%
`CIFAR-100` 10k	55.15%	81.34%

References

[1] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," arXiv, 2021. [Online]. Available: https://arxiv.org/abs/2103.00020.

[2] N. Fiedler, M. Bestmann, and N. Hendrich, "Imagetagger: An open source online platform for collaborative image labeling," in *RoboCup 2018: Robot World Cup XXII 22*, Springer, 2019, pp. 162–169.

[3] J. Tesic, "Metadata practices for consumer photos," *IEEE MultiMedia*, vol. 12, no. 3, pp. 86–92, 2005.

[4] J. Kim, S. Lee, J.-S. Won, and Y.-S. Moon, "Photo cube: An automatic management and search for photos using mobile smartphones," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, IEEE, 2011, pp. 1228–1234.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[6] B. N. Lee, W.-Y. Chen, and E. Y. Chang, "A scalable service for photo annotation, sharing, and search," in *Proceedings of the 14th ACM international conference on Multimedia*, 2006, pp. 699–702.

[7] J. Zhang, A. Hallquist, E. Liang, and A. Zakhor, "Location-based image retrieval for urban environments," in *2011 18th IEEE International Conference on Image Processing*, IEEE, 2011, pp. 3677–3680.

[8] V. Balakrishnan and E. Lloyd-Yemoh, "Stemming and lemmatization: A comparison of retrieval performances," *Lecture Notes on Software Engineering*, vol. 2, no. 3, pp. 262–267, 2014.

[9] M. Günther, "Freddy: Fast word embeddings in database systems," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1817–1819.

[10] T. Kenter and M. De Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 1411–1420.

[11] M. M. B. Ismail, *Image annotation and retrieval based on multi-modal feature clustering and similarity propagation*. University of Louisville, 2011.

[12] Y. Mikhalevich, *rclip*, version 1.2.5, Jan. 2023. [Online]. Available: https://github.com/yurijmikhalevich/rclip.

[13] R. D. Hipp, *SQLite*, version 3.31.1, 2020. [Online]. Available: https://www.sqlite.org/index.html.

[14] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[15] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," University of Toronto, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

 ISBN: 978-1-68558-048-3

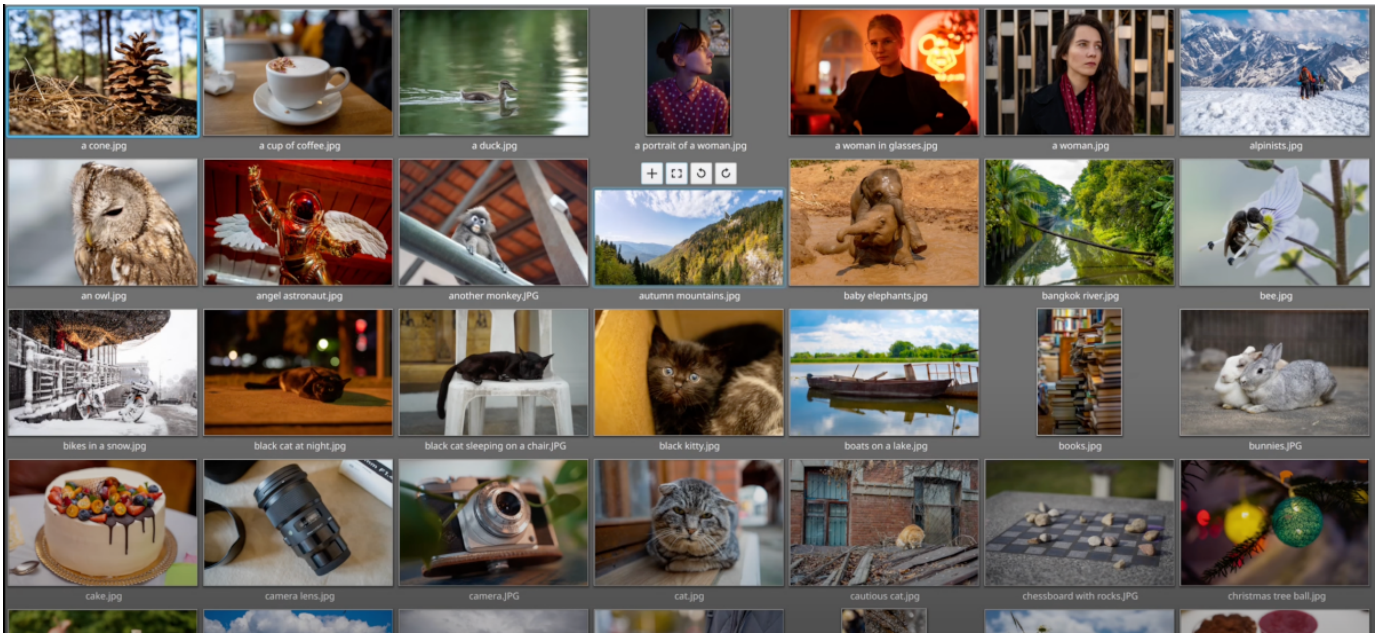

Figure 4. Demo set sample

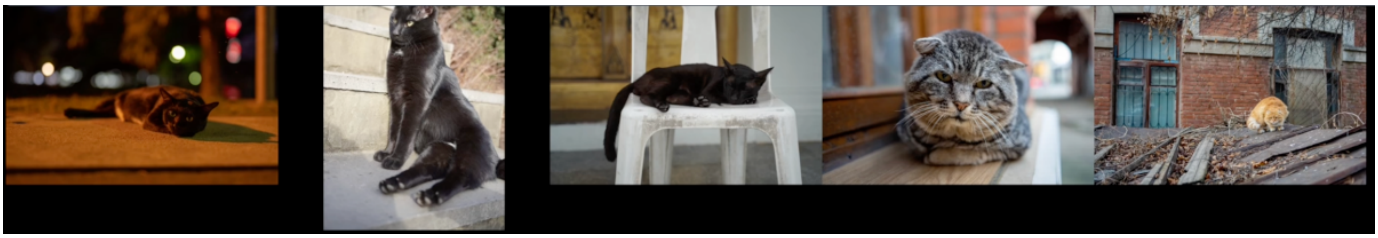

Figure 5. Search result for query “cat”

Figure 6. Search result for query “leading lines”

Figure 7. Top result for query “a kitten peeking from behind a corner”

 ISBN: 978-1-68558-048-3