SeeSaw: Interactive Ad-hoc Search Over Image Databases

OSCAR MOLL, MIT CSAIL, USA
MANUEL FAVELA, MIT, USA
SAMUEL MADDEN, MIT CSAIL, USA
VIJAY GADEPALLY, MIT Lincoln Laboratory, USA
MICHAEL CAFARELLA, MIT CSAIL, USA

As image datasets become ubiquitous, the problem of ad-hoc searches over image data is increasingly important. Many high-level data tasks in machine learning, such as constructing datasets for training and testing object detectors, imply finding ad-hoc objects or scenes within large image datasets as a key sub-problem. New foundational visual-semantic embeddings trained on massive web datasets such as Contrastive Language-Image Pre-Training (CLIP) can help users start searches on their own data, but we find there is a long tail of queries where these models fall short in practice. SeeSaw is a system for interactive ad-hoc searches on image datasets that integrates state-of-the-art embeddings like CLIP with user feedback in the form of box annotations to help users quickly locate images of interest in their data even in the long tail of harder queries. One key challenge for SeeSaw is that, in practice, many sensible approaches to incorporating feedback into future results, including state-of-the-art active-learning algorithms, can worsen results compared to introducing no feedback, partly due to CLIP's high-average performance. Therefore, SeeSaw includes several algorithms that empirically result in larger and also more consistent improvements. We compare SeeSaw's accuracy to both using CLIP alone and to a state-of-the-art active-learning baseline and find SeeSaw consistently helps improve results for users across four datasets and more than a thousand queries. SeeSaw increases Average Precision (AP) on search tasks by an average of .08 on a wide benchmark (from a base of .72), and by a .27 on a subset of more difficult queries where CLIP alone performs poorly.

CCS Concepts: • **Information systems** → *Image search.*

Additional Key Words and Phrases: image retrieval, image search, machine learning, multi-modal embeddings, relevance feedback, active search, data exploration

ACM Reference Format:
Oscar Moll, Manuel Favela, Samuel Madden, Vijay Gadepally, and Michael Cafarella. 2023. SeeSaw: Interactive Ad-hoc Search Over Image Databases. *Proc. ACM Manag. Data* 1, 4 (SIGMOD), Article 260 (December 2023), 26 pages. https://doi.org/10.1145/3626754

1 INTRODUCTION

Increasingly inexpensive cameras and storage make it ever easier to collect image data. Large quantities of video and images are now captured from dedicated cameras as well as mobile phones, vehicles, and drones. Nevertheless, the ability of an engineer or team to explore their own image data and discover ad-hoc items of interest lags far behind their ability to collect that data.

For example, an engineer at an autonomous vehicle company with a large repository of data may wish to find examples of people in wheelchairs to extend an object detector or to find examples of bikes in the snow to test an existing detector. Or, an ornithology researcher may want to search

Authors' addresses: Oscar Moll, orm@csail.mit.edu, MIT CSAIL, Cambridge, MA, USA; Manuel Favela, mfavela@alum.mit.edu, MIT, Cambridge, MA, USA; Samuel Madden, madden@csail.mit.edu, MIT CSAIL, USA; Vijay Gadepally, vijayg@ll.mit.edu, MIT Lincoln Laboratory, USA; Michael Cafarella, michjc@csail.mit.edu, MIT CSAIL, USA.

2836-6573/2023/12-ART260
https://doi.org/10.1145/3626754

their camera archives to know which of all their camera locations seem to show more sightings of a particular type of bird [20, 46, 50].

Whether the goal is to extend the capabilities of an object detector model, to enhance test cases for existing models, or simply to explore a large trove of image data, finding relevant images within a dataset of images or video is a key problem for many users. For example, depending on the dataset, wheelchairs or bikes in the snow may be rare, appearing in only one in a thousand images or less. Hence, ad-hoc searches can be challenging without efficient image search tools.

A well-known high-level approach to the search problem is representing the contents of the database as well as the queries as vectors: a text query becomes a *query vector* q, and an image in the database becomes an *image vector* x, and the relevance of an image x to a query q is estimated by its inner product $q \cdot x$, which measures the geometric *alignment* of the query and the database element (every vector is unit normed). The most relevant results for a query vector q are the solution to $\text{argmax}_x\, x \cdot q$, the elements with a maximum inner product with q. One advantage of this modeling approach is the availability of scalable and low-latency vector stores. These stores create an index of all vectors x ahead of time and then solve the search at query time without a full scan of the database of x, enabling interactive searches for any query.

This vector paradigm relies on mapping queries and images to vectors, and the deep learning-based method to achieve the goal of mapping images and text to vectors today is through *cross-modal* or *visual-semantic* embeddings [7, 14, 15, 22, 38]. The accuracy of this approach depends on the quality of the cross-modal embedding used to capture the important concepts in an image.

State-of-the-art cross-modal embedding models consist of large neural networks trained on massive datasets of corresponding image-text pairs. A notable example is CLIP [37], which is a model pre-trained on a crawl of hundreds of millions of web images and their `alt-text` attributes as a low cost proxy for a caption.

Zero-shot CLIP. With CLIP, users can often cold start many searches using text alone in new datasets without fine-tuning any model, an approach called *zero-shot* learning. Zero-shot CLIP is surprisingly accurate even when used on datasets it was not trained on, including many of our evaluation datasets.

Limitations of existing approaches. In the case of CLIP, despite the high average zero-shot quality of the model, the quality of the results on a specific dataset varies substantially between queries. For example, using CLIP embeddings on the Berkeley Deep Drive (BDD) dataset [51] of street scenes, we can easily find scenes with bicycles. For wheelchairs, however, which only occur in a handful of scenes, using CLIP alone requires looking through more than 100 images before the first wheelchair is found. Figure 1 shows the accuracy distribution in queries from four evaluation datasets, measured as AP, a common accuracy metric. The step on the right edge for each dataset corresponds to a substantial number of queries with optimal results ($AP = 1$). The trailing slope on the left shows a long tail of queries with lower accuracy. The annotations on the dashed line quantify the fraction and amount of queries with $AP < .5$ using zero-shot CLIP, which is large in some of the datasets. We note that the "queries" used in these plots correspond only to labeled categories in these datasets. From a user's point of view, the plotted distribution is not as relevant as the distribution seen on their queries of interest; i.e., high accuracy on bicycles does not compensate for poor results on wheelchair if that is the query of interest.

Query alignment. CLIP sometimes provides low-relevance results for two reasons. First, the CLIP embedding of the string (e.g., "wheelchair") may not be close enough to the relevant image vector embeddings; we call this a *query alignment* deficit. Figure 2a shows a visual representation of lack of query alignment: all the image vectors, represented as circles arranged along an imaginary

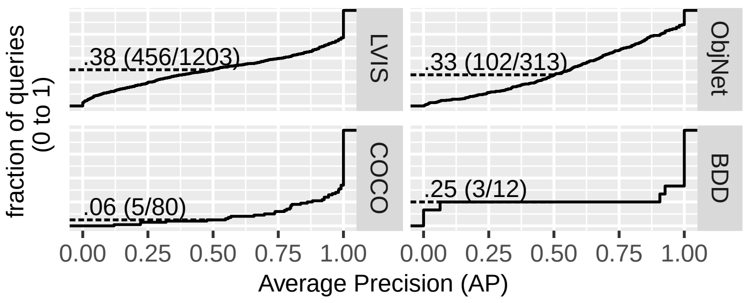

Fig. 1. The solid line plots the cumulative distribution function of Zero-shot CLIP Average Precision (AP) across four evaluation datasets introduced later. The horizontal dashed line marks the fraction and absolute number of queries with $AP < .5$, for each dataset.

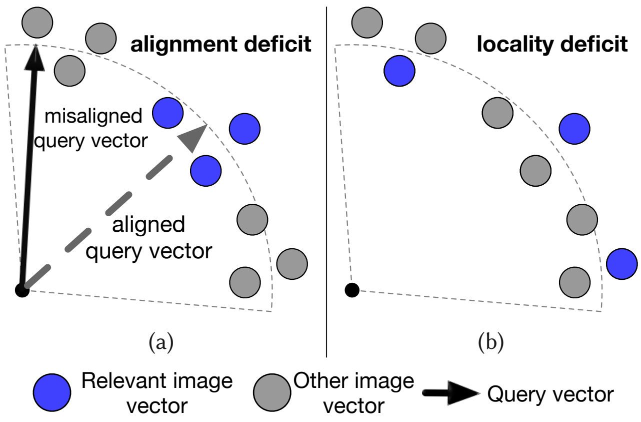

Fig. 2. Two causes of suboptimal search results: alignment deficit (left), and locality deficit (right). Both circles and arrows are unit norm, lying on the unit sphere (dashed circle).

circular arc (due to image vectors being normalized unit size). The blue colored circles represent vector embeddings of the relevant images (or, as we introduce later, relevant patches within images), in this case images of wheelchairs in BDD. The initial query maps, via the string embedding, to the vector represented by the solid arrow. In the diagram, this initial query vector aligns with the gray circles at the top more than with the blue circles to the right. Therefore, these non-relevant results would appear first in a search, ahead of the relevant results, causing difficulties to users.

Concept locality. A second possible problem causing poor results is that embeddings for images of interest (e.g., wheelchairs) may not be clustered tightly together in the database, as shown in (Figure 2b). In this diagram, we see that no single query arrow would be able to align well with all three blue image vectors because they are diffused among non-relevant gray vectors. Regardless of what vector a user may conjure up, the results will never be wholly relevant. We call this situation a *concept locality* deficit. Queries often present both types of deficits, so they benefit from improving either.

Our solution: SeeSaw. SeeSaw's goal is to allow users to search their data leveraging embeddings such as CLIP and helping users improve their results when needed. Users work in a loop with SeeSaw providing feedback in the form of boxes around relevant regions of images. This process results in better-aligned query vectors, such as the dashed line vector in Figure 2a, thereby improving results. A user interacts with SeeSaw following the pseudo-code of Listing 1: a search starts with a text string, which is converted the text into a vector value q_0 using an embedding model like CLIP (line 2). q_0 is used as the query vector for a lookup operation into the vector store (line 2), which locates the most relevant vector x in the store for the query vector, i.e., the one with the largest inner product. The corresponding image is presented to the user, who provides feedback. The query vector is now updated to value q_1 via `query_align` in line 7, which includes previous feedback. In the next round, q is updated to value q_2, and so on. Ideally, results improve on every round of feedback. In reality, each loop consists of a batch of a user specified size.

```
  input  : text_query
1 feedback_map ← {}
2 query_vector ← CLIP.embed_string(text_query)
3 while True do
4     img_id ← vector_store.lookup(query_vector)
5     img_feedback ← UI.show(img_id)
6     feedback_map.update(img_id, img_feedback)
7     query_vector ← query_align(feedback_map)
8 end
```

Listing 1: high-level structure of search with user feedback. SeeSaw focuses on the logic of `query_align`

Related work. The general idea of leveraging user feedback to locate results is known as *relevance feedback* in information retrieval [24, 41],[32, Ch.9], and as *active search* within active learning [16]. However, SeeSaw must address two challenges not present in prior work: first, we find that basic implementations of `query_align` as well as state-of-the-art active search [23] can decrease result relevance when the starting point is zero-shot CLIP, even when all approaches start with the same high-quality CLIP embeddings. Second, practical approaches also need to provide interactive latency for large datasets, meaning the computational work needed on every round should grow sub-linearly with the dataset size, which is not true of state-of-the-art active search approaches.

SeeSaw insights. SeeSaw addresses both challenges based on three main insights: The first insight is that we should merge user labels with the original CLIP query rather than relying on either alone. SeeSaw accomplishes through its implementation of `query_align` of line 7, within which it searches for a query vector minimizing a custom loss function. This loss function goes beyond reflecting accuracy on the observed feedback as a standard supervised-learning approach would: SeeSaw additionally encourages similarity between the aligned query and the original embedding of the concept being searched through a novel regularization term, ensuring the CLIP text query is used for both the initial search and also within the minimization problem. We refer to this idea as CLIP alignment.

The second insight is that the process to improve the query should reflect the data distribution of the entire database rather than only that of user feedback: since user feedback data in SeeSaw up to iteration t is based on those vectors in X_D nearest to q_0, q_1,... and q_{t-1}, the observed data is skewed toward this very specific region of the database. One hypothetical way to address this problem is sampling randomly from the dataset and labeling these nodes, but this approach imposes labeling requirements on the user, and for many hard-to-find objects this approach would find no positive examples. SeeSaw cheaply approximates this hypothetical process by adding a second database-dependent regularization term to the loss function. We show this regularization term is conceptually equivalent to synthesizing a new training set where elements are instead picked randomly and uniformly from the vector database, and their unobserved labels are approximated through label-propagation[54]. We refer to this as database alignment or DB alignment.

SeeSaw models both CLIP alignment and DB alignment into a loss function that can be quickly minimized so that SeeSaw can produce a better-aligned query vector with little input from the user and with low latency, and so that SeeSaw can leverage fast vector stores for search. Moreover, the amount of work SeeSaw does at query time within the loop in Listing 1 grows only with the size of the observed dataset, unlike active learning approaches we evaluate that rely on linear time scoring of the full database after each feedback iteration.

In addition to the above techniques to improve query alignment, SeeSaw employs a third insight: a multi-vector, multi-scale representation of images derived from separately embedding patches of different sizes and positions within a single image. This representation is motivated by observing that in complex images the object of interest may not be the most prominent feature in an image; this is a common cause of accuracy problems for zero-shot CLIP. This technique is conceptually simple and orthogonal to CLIP and DB alignment, but in practice, because this representation multiplies the number of vectors in the database by an order of magnitude, only search techniques whose latency does not depend linearly on the vector database size can be integrated easily with it.

Contributions. In summary, our contributions are:

(1) We introduce SeeSaw's custom *query alignment* algorithms for user-in-the-loop image search: CLIP alignment combined with database alignment, which provide high quality results with a fixed and limited amount of user feedback while avoiding any linear time computational costs at query time that could hinder interactivity on larger datasets.
(2) We combine the query alignment algorithms with a multi-scale feature representation for images that is possible due to the scalability of the alignment algorithms.
(3) We demonstrate with extensive benchmarks across 4 datasets and hundreds of queries, that SeeSaw consistently improves retrieval metrics; overall SeeSaw improves Average Precision (AP) from .19 to .46 on a subset of more challenging queries.
(4) We demonstrate that alternative techniques to implementing relevance feedback can often either reduce search accuracy with respect to the zero-shot CLIP approach, or scale poorly with data, or both.

2 SYSTEM

SeeSaw[1] consists of the following components: 1) a graphical user interface, 2) a pre-trained visual semantic embedding model (CLIP [38]), 3) an indexed vector store for max inner product queries (Annoy [5]), and 4) a server layer, which we will call the query aligner, mediating between the other components and implementing the query alignment described above. Figure 3 shows how data moves between these components.

2.1 CLIP

SeeSaw uses the CLIP [37] pre-trained visual-semantic embedding model for preprocessing and during querying. SeeSaw embeds images into vectors using the visual component of CLIP during preprocessing. During querying, SeeSaw uses the string embedding component of CLIP to translate string inputs from the user into query vectors.

2.2 Vector Store

After raw image data is processed into vectors, and is indexed by the vector store, the vector store provides a maximum inner product lookup interface with low latency, which is important because

[1]The source code this work is available at https://github.com/orm011/seesaw

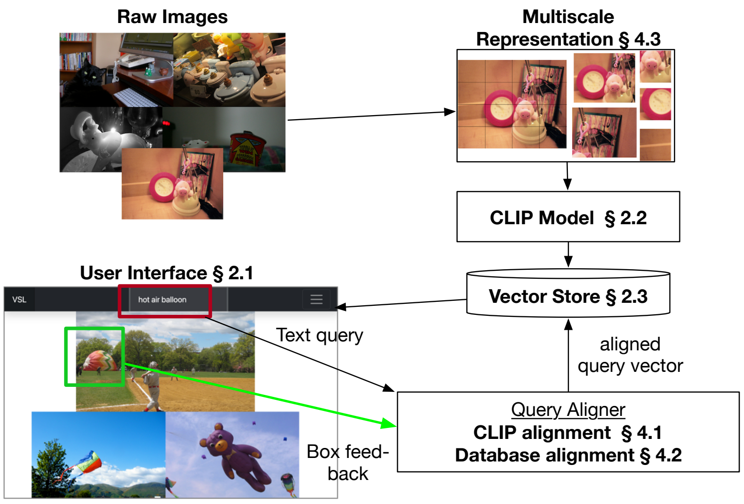

Fig. 3. SeeSaw component diagram. Top: preprocessing steps; Bottom: dataflow during interaction loop.

the user waits on results from the system. The vector store needs to be accurate, but does not need to be exact: it is acceptable for the result in line 4 of Listing 1 to be among the top largest rather than exactly the largest, as even if the exact result were returned, there is already error inherent to the embedding representation as shown in Figure 4

Our implementation uses the Annoy store[5], which offers only *approximate* maximum innner product lookup, which is also what most vector stores offer. We saw only a minor drop in accuracy metrics in our benchmarks using Annoy vs an exact but slow scan.

2.3 User Interface (UI) and Querying

Figure 3, bottom left, shows a screenshot of the SeeSaw UI for the hot air balloon query as one of the components of SeeSaw. A user wishing to make a model to detect hot-air balloons can begin the search process with SeeSaw through text by typing "hot-air balloon" into the search box. The loop of Listing 1 runs, and through the UI, the user provides feedback on the results offered so far (line 5). This flow of data is diagrammed in Figure 3.

2.4 Preprocessing

Before using SeeSaw, we perform a one-time pre-processing pass over the image data. Pre-processing in SeeSaw consists of converting raw image data into semantic feature vectors using a pre-trained visual embedding (CLIP, in our case). For SeeSaw, the runtime of this preprocessing pipeline depends on four variables: the number of images in the dataset, the pixel sizes of the images in the dataset, the inference cost of the embedding, and the number and type of Graphics Processing Units (GPUs) available. On COCO, a dataset of 120000 images, SeeSaw preprocessing in our un-optimized single GPU pipeline takes less than an hour. Because this task is data parallel, the runtime can be reduced to minutes by using more GPUs. Furthermore, model optimization techniques just as JIT compilation would further reduce this runtime for real applications with larger datasets.

The vector store, Annoy, takes less than 20 minutes to build the index over the vectors computed above. These costs are incurred once per dataset and are then amortized across all subsequent queries.

3 APPROACH

The key idea of our approach is to improve query alignment by leveraging user feedback and by enriching that feedback with two other sources of information: the CLIP embedding of the query itself (§4.1) as well as the structured of the unlabeled database (§4.2). SeeSaw incorporates these different sources of information within a single loss function, which is minimized with respect to an internal query vector parameter on every round to yield the next query vector SeeSaw will use internally. A secondary aspect of our approach is using a multi-vector, multi-scale representation of the data which we cover in §4.3.

3.1 Motivation for Query Alignment

In the introduction we explained how both *query alignment* and *concept locality* are important for searches to work well (diagrammed in Figure 2). SeeSaw focuses on query alignment, so it is valuable to quantify the potential gains and the limitations of this approach. Because our evaluation datasets have complete labels for different categories, which we will use as evaluation queries, we can understand how far from the ideal any approach is. In this section, we show a large fraction of labeled categories in the ObjectNet dataset presents a combination of high concept locality, and a lot of the error is due to lower query alignment, and therefore adjusting alignment alone can improve results substantially.

Ideal query vector. For a query such as "wheelchair" within ObjectNet, we can measure query alignment deficit by comparing the accuracy of using the embedding of the string "a wheelchair" to the accuracy of an *ideal query vector* to find wheelchairs within the database, one derived with full knowledge of ObjectNet images and their ground-truth labels. We can compute an ideal query vector by fitting a linear classifier model on the CLIP image embedding vectors X, where each embedding is labeled with $y = 1$ or 0 depending on whether or not the image is labeled as having a wheelchair in it. This linear model is certainly over-fit from a prediction perspective; but, in this case, model fitting is a simple and efficient search method to find out whether there are *any* high-accuracy query vectors.

Using the labels for "wheelchair" we can then compute the accuracy of results when using the string-derived query vector or the ideal query vector. We will see shortly that there are queries where even this best-fit vector has low accuracy: a strong indication of low concept locality; and conversely, a common case is when the best-fit vector has high or perfect accuracy, but the string-derived vector has low accuracy, a strong indication of high locality for the concept but low alignment for the initial query, the kind of scenario where SeeSaw would work best.

After carrying out the above process for 300 different queries, we measure accuracy for each query using Average Precision scores, and plot the AP of the ideal and initial queries for all 300 ObjectNet labeled categories as the y and x coordinates of scatter-points in Figure 4.

Average Precision (AP). AP is an accuracy metric common for information retrieval [31], because it rewards earlier relevant results more heavily than simpler metrics such as precision or F-score, and without picking an arbitrary result cutoff. AP values range from 0 to 1. An AP of 1 means perfect accuracy: when all positive results appear before the negative search results.

The figure shows the median AP for the ideal queries (see vertical boxplot) is above .9, and more than 25% reach 1, while the median AP of initial queries (see horizontal boxplot) is around .2, which shows that concept locality in the embedding is high because the ideal vectors perform much better than the string derived query vectors. Note several best-fit queries score 1, indicating those queries have very high locality and improving alignment is all that is needed. This is not always the case, but even queries with low y coordinate values where locality may be a problem show benefits from improving alignment (hence lie comfortably above the diagonal dashed line).

Note that these numbers do not mean we can easily find ideal vectors in practice, given the lack of labeled data and the very few samples available from feedback to the system, but they show that focusing on alignment makes sense for CLIP embeddings of this dataset.

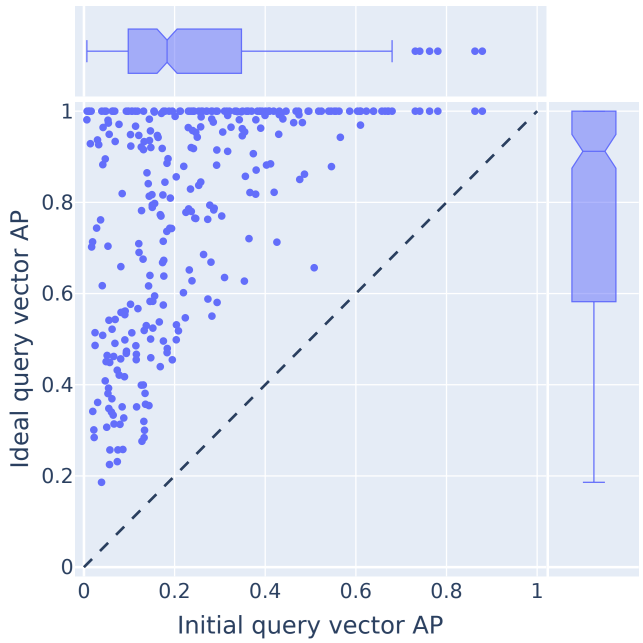

Fig. 4. Comparing the Average Precision (AP) of the ideal query vector vs the initial CLIP string embedding for each labeled class on the ObjectNet dataset. Each point in the scatter plot represents one of the 300 categories in ObjectNet. The median AP for the ideal queries (see vertical boxplot) is above .9, and more than 25% reach 1, while the median AP of initial queries (see horizontal boxplot) is around .2; showing that even though concept locality in the embedding is high, the initial query alignment can be relatively poor. Section §3.1 explains the setup in more detail.

3.2 Motivation for CLIP Alignment Approach

One natural way to implement query alignment in the context of a text search is to use a CLIP string vector q_0 to locate a few examples that we ask the user to label. This results in a set of examples (x, y) which we can use to learn a new query vector as part of `query_align` in line 7 in Listing 1. In the simplest approach, we can simply train a standard logistic regression model on the user's labels for results seen so far. After round t of feedback from the user, we pick q_{t+1} as follows:

$$q_{t+1} = \underset{w}{\operatorname{argmin}}\, \mathcal{L}_1(\mathbf{w})$$

$$\mathcal{L}_1(\mathbf{w}) = \sum_{i=1}^{t} \operatorname{LogLoss}\left(y_i, \operatorname{sigmoid}(\mathbf{w}^\top \mathbf{x}_i + b)\right) + \lambda |\mathbf{w}|^2 \tag{1}$$

Few-shot CLIP. The summation term is the logistic loss function added over elements with user feedback, the weight vector $\mathbf{w}$ is learned, as is the scalar bias b, typical in standard logistic regression and necessary to achieve well-calibrated output probabilities. In practice we find fitting both $\mathbf{w}$ and b as opposed to forcing b to be 0 substantially reduces the accuracy of the learned $\mathbf{w}$ as a query, so we do not use the b parameter.

The extra term, $\lambda|\mathbf{w}|^2$, where λ is a scalar hyperparameter, penalizes large magnitudes in the weight vector and is again a standard penalty applied to the loss function of logistic regression to prevent selecting values of $\mathbf{w}$ that are very large when the data are fully separable [33]. In the interactive setting we work in, with very few labeled examples, it is essentially guaranteed that the labeled data will be separable because the number of labeled data items is small compared to the dimension (512) of the CLIP embedding, so this penalty is always necessary.

This approach based on Equation 1 is called the *few-shot* CLIP approach, as opposed to the *zero-shot* CLIP approach of using q_0 derived from a string with no feedback. Few-shot CLIP has some advantages over zero-shot CLIP because the learned query vector q_{t+1} is now based on actual vectors from the database. Moreover, CLIP embeddings of images show high average few-shot learning accuracy on many datasets: a handful of examples are often enough to train a linear model with high accuracy [38]. However, we find the few-shot approach using Equation 1 in the way we described is less accurate than the zero-shot CLIP approach, and the accuracy drop is evident empirically on all our datasets, as we will explain and show later in Table 2. The few-shot approach is a baseline in its own right, and we evaluate it together with other baselines in §5.4.

There are several reasons for the drop in accuracy of the few-shot CLIP approach from the zero-shot CLIP approach: First, the absolute accuracy of zero-shot CLIP can be high, hence no method can improve substantially on it. Second, q_1 in the few-shot approach is computed from very few vectors from the database, depending on the batch size, unlike the ideal vectors computed in Figure 4 which beat the zero-shot CLIP vector but are trained on thousands of samples. In machine learning algorithms small samples lead to larger generalization error. Third, the vectors X_t, y_t from relevance feedback up until round t are not necessarily representative either of the region of the vector space that contains relevant results, nor of the full database, where the learned q_{t+1} will be used as a query.

We note that on some occasions, when the initial embedding query q_0 is of sufficiently poor quality, the few-shot CLIP approach does improve results beyond what the zero-shot approach offers. However, even in that case, the approaches we introduce offer strong advantages over either approach alone, as we show in our evaluation.

4 DETAILED APPROACH

SeeSaw leverages the previous high-level insights into multiple techniques which we explain in detail in this section. First, we integrate the zero-shot and few-shot approaches into a single combined approach. We call this approach *CLIP alignment*. Second, we guide our query vector improvement process to also account for the structure of the unlabeled vectors in the database, which we call *Database alignment*. Finally, we add a multiscale image representation, where we extract multiple vectors for different patches of the image at preprocessing time to allow us to capture objects that appear in different scales and positions in images.

4.1 CLIP Alignment

Consider a search scenario where we first find a single positive example x_0 and then a single negative example x_1, in that order, using the CLIP string embedding as the initial query vector q_0. There are many possible query vectors that produce the same observed ordering of x_0, x_1: for example, clearly q_0 produces this order, as does x_0 used as a query itself, which is guaranteed to score itself as the maximum possible score of 1. A third possible query vector is found by minimizing a loss function such as logistic loss (similar to Equation 1) which would align somewhat with the positive example and away from the negative example. There are many more possible solutions that produce this same observed ranking.

However, all these solutions will produce potentially different search results in the next round, and these will have different qualities. How should we select the query vector? A typical approach is cross-validation, where data is repeatedly separated into test and training sets to learn a model that generalizes well on unseen data, but in this context with a handful of points, cross-validation is not feasible.

Instead, we rely on a rule-of-thumb based on the *principle of stability* for machine learning algorithms [43, p. 174]. The principle states that when choosing between two methods that both fit the data equally well, the method more likely to generalize is the one that changes least when we include or exclude a data point from the training set. Intuitively, a method that overly relies on any one data point x is also most susceptible to generalization errors due to sample variance. In our setting, the original query vector q_0 of zero-shot CLIP is not influenced by the observed data at all, so there are reasons to prefer that vector as our next query by default.

CLIP alignment loss term. In reality, however, our initial query vector q_0 will often fall short, as we showed in the x-axis of Figure 4, for example. In many cases we must strike a balance between being unduly influenced by sample noise and incorporating information from feedback. This observation is the basis of CLIP alignment, which we implement by adding an extra penalty term to the loss function in Equation 1:

$$\mathcal{L}_2(\mathbf{w}) = \mathcal{L}_1(\mathbf{w}) + \lambda_c \left(1. - \mathbf{w} \cdot q_0/|\mathbf{w}|\right) \tag{2}$$

The term $(\mathbf{w} \cdot q_0)/|\mathbf{w}|$ is the cosine distance between the parameter $\mathbf{w}$ and the initial query q_0 (which is normalized), encouraging the optimal $\mathbf{w}$ to geometrically align with the original CLIP text query q_0, in addition to minimizing the previous loss.

In other words, if two possible query vectors w_1 and w_2 have the same classification loss, the one with the highest cosine similarity to the original query q_0 will be favored by this loss function.

λ_c is a new hyper-parameter that governs the trade-off between fitting the feedback data and preserving the new weight's similarity to the original. A large λ_c parameter means we ignore the user labels and a small one means we ignore the initial text query.

As more user examples come in, the user input is weighted more highly with respect to the CLIP prior. We show in the evaluation section that the resulting query vectors from adding this additional loss term are more accurate than *either* the original q_0 (the zero-shot approach) or one learned purely from the data (the few-shot approach), as in Equation 1, and this is the case even when q_0 has a poor initial performance.

4.2 Database (DB) Alignment

From a supervised learning point of view, a query minimizing Equation 2 on a large sample is also likely to show a low error over the full database. However, in SeeSaw we minimize Equation 2 over a small sample x_t of examples previously shown to the user, for which we got feedback y_t. Besides their small size, these samples are not a random sample of the database X_D because they were elements with high similarity to previous query vectors $q_t, q_{t-1}..., q_0$. This is an instance of a *domain shift* problem, where the target domain distribution X_D differs from the training set distribution X_t, y_t, even if the mapping $X \rightarrow y$ being learned is the same [26].

SeeSaw uses this observation as a starting point to further improve query alignment with the real database. At a high level, our key observation is that we can approximate an unbiased and large sample of the data X_D, y_D using the label propagation algorithm defined in [54], which we will explain below, and then use this new training set consisting of X_D and $\hat{y}_D$ when solving for $\mathbf{w}$ in Equation 2.

Note that while the "propagated" labels $\hat{y}_D$ are only an estimate of the unobserved labels, the distribution of X_D faithfully reflects the distribution of the database vectors.

We found the propagation step improves the final classifier, but that the propagation algorithm can reduce interactivity, as it must run after every round of feedback to propagate the new labels and requires iterating over a full k-Nearest Neighbor graph (kNN graph) of the vectors in the database.

DB alignment loss term. Hence, the version of database alignment we use in SeeSaw takes the propagation approach only as a conceptual starting point but achieves the effect by adding a second alignment term to the loss function Equation 2, producing Equation 3.

In this updated loss function, the matrix M_D in the expression $\mathbf{w}^T M_D \mathbf{w}$, which we will explain in detail shortly, depends on the vectors in the database X_D, and on their kNN graph, but not on the query q_0, so it can be computed once per dataset ahead of time. M_D's size is 512×512, and is constant with respect to the size of the database: its size is only a function of the CLIP embedding dimension of 512, not of dataset size.

$$\mathcal{L}_3(\mathbf{w}) = \mathcal{L}_2(\mathbf{w}) + \lambda_D(\mathbf{w}^T M_D \mathbf{w})/|\mathbf{w}|^2 \tag{3}$$

In the remainder of this section, we explain how this regularization term relates to the original high-level idea of learning $\mathbf{w}$ from a larger sample and how M_D is computed. We do not know the labels, y_D, but we do know the true distribution of the vectors X_D, and thanks to user feedback we have a small set X_t, y_t of true labels. Label propagation [54, 55] is a semi-supervised learning algorithm that takes the above as inputs and generates a soft (non-binary) approximation $\hat{y}_D$ of y_D given X_D and y_t.

The high-level assumption of the propagation algorithm is that similar points in X_D should have similar values of y_D. Implicitly, this leads to high-density clusters in X_D having homogeneous values.

Operationally, label propagation requires using a k-nearest neighbor graph of elements in X_D, which we compute using an implementation of NN-descent[12], an approximate but scalable way to compute a kNN graph over large datasets. We can represent this kNN graph by its adjacency matrix W. w_{ij} in the adjacency matrix is a similarity score x_i and item x_j in the database. Following [54] we use $w_{ij} = \exp\left(-(X_i - X_j)^2/2\sigma^2\right)$ as our similarity metric i.e., we let the similarity metric decay exponentially with the distance between embedding vectors decreases. σ is a scalar hyper-parameter controlling how fast the similarity metric drops. The propagation algorithm in [54] minimizes the total differences between neighboring vertices in the kNN graph: $\sum_{i,j} w_{ij}(y_i - y_j)^2$. As explained in [54], this sum can be stated concisely as:

$$y^T(D - W)y \tag{4}$$

where W is the adjacency matrix, and D is a closely related "degree" matrix, a diagonal matrix where each diagonal entry is the sum of the corresponding row in W, both matrices are derived from computing a kNN graph for the vectors in X_D. Both D and W are square matrices of size $N \times N$ where N is the size of the database X_D. In practice, W is large but sparse because it only has k non-zero entries per row, one for each of the k neighbors

DB alignment approximation. We can obtain Equation 3 by observing the final $\mathbf{w}$ will be fitted to these synthetic labels $\hat{y}_D$ using logistic regression, which fits a sigmoid to the data. Because we empirically observe linear models fit CLIP embeddings well (Figure 4), i.e., concepts are clustered, then it is reasonable to assume sigmoid$(X_D\mathbf{w}) \sim y_D$ in practice, and therefore, we can replace y in

Equation 4 with $y(\mathbf{w}) \coloneqq \text{sigmoid}(X_D\mathbf{w})$, to obtain $y^T(\mathbf{w})(D-W)y(\mathbf{w})$. Instead of one minimization to find $\hat{y}$ and a separate one to find $\mathbf{w}$, we can now add a term $y^T(\mathbf{w})(D-W)y(\mathbf{w})$ to Equation 2 and perform a single minimization with respect to $\mathbf{w}$. The expression $D-W$ is an $N \times N$ sparse matrix, where at most $k \times N$ entries are non-zero, and where N is the size of the database. X_D is of size $N \times 512$, where 512 comes from CLIP, and the vector y_D is of size N. It is easy to see that this unified approach can be slow because it scales with N, the size of the dataset.

However, we can bypass this blow-up problem if we replace sigmoid($X_D\mathbf{w}$) with $X_D\mathbf{w}/|\mathbf{w}|$, yielding $\mathbf{w}^T X_D^T (D-W) X_D\mathbf{w}/|\mathbf{w}|^2$.

$\mathbf{w}^T X_D^T (D-W) X_D\mathbf{w}/|\mathbf{w}|^2$ can be interpreted on its own right as penalizing drastic variation of the cosine score in highly dense regions of the graph. Normalizing $\mathbf{w}$ is meant to avoid $\mathbf{w}$ being pulled to 0 in order to make the expression 0. Because the derivative of the cosine similarity between vectors is minimized when the value of the cosine is 1, this term points $\mathbf{w}$ toward the center of a dense region instead of its periphery when either direction explains the few labeled samples equally well.

We define $M_D = X_D^T (D-W) X_D$, grouping the matrix in the middle of the expression. M_D can be computed during dataset pre-processing by building a kNN graph, which gives us both $\mathbf{w}$ and D, as we do for propagation, and then computing the product $X_D^T (D-W) X_D$. This product is computed only once at preprocessing. If this preprocessing cost becomes prohibitive, we found that using a sample of a few thousand vectors from X_D, instead of the full X_D database, produces a very similar M_D (note that we did not enable this optimization in our experiments).

We note that it makes sense to ask: if we had access to these propagated labels $\hat{y}_D$, we could also use these propagated labels directly as a score, instead of now fitting a linear $\mathbf{w}$. However, fitting a $\mathbf{w}$ as we do above is not only a runtime optimization, it also increases accuracy. The $\hat{y}_D$ do not work as well in practice as the fitted $\mathbf{w}$. As we saw in Figure 4, a linear model is a good description for many queries, reflected in the low error for the best-fit linear models, which suggests restricting the model to be linear may work in our favor.

4.3 Multiscale Representation

In this section we describe the multi-scale representation. In the evaluation, we show this basic technique can greatly help zero-shot CLIP searches on their own, as well as when combined with CLIP and DB alignment. Multi-scale representation maps images to multiple vectors, increasing the size of the vector database. While this is not an issue for the vector database we use, it can be an issue for techniques that scale poorly with the size of the database. Because CLIP and DB alignment scale with the size of the data seen by the user, it is possible to combine them while keeping latency interactive.

The CLIP image embedding model is trained on images of 224×224 pixels. However, in real-world datasets, and also those from the COCO, LVIS and BDD datasets in our evaluation, the raw images are typically larger, in the 800×1000 range. The simplest option to use CLIP with these datasets is to rescale images to fit within this window, which we will call a *coarse* embedding. CLIP itself was trained this way. This coarse embedding approach is how CLIP is most commonly used. When analyzing the types of queries where SeeSaw performs poorly, however, we found this coarse approach misses many objects depending on the object size within the image. For example, wheelchairs and animals often occupy just a few tens of pixels in dash-cam images from the BDD dataset.

Multi-scale patches. An alternative to the coarse embedding approach is to treat images as a tiling of image patches at multiple size scales. Each patch is then encoded separately using CLIP, yielding multiple vectors per image. In our experiments with this multi-scale representation, we used the simplest possible combination of scales: a large-scale patch covering the full image, i.e., the coarse embedding, plus a finer-grained tiling of 1/2 the size of the image, as long as the resulting patch was larger than 224 pixels. For example, an image of size 448×448 maps to one coarse tile of size 448×448, plus 9 finer-grained tiles of size 224×224, corresponding to a patch of size 224×224 striding the image with a stride length of 224/2, i.e., a 10x increase in vectors per image. A smaller image would only map to one vector. A larger square image would still only include 9 vectors, though a wider image may add more along that dimension.

At query time, an image's score is computed as the maximum score of any of its patches. This choice means SeeSaw uses the vector store to find high-scoring patches, rather than high-scoring images, helping SeeSaw return results where only a part of the image is relevant to the final result, and its individual benefit is shown in the evaluation.

To integrate the multi-scale representation with the user's box annotations, the region boxes corresponding to patches on the image that we have indexed ahead of time are compared to the box patches drawn as feedback by the user. Boxes that overlap with the user feedback are considered positives, and boxes with no overlap are considered negatives for the purposes of creating a training set X_t, y_t for the next round.

4.4 Solving for w

$$\mathbf{q}_{t+1} = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{w}, \mathbf{x}_{i=1}^{t}, y_{i=1}^{t}) \tag{5}$$

$\mathcal{L}(\mathbf{w}, \mathbf{x}_{i=1}^{t}, y_{i=1}^{t}) \coloneqq$	
$\sum_{i=1}^{t} \text{LogLoss}\left(y_i, \text{sigmoid}(\mathbf{w}^\top \mathbf{x}_i + b)\right)$	*fit user feedback*
$+\lambda\lvert\mathbf{w}\rvert^2$	*but avoid* $\lvert\mathbf{w}\rvert \to \infty$
$+\lambda_{\text{text}}\left(1 - \frac{\mathbf{w}^\top \mathbf{q}_{\text{text}}}{\lvert\mathbf{w}\rvert\lvert\mathbf{q}_{\text{text}}\rvert}\right)$	*prefer* $\mathbf{w}$ *aligned with* $\mathbf{q}_{text}$
$+\lambda_{\text{DB}}\left(\frac{\mathbf{w}}{\lvert\mathbf{w}\rvert}^\top \mathbf{M}_{\text{D}} \frac{\mathbf{w}}{\lvert\mathbf{w}\rvert}\right)$	*prefer* $\mathbf{w}$ *aligned with DB*

Table 1. Loss function

The full loss function is written down in Equation 5 and Table 1

We minimize $\mathcal{L}$ using the PyTorch [35] implementation of the L-BFGS [30] optimization algorithm to solve for $\mathbf{w}$, then we use the solution vector as the next query q_{t+1} into the vector database.

L-BFGS finds the optimal solution in a few tens of steps (taking a few milliseconds) by using second-order derivative information in addition to gradients. We use it for SeeSaw because it converges quickly and also removes the need for learning rate tuning (and also the possibility of divergence or no convergence). Note in particular that the loss function computations grow with the amount of feedback the user provides, not with the size of the database.

5 EVALUATION

The main goals of this evaluation are:

(1) To compare SeeSaw to zero-shot CLIP, which uses clip alone and no feedback, few-shot CLIP, which uses user feedback to train a logistic regression model, and a state-of-the-art active learning search baseline which also leverages user feedback: Efficient Non-myopic Search (ENS) [23]. We find SeeSaw is able to show consistent improvement in Average Precision (AP) across

four datasets and thousands of search queries, in contrast to few-shot CLIP and ENS, which show consistent drops in AP with respect to zero-shot CLIP (§5.1). We discuss how the ENS accuracy drop is partly due to a stronger reliance on ranking scores being well calibrated as probabilities in §5.4, something CLIP does not provide.
(2) To show how the different SeeSaw components: CLIP alignment, query alignment, and multiscale representation interact and contribute to the overall SeeSaw results in (§5.3).
(3) To show SeeSaw translates these algorithmic insights to time savings for users (§5.5).

5.1 Accuracy Benchmark

The ideal benchmark datasets and queries for automated testing of a system like SeeSaw would be varied large-scale image datasets from different domains. The images would not be from the open web, as CLIP and similar models are likely to have been trained on them. The datasets would include labels for thousands of queries of different complexity. Results for these queries within the datasets should be both rare in an absolute sense and, more importantly, hard to find with zero-shot CLIP alone. A smaller subset of hundreds of queries where zero-shot CLIP performs very well would be included to flag regressions caused by different proposed methods. The annotations for each query would include the relevant region within the image in order to let region-based feedback algorithms approximate human input for each image.

Because such a benchmark does not exist and creating it would be extremely costly and laborious, we opt instead to adapt several object detection and classification datasets: LVIS[19], ObjectNet[3], BDD[51], and COCO[28]. These labeled object detection datasets allow us to use their object annotations for testing region feedback, as well as for evaluating results.

Benchmark task. For a given category and dataset, our benchmark task consists of finding 10 examples of the category, starting with the category name string as the initial text query. We stop at 60 images if 10 examples have not been found by then. 60 images correspond to bounding searches to less than 5 minutes based on our measurements that show users may take up to 5 seconds per image (Table 5), a quantity that will depend on the dataset and amount of box feedback provided. Though the exact cutoff choices of 10 and 60 are not critical for the results we report, having some kind of cutoff at low values ensures the reported metrics do not reflect improvements attainable only after receiving a large number amount of feedback or improvements which are only visible deep into the list of results.

Average Precision (AP). We measure result quality through AP. As discussed earlier, AP for a single query is the average of the individual precision scores computed at every possible recall threshold: $AP = (\sum_{i=1}^{R} P_i)/R$, where R is the number of relevant results in the data and P_i is the precision if we cut results off after the i^{th} relevant result in the data. Because we consider up to 10 relevant results, only ten precision scores will be included in the average. For queries with R < 10 we use R instead. When less than the target number of results are found, the remaining precision scores are set to 0. AP ranges 0 to 1, with 0 meaning no results were found within 60 images, and 1 meaning the first ten images were all positive. In addition to stopping the benchmark at 60 images, within those 60 images AP favors earlier results rather than later ones.

Zero-shot CLIP results. The benchmark task simulates a scenario where a user starts a search query via text and then interacts with it via region box feedback. For a given dataset category such as wheelchair, we use the text string “a wheelchair” to start the search. After receiving the first image result, the benchmark code uses the dataset ground truth to determine when the image is relevant, and then provides box labels from the dataset as region based feedback around the relevant image area. For zero-shot CLIP this feedback is ignored, and we obtain the results shown in Figure 1.

Note the long left-tails in the figure. ObjectNet and LVIS, in particular, help broaden this evaluation thanks to their large number of categories (respectively 300 and 1400), and a corresponding large number of queries on the left tails. COCO and LVIS, like many other standard benchmarks in object detection, share the same underlying set of images taken from the open web (Flickr), which suggests CLIP is likely to have seen the images during training. LVIS includes annotations for many objects besides the main subject of their images, in contrast to COCO, which is reflected in COCO having almost no density at lower accuracies. BDD consists of a small number of classes from driving scenes, and shows a similar problem of low density because most of the classes (car, person, etc.) are ones CLIP was likely trained extensively on.

These observations suggest even datasets with high mean AP on standard labels in practice have hidden long-tails when it comes to ad-hoc queries. Because average metrics over queries in a dataset are dominated by the high-density regions where CLIP performs well, but we are interested in “long tail" queries in SeeSaw, we use a cutoff line at $AP < .5$ to define a hard subset of the queries, marked in the plot by the dashed line. The labels on the dashed line indicate the fraction and total number of classes in the dataset that are below this threshold.

COCO and BDD are designed to test object detection search, hence many labeled classes are not rare in absolute terms, which makes their left tails much less informative. However they are larger datasets both in image size and number of images, so they help test the latency of different algorithms.

ObjectNet only includes images of size of size 224×224, the same size used to train CLIP, with intentionally centered objects. This dataset feature could push baseline CLIP to perform better than it would otherwise, but ObjectNet still challenges CLIP for a several tens of queries.

5.2 SeeSaw Benefits

In later sections we show zero-shot CLIP performs better on average than the other baselines we evaluate. Hence, in this section we show that SeeSaw beats zero-shot CLIP, especially on classes that CLIP struggles with, by running SeeSaw on the benchmark task defined previously and quantifying the distribution of the change in AP (ΔAP), rather than only the average change.

The hyperparameters of SeeSaw for this experiment are $k = 10$ for the kNN graph, $\sigma = .05$ for the distance kernel used between vertices in the graph, $\lambda = 100$ for the norm regularization, $\lambda_c = 10$ for clip-alignment regularization, and $\lambda_D = 1000$ for database-alignment regularization. We chose these parameters based on average performance on the queries. We note the same hyperparameters were used for all queries and datasets, and the different datasets seem to peak at the same parameter values. Varying these λ one order of magnitude decreased the absolute accuracy slightly, but SeeSaw remains substantially more effective than the baseline. Varying k from 5 to 20 also did not substantially affect results.

Figure 5 shows ΔAP broken down by dataset (left column). Note that zero-shot CLIP reaches AP of 1 (see Figure 1) for several object classes, which guarantees $\Delta\text{AP} \leq 0$ for those queries, reflected by the vertical jump at $\Delta\text{AP} = 0$ in the CDF. In order to highlight the change on the tail queries, we show the distribution of ΔAP restricted to the hard subset of queries on the right column. A lower starting AP allows ΔAP to be generally larger.

Here, the gray shaded area marks the $[.1, .9]$ quantile interval. The red-shaded region (AP<0) shows queries where SeeSaw did not improve performance. In general, SeeSaw is quite robust, with more than 90% of the queries improving or staying the same (we a breakdown of average AP results on these datasets in the next section). The solid bars plot the min, median and max ΔAP. In many cases the min is very close to 0. The main reason for a few outliers on the left seems to be the multiscale representation, which in most cases is beneficial, but can sometimes push the first result of a query a few spots down the rank. This can affect AP strongly if the original AP was 1. Methods to reap the benefits of multiscale while limiting any downside are an interesting line of future work.

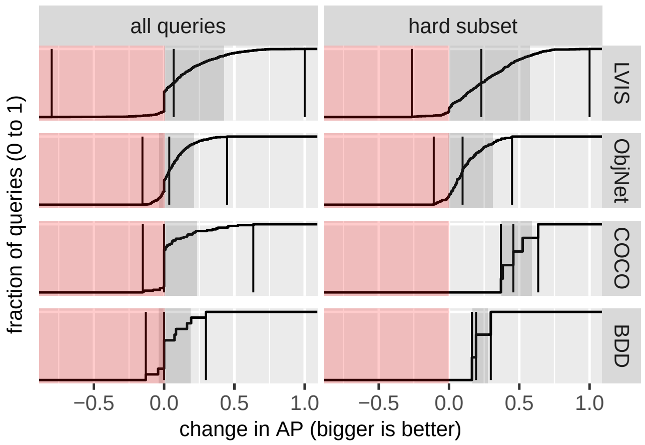

Fig. 5. CDF of change in AP of SeeSaw on top of zero-shot CLIP, broken down by dataset. The red shade marks the region with ΔAP < 0. The darker gray shaded area marks the [.1, .9] quantile interval of observed ΔAP so that the robustness of the improvement is evident. The solid vertical lines mark the minimum, median and maximum change observed for completeness. The average values are shown in Table 2

5.3 SeeSaw Breakdown

In this section we breakdown the contributions from the different SeeSaw components to the overall accuracy in the benchmark presented before.

Table 2 shows how much each of these optimizations contributes to the total Average Precision (AP) in SeeSaw by adding optimizations one at a time and recording the aggregate changes in AP. It shows increases in mean AP for each optimization (rows), on each dataset (columns). The row states the mean AP when using CLIP embeddings out of the box. Each later row adds one of the optimizations in SeeSaw. The rightmost column "avg" is the average of the values to the left. It shows all optimizations contribute to the final results on average. The rest of the columns show the mean AP for each dataset, and each shows different benefits from different optimizations in SeeSaw. Each row represents the addition of a different optimization described in the indicated section of the paper. Zero-shot CLIP is CLIP without any user input; few-shot CLIP is CLIP (with multiscale, in this case) plus logistic loss from user feedback (Equation 1), which is also a baseline approach to relevance feedback on its own right.

The exact mean Average Precision (mAP) values mostly reflect the high initial mAP of zero-shot CLIP for many queries, as explained in Figure 1, here focus more on the relative sizes of the changes, which may transfer better to ad-hoc scenarios. Every row of optimization consistently increases mAP with the exception of few-shot CLIP, which causes a drop on most columns. We referenced this phenomenon in the introduction, as few-shot CLIP is a reasonable, baseline for relevance feedback. We note few-shot CLIP when combined with alignment methods undo this regression. The bottom row is equivalent to total SeeSaw mAP.

Few-shot CLIP does improve the mean AP for the hard subset of queries in LVIS, something we observe consistently on the hard subset of queries when multiscale is off (not shown in this table). Because zero-shot CLIP is very accurate, the weakness of the few-shot approach is more apparent on the top half of the plot and motivates CLIP alignment, which turns out to also offer strong benefits in hard queries where few-shot CLIP improves results, as in LVIS. When the zero-shot CLIP vector performs the worst at finding initial results we may be the most tempted to ignore that vector weight and prefer the data, but this would be a mistake in the setting of small samples because the generalization error of the learned vector is likely dominated by *variance* error, whereas the zero-shot CLIP vector error consists of *bias* error.

Finally, we highlight the importance of algorithms that can scale to handle the growth in vectors that multiscale brings, as this brings has strong benefits for the search. In Table 2 these rank second only to CLIP align, especially on BDD which has the largest images, but also on LVIS, which includes many objects on different image locations. ObjectNet does not benefit from multiscale, as it consists of standard fixed size images. Additionally, multiscale interacts with CLIP align in more complex ways than just latency: CLIP align and DB align are less effective in absolute terms when multiscale is not enabled. The data on Table 3, which we will explain fully later, includes a row for SeeSaw without multiscale enabled. Especially on BDD, the 3 hard queries improve from .02 to .07 without multiscale, but from .10 to .24 with it.

		LVIS	ObjNet	COCO	BDD	avg.
all queries	zero-shot CLIP	0.63	0.64	0.90	0.74	0.72
	+multiscale (§4.3)	0.70	0.64	0.95	0.76	0.76
	+few-shot CLIP	0.67	0.59	0.87	0.68	0.70
	+Query align (§4.1)	0.75	0.69	0.96	0.77	0.79
	+DB align (§4.2)	0.76	0.70	0.96	0.79	0.80
hard subset	zero-shot CLIP	0.19	0.28	0.27	0.02	0.19
	+multiscale	0.32	0.28	0.58	0.10	0.32
	+few-shot CLIP	0.34	0.28	0.57	0.07	0.31
	+Query align	0.42	0.39	0.74	0.20	0.44
	+DB align	0.44	0.40	0.75	0.24	0.46

Table 2. Increases in mean Average Precision (mAP) for each optimization (rows), on each dataset (columns). The top panel shows the averages over all queries, while the bottom panel is restricted to the hard subset of queries for each dataset. All techniques contribute to the final result, their relative contribution depends on properties of the data.

5.4 Comparison with Baselines

We compare SeeSaw to the following baselines: ENS from the recent active learning literature and Rocchio's algorithm,[40], a classic relevance feedback algorithm.

Rocchio's algorithm[31, 40]. Rocchio's algorithm creates a new query vector at every iteration, q_t, by taking the initial query vector used, q_0 and adding the average of the set of relevant example vectors returned so far (D_r) (with a weight of β, and subtracting the average of the nonrelevant example vectors seen so far(D_n) (with a weight of γ). The formula we use is Equation 6:

$$\mathbf{q}_n = \alpha \mathbf{q}_0 + \frac{\beta}{|D_r|} \sum_{d_j \in D_r} \mathbf{d_j} - \frac{\gamma}{|D_n|} \sum_{d_j \in D_n} \mathbf{d_j} \tag{6}$$

Like SeeSaw, Rocchio's algorithm includes hyperparameters that weight the three terms. For the experiments below, we use the hyperparameters that maximized the average AP across all datasets: $\beta = .5$ and $\gamma = .25$. Additionally, following [31], we tested $\gamma = 0$, but AP was higher with our choice of $\gamma = .25$. α was set to 1 as any other value would be equivalent after rescaling.

Efficient Non-myopic Active Search (ENS) [23]. ENS is a state-of-the-art baseline from the active learning literature. Unlike much active learning work that aims to optimize the accuracy of a model by judiciously choosing data points to label, ENS is an *active search* algorithm whose goal is to maximize the number of results found for a fixed amount of human input, which is similar to the setting of SeeSaw. ENS has been shown to be more efficient than previous work in the area such as [16].

While both SeeSaw and zero-shot CLIP select the next result greedily on every iteration, ENS takes a long view: it picks the next result based on its estimate of how this choice affects the total number of results found after t steps in the future (in the benchmark case, $t = 60$). For example, ENS may pick the second-highest scoring image now if it lies within a dense cluster of similarly scored points. In that scenario, if the image turns out to be relevant then ENS can estimate it is more likely to find many more results in succession. This choice could be wiser than picking the highest scoring vertex in the kNN graph if this result is isolated. Zero-shot CLIP would not consider this scenario. SeeSaw would pick the highest scoring vector just like zero-shot CLIP, though DB alignment will affect the scores of isolated points.

We implemented the ENS algorithm in SeeSaw. ENS makes use of a kNN graph, the same one we used for DB alignment. For its hyper-parameters we used $k = 20$ since that improved ENS results, and we used the same $\sigma = .05$ as for SeeSaw. We changed our version of ENS to integrate CLIP into it: we use CLIP scores as an individual γ_i for each vertex so ENS also has access to CLIP as a prior. The γ parameter acts as a prior score in ENS, providing a score for vertices without labeled neighbors. The original ENS uses a global $\gamma = .1$ hyperparameter. Additionally, we wait for zero-shot CLIP to find a first positive result from the data before starting ENS. Both of our modifications help ENS perform better in the benchmark. For simplicity, for this benchmark we set the time horizon $t = 60$ initially, and reduce it after every step so ENS can make optimal decisions given the time remaining.

Baseline results. We ran the benchmarks described in §5.1, including zero-shot CLIP, SeeSaw, and show the mAP results in Table 3. In the table, each row corresponds to one of these methods and each column to a different dataset. The rightmost column is the average of those to the left and shows that SeeSaw increases mAP further. As we do in Table 2, we show averages over both the hard subset and all queries. The hard subset numbers tend to spread out wider, which is more helpful to display differences, We note SeeSaw AP is highest for all datasets in this long tail of hard queries. Rocchio comes in second, and is slightly better than SeeSaw for COCO when aggregating across all queries. This behavior is consistent with Rocchio also using a form of CLIP alignment through the q_0 term. We note that few-shot CLIP generally lags behind Rocchio in AP, and for some datasets is also worse than zero-shot CLIP, showing the importance of regularizing by the initial query, done implicitly in Rocchio and explicitly in SeeSaw. Table 3 also shows ENS can decreases AP with respect to zero-shot CLIP. Note that because we only implemented ENS for coarse embedding, without multiscale, in Table 2 we compare all baselines without multiscale enabled. SeeSaw with multiscale enabled was evaluated in §5.3.

ENS analysis. One reason for the drop in ENS is that the ENS logic to estimate the expected reward is sensitive to score calibration. Calibrated scores need to be correct not only in the ranking they produce but also correct when interpreted as probabilities: for example, 10% of points with a calibrated score of .1 should be positive. Calibration is a strong assumption; we note that CLIP scores are not calibrated.

ENS depends on score calibration because it uses the probabilities as weights to compute expected values. The longer the reward-horizon hyper-parameter t, the more terms this sum has and the more susceptible the score is to calibration errors. We test this hypothesis by calibrating CLIP

		LVIS	ObjNet	COCO	BDD	Avg.
all queries	zero-shot CLIP	0.63	0.64	0.90	0.74	0.72
	few-shot CLIP	0.65	0.58	0.88	0.73	0.71
	ENS[23]	0.50	0.43	0.86	0.70	0.62
	Rocchio[40]	0.68	0.70	**0.93**	0.75	0.76
	this work	**0.69**	**0.70**	0.92	**0.76**	**0.77**
hard sub-set	zero-shot CLIP	0.19	0.28	0.27	0.02	0.19
	few-shot CLIP	0.25	0.28	0.32	0.06	0.23
	ENS	0.16	0.24	0.37	0.03	0.20
	Rocchio	0.28	0.38	0.49	0.05	0.30
	this work	**0.30**	**0.40**	**0.55**	**0.07**	**0.33**

Table 3. Table comparing mean Average Precision (mAP) of SeeSaw different baseline algorithms. No method used multiscale in this table.

reward horizon $t =$	1	2	10	60
raw γ_i	0.63	0.62	0.61	0.55
calibrated γ_i	0.65	0.65	0.65	0.63

Table 4. Mean AP scores for ENS averaged over the four datasets, showing the effect of varying the time horizon hyperparameter (columns), and the effect of calibrating the initial γ_i scores into probabilities (second row). The results of the calibrated row are not attainable in practice because calibration requires labeled data.

scores s_i for each vector into probabilities via Platt scaling [34, 36]. We emphasize this calibration is not possible in a real deployment because it requires labeled data ahead of time. Then we run ENS using this carefully tuned γ_i, and show the average of mAP over all the datasets for of ENS without γ_i calibration on the top row of Table 4, and those with calibrated γ_i in the second row.

The table shows ENS performs better if the initial γ_i are calibrated using Platt scaling, with access to the ground truth data, showing ENS is sensitive to calibration. Furthermore, mAP degrades sharply as we increase the reward horizon parameter used internally by ENS to compute expected values, whereas it degrades less sharply in the bottom row, showing that larger reward horizons are more sensitive to poor calibration. In contrast, greedy methods such as zero-shot CLIP do not depend on the calibration of the scores. SeeSaw, in particular, optimizes dot product alignment in Equation 3 rather than probability estimates.

A second reason for the mAP drop in ENS is that the kNN classifiers used internally for ENS are less efficient at learning from small samples than the linear models used by SeeSaw, as long as a linear model is a good reflection of the data, which is true for CLIP embeddings. This is reflected in Table 4 on the column for time horizon $t = 1$, an extreme setting where ENS effectively becomes a greedy kNN-model.

5.5 End-to-End Tests

The primary goal of this section is to test SeeSaw with real people in different scenarios to how different variables affect the overall time it takes to complete a task with SeeSaw or otherwise. The overall time includes not just the time savings due to increased accuracy but also the time costs due to user feedback latency as well as any possible system latencies. A secondary goal is to show estimates of how much time it takes a user to provide feedback to SeeSaw, and how this time varies with the type of annotation.

The effect of annotation time overhead from using SeeSaw is ameliorated by two factors: the first is that users only annotate positive examples, and in hard searches that require inspecting many images, positive examples are rare. Conversely, when positive examples are common, SeeSaw overhead due to labeling is larger, but the searches themselves complete much more quickly.

The baseline for this evaluation is the same user interface and back-end of SeeSaw, but with all the optimizations of §4 disabled, which means zero-shot CLIP with a User Interface (UI) for searching.

In this baseline system, SeeSaw provides users with a keyboard binding to mark the whole image as relevant. This input is necessary to ensure that users know when they completed the task (the system shows a total count), but is not used as feedback. We recruited 20 graduate students and 20 Amazon MTurk workers for these tasks. Both groups were assigned the same mix of tasks on both systems.

Like our task in §5.1, we ask users to find 10 examples of the given concepts and we stop each task after 6 minutes. Note the evaluation in §5.1 involved no real users – instead we use dataset ground truth to provide feedback and we stop the query after finding 10 examples or going through 60 images. Instead of measuring AP, in this evaluation we measure the *time* it takes users to complete the task (find 10 examples).

Because human time is a very limited resource, in this section we evaluate SeeSaw and the baseline on 7 queries. Unlike in §5.1, our goal in this experiment is not to be comprehensive, but to observe how searches play out in different scenarios. Guidance on how to use, including how to provide feedback and the keyboard bindings of the UI was provided through a tutorial screen followed by two initial queries used for training users on each version of the system, and whose results were discarded. Each user then completed seven tasks using both interfaces. The subset of queries each user completed on each interface did not overlap, as in early tests we found the time it takes to provide feedback or skip an image can be artificially low the second time the same image is seen for the same query.

We show aggregate results for these experiments in Figure 6, where the x axis measures the time elapsed and ends at 6 minutes (360 seconds). The 7 queries we tested are logically grouped into two sets: one set of "hard" queries and one of "easy" queries, which in this context simply means queries where the zero-shot CLIP has a low or high AP accuracy. The main goal of this section is to show how this affects time.

We show the results for the baseline and SeeSaw side by side distinguished by color and dashes. The middle triangle corresponds to the median time taken by users. The error bars on the plot correspond to a bootstrapped 95%-CI of the time required, capturing inter-user variability for a given query.

For the hard queries, the median time for the baseline is 360 seconds, ie. half or more of the users could not complete the task within that time limit, a qualitative difference with SeeSaw.

For "wheelchair" and "car with open door" none of the baseline users were able to complete the task, so the corresponding error bars collapsed into the 360 mark. The most challenging query, "car with open door," was completed only by a few people, all using SeeSaw, though most people, even most of those using SeeSaw, were not able to complete it. The times required to find wheelchairs and dogs show SeeSaw helped substantially shorten times in these cases, even after accounting for inter-user variability.

For easy queries, the figure shows that SeeSaw can be slower than the baseline; this difference boils down to the annotation overhead per frame. Table 5 shows when images are marked as relevant the overhead for selecting the region bounding box adds about a 50% latency (4.5 vs. 3 seconds per image). For these queries, SeeSaw adds the worst possible relative overhead but the absolute latency values are low for both methods.

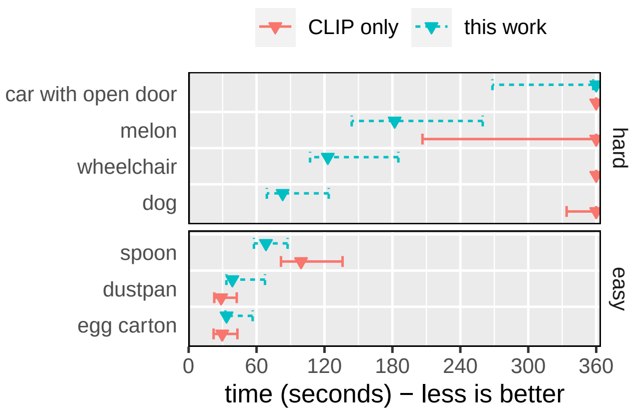

Fig. 6. Comparing time elapsed before users find 10 examples of each of the categories, or 6 minutes pass (rightmost on x). A value of 360 seconds corresponds to not complete the task. The median time for each method is represented by a triangle. The error bars mark the 95^{th} percentile CI for the mean time, to show variation between users.

	baseline	seesaw
not marked	1.98 ± .10	2.40 ± .19
marked relevant	3.00 ± .28	4.40 ± .45

Table 5. User annotation time (s) per image depends on whether the image is marked relevant. (2s vs 3s). Localized feedback adds an overhead of about 1.5 seconds per image (4.5s) The ± denotes the 95% CI.

5.6 System Latency and Scalability

SeeSaw aims to scale to larger image datasets. One challenge when scaling to larger datasets is to keep system latency low, and system latency depends partially on the algorithms used for selecting new results based on feedback. Table 6 shows system latencies as we increase the scale of the datasets. Each row corresponds to a different dataset, ordered on increasing scale (number of vectors in database). We include both the coarse and multiscale representations as different rows, as they imply differences in number of vectors.

We highlight how SeeSaw latency remains within the hundreds of milliseconds, while a propagation-based version of SeeSaw shows latency increasing beyond a few seconds, one motivation for our DB alignment approximation in §4.2.

	vectors	CLIP	ENS	Rocchio	SeeSaw	prop.
ObjNet$^-$	50K	0.11	0.10	0.14	0.27	0.83
BDD$^-$	80K	0.09	0.11	0.10	0.23	0.90
COCO$^-$	120K	0.10	0.22	0.16	0.34	1.11
BDD	1.6M	0.13	NA	0.16	0.34	2.95
COCO	1.6M	0.14	NA	0.23	0.47	2.88

Table 6. System latency per iteration (seconds) vs dataset size (# vectors). The extra $^-$ sign next to a dataset name means coarse indexing (one vector per image), as opposed to multiscale indexing. COCO and LVIS share the same database, so we only show one.

5.7 Hyperparameters

SeeSaw requires hyperparameters λ, λ_c, λ_D. In this section we show SeeSaw handles hyperparameter values varying an order of magnitude while still improving results vs. zero shot CLIP. Table 7 shows values for each dataset, highlighting the maximum achieved. The enclosed row highlights the values we used for the evaluation benchmark. We note two things: the AP values are emparically optimal or near-optimal at similar hyperparameter values, even though the datasets are different. For a new dataset, we recommend starting with hyperparameter values in the same range.

λ_c	λ_D	λ	BDD	COCO	LVIS	ObjNet	Avg.
3	300	100	0.78	0.96	0.76	0.68	0.80
3	1000	100	0.77	0.97	**0.77**	0.68	0.80
3	3000	100	0.77	0.96	0.76	0.63	0.78
10	300	100	0.78	0.96	0.75	0.69	0.80
10	1000	30	0.79	0.96	0.76	0.70	0.80
10	1000	100	**0.79**	0.96	0.76	**0.70**	**0.80**
10	1000	300	0.79	0.96	0.76	0.70	0.80
10	3000	100	0.79	**0.97**	0.77	0.69	0.80
30	300	100	0.77	0.96	0.73	0.68	0.79
30	1000	100	0.77	0.96	0.74	0.69	0.79
30	3000	100	0.77	0.96	0.74	0.69	0.79

Table 7. SeeSaw AP for different hyperparameter settings

6 RELATED WORK

SeeSaw relates to work in the following areas:

Relevance feedback [41] with explicit region annotation feedback, a common interface for images[53]. SeeSaw's novelty lies in the specific algorithms it uses, which make it work better than existing approaches on CLIP embeddings. Rocchio's algorithm[40] is one of the baselines we used in this paper.

Systems for image search, such as [47], PicHunter [10], MindReader [21] and Falcon [48]. Much of this work aims to help users query by example. CLIP enables us to start the query by using text, and we find it is important to use the CLIP query vector as a regularizer term in 4.1. Some existing work, such as the baseline approach in [18], use the query as a positive example. Unlike Falcon, and other work such as [39] which aim to navigate non-convex results. However, because CLIP embeddings seem to well clustered SeeSaw can focus on this specific case.

Text-image retrieval aims to search images with text. Recent work includes Drilldown [45]. The goal is this work is to provide a richer interface with text-based refinement, though we are able to assume some amount of labeling on the user's data. SeeSaw aims to help the tail of queries on a user's database where CLIP does not work well by leveraging user input, without requiring user labels.

Active learning [42] also studies human-in-the-loop labeling but does not directly address the search problem of relevance feedback. **Active search** [17] is a subset of active learning techniques for searching datasets more directly related to SeeSaw. ENS is a state of the art of active search approach we compared SeeSaw to in our evaluation. Active learning can often be expensive at runtime on large datasets due to linear or super-linear scaling. Recent work such as [9] points out this scaling problem and instead aims to scale these approaches by restricting active learning algorithms only to the neighborhood of search queries and leveraging vector stores to explore that neighborhood. SeeSaw's approach to database alignment§4.2 shows there can be some value to considering the global database structure in addition to the immediate neighborhood of the query.

Semi-supervised learning [55] proposes approaches for leveraging both labeled and unlabeled data. §4.2 in particular, can be seen as semi-supervised learning, or domain adaptation. Label propagation [54] is a semi-supervised learning technique and has been used together with active learning for search in [56]. Label propagation can be expensive to scale. Manifold regularization [4], another topic in semi-supervised learning, is closely related to both label propagation and to the regularization term we use in §4.2. Manifold regularization by itself is expensive, just like label propagation, but collapsing the term into a small quadratic term as done in §4.2 makes it possible.

High-recall or total-recall retrieval [18],[1],[52], aims to find all relevant results within a database. This is an important task in legal contexts for example, where legal discovery needs to find examples and also provide some assurance that a substantial part of the results was not missed. Some of the approaches proposed in the area are similar to the few-shot CLIP approach. Total-recall retrieval is a harder problem than finding some positive results within the database, and may require a large amount of exploration of likely irrelevant results to provide some assurance about all regions of the database.

Multiscale representations The usefulness of multiscale representations of images of one form or another has been recognized for a long time [2, 6, 44], and internal representations used by CNN-based object detectors [11, 29] incorporate it in different ways. In this work, this general idea is used to overcome some of CLIPs limitations, allowing us to handle more complex images with relevant object in small portions of images.

Query relaxation techniques[13] rewrite or modify user queries by removing tokens or constraints from the query. These techniques help increase recall when initial queries are unnecessarily restrictive. In SeeSaw, the text regularization term is important to improve results compared to using only image examples, which could be interpreted as a type of query relaxation. On the other hand, typical query relaxation techniques will textually drop modifers.

Active learning for data cleaning [27], aims to help users build better models by reducing the amount of data that must be cleaned. Additionally active learning has been applied to other database problems like **entity resolution** [25] and **schema matching** [49]. Here we also employ active learning, focusing on the problem of dataset labeling.

Finally, ***Why* and *Why-Not* provenance** techniques help users understand how their data relates to their query results and, conversely, why some of their data was excluded from the results (see [8] for a general survey of many papers). Systems like SeeSaw that evolve their result set of over time using models are generally related, although the specifics of our techniques differ as we focus on CLIP query vector refinement rather than a more relational setting.

7 CONCLUSION

We presented SeeSaw, a system to help users find objects in their image datasets that incorporates their feedback in the form of region-based annotations in order to improve their results. SeeSaw helps users leverage pretrained embeddings which have not been fine-tuned for their specific datasets due to a lack of captioned data. SeeSaw's approach consists of framing this problem as a semi-supervised learning problem, combining two kinds of regularization objectives into a loss function, in order to handle the lack of labeled data. At the same time SeeSaw leverages a multi-scale multi-vector representation of images in order to handle embedding model limitations, and integrates it with the relevance feedback mechanisms. In order for this integration to be successful, SeeSaw focuses on keeping latency low despite the larger dataset sizes implied by this representation. We benchmark SeeSaw extensively on thousands of queries, showing consistent empirical advantages over an active learning baseline, a relevance feedback baseline from the information retrieval community, and practical baselines such as few-shot learning.

Interesting directions for future work include reducing or removing explicit user feedback, for example by leveraging newer generations of models such as GPT-4, as well as further taking advantage of the abilities of vision-language models such as CLIP to provide textual feedback.

ACKNOWLEDGMENTS

Research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Mustafa Abualsaud, Nimesh Ghelani, Haotian Zhang, Mark D Smucker, Gordon V Cormack, and Maura R Grossman. 2018. A System for Efficient High-Recall Retrieval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 1317–1320.

[2] E Adelson, P Burt, C Anderson, J M Ogden, and J Bergen. 1984. PYRAMID METHODS IN IMAGE PROCESSING. *undefined* (1984). https://www.semanticscholar.org/paper/e49793511ba203e26b99e7e81fd15a7d505b5cea

[3] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. 2019. ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In *Advances in Neural Information Processing Systems*, H Wallach, H Larochelle, A Beygelzimer, F d\'Alché-Buc, E Fox, and R Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/97af07a14cacba681feacf3012730892-Paper.pdf

[4] Mikhail Belkin and Partha Niyogi. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. https://www.jmlr.org/papers/volume7/belkin06a/belkin06a.pdf. https://www.jmlr.org/papers/volume7/belkin06a/belkin06a.pdf Accessed: 2023-3-7.

[5] E. Bernhardsson. [n.d.]. ANNOY: Approximate Nearest Neighbors Oh Yeah. https://github.com/spotify/annoy. Accessed: 2021-05-20.

[6] P Burt and E Adelson. 1983. The Laplacian Pyramid as a Compact Image Code. *IEEE Trans. Commun.* 31, 4 (April 1983), 532–540. https://doi.org/10.1109/TCOM.1983.1095851

[7] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. 2019. UNITER: UNiversal Image-TExt Representation Learning. (Sept. 2019). arXiv:1909.11740 [cs.CV] http://arxiv.org/abs/1909.11740

[8] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Found. Trends Databases* 1, 4 (2009), 379–474. https://doi.org/10.1561/1900000006

[9] C Coleman, E Chou, J Katz-Samuels, and others. 2022. Similarity search for efficient active learning and search of rare concepts. *Proceedings of the* (2022). https://ojs.aaai.org/index.php/AAAI/article/view/20591

[10] I J Cox, M L Miller, T P Minka, T V Papathomas, and P N Yianilos. 2000. The Bayesian image retrieval system, PicHunter: theory, implementation, and psychophysical experiments. *IEEE Trans. Image Process.* 9, 1 (Jan. 2000), 20–37. https://doi.org/10.1109/83.817596

[11] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. 2014. Fast feature pyramids for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 8 (Aug. 2014), 1532–1545. https://doi.org/10.1109/TPAMI.2014.2300479

[12] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web - WWW '11* (Hyderabad, India). ACM Press, New York, New York, USA. https://doi.org/10.1145/1963405.1963487

[13] Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. 2011. Query relaxation for entity-relationship search. In *The Semanic Web: Research and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 62–76. https://doi.org/10.1007/978-3-642-21064-8_5

[14] Fartash Faghri, David J Fleet, Jamie Ryan Kiros, and Sanja Fidler. 2017. VSE++: Improving Visual-Semantic Embeddings with Hard Negatives. (July 2017). arXiv:1707.05612 [cs.LG] http://arxiv.org/abs/1707.05612

[15] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc Aurelio Ranzato, and Tomas Mikolov. 2013. DeViSE: A Deep Visual-Semantic Embedding Model. In *Advances in Neural Information Processing Systems*, C J Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/7cce53cf90577442771720a370c3c723-Paper.pdf

[16] Roman Garnett, Yamuna Krishnamurthy, Xuehan Xiong, Jeff Schneider, and Richard Mann. 2012. Bayesian Optimal Active Search and Surveying. (June 2012). arXiv:1206.6406 [cs.LG] http://arxiv.org/abs/1206.6406

[17] Roman Garnett, Yamuna Krishnamurthy, Xuehan Xiong, Jeff Schneider, and Richard Mann. 2012. Bayesian Optimal Active Search and Surveying. (June 2012). arXiv:1206.6406 [cs.LG]

[18] Maura R Grossman, G Cormack, and Adam Roegiest. 2016. TREC 2016 Total Recall Track Overview. *TREC* (2016).

[19] Agrim Gupta, Piotr Dollár, and Ross Girshick. 2019. LVIS: A Dataset for Large Vocabulary Instance Segmentation. *arXiv [cs.CV]* (Aug 2019). https://arxiv.org/abs/1908.03195

[20] Wildlife Insights. [n.d.]. Wildlife Insights. https://www.wildlifeinsights.org/ Accessed on Mar 26, 2023.

[21] Y Ishikawa, R Subramanya, and C Faloutsos. 1998. MindReader: Querying Databases Through Multiple Examples. *VLDB J.* (1998). https://www.semanticscholar.org/paper/04938be9fd727ea6363cc950efd263ff82d02b77

[22] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V Le, Yunhsuan Sung, Zhen Li, and Tom Duerig. 2021. Scaling up visual and vision-language representation learning with noisy text supervision. (Feb. 2021). arXiv:2102.05918 [cs.CV] http://proceedings.mlr.press/v139/jia21b/jia21b.pdf

[23] Shali Jiang, Gustavo Malkomes, Geoff Converse, Alyssa Shofner, Benjamin Moseley, and Roman Garnett. 2017. Efficient Nonmyopic Active Search. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1714–1723. https://proceedings.mlr.press/v70/jiang17d.html

[24] F Jing, M Li, H-J Zhang, and B Zhang. 2004. Relevance feedback in region-based image retrieval. *IEEE Trans. Circuits Syst. Video Technol.* 14, 5 (May 2004), 672–681. https://doi.org/10.1109/tcsvt.2004.826775

[25] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution with transfer and active learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (Florence, Italy). Association for Computational Linguistics, Stroudsburg, PA, USA. https://doi.org/10.18653/v1/p19-1586

[26] Wouter M Kouw and Marco Loog. 2021. A review of domain adaptation without target labels. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 3 (March 2021), 766–785. https://doi.org/10.1109/TPAMI.2019.2945942

[27] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. ActiveClean. *Proceedings VLDB Endowment* 9, 12 (Aug. 2016), 948–959. https://doi.org/10.14778/2994509.2994514

[28] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312 http://arxiv.org/abs/1405.0312

[29] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2016. Feature Pyramid Networks for Object Detection. (Dec. 2016). arXiv:1612.03144 [cs.CV] http://arxiv.org/abs/1612.03144

[30] Dong C Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Program.* 45, 1 (Aug. 1989), 503–528. https://doi.org/10.1007/BF01589116

[31] C.D. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press. https://books.google.com/books?id=t1PoSh4uwVcC

[32] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press. https://doi.org/10.1017/CBO9780511809071

[33] Kevin P Murphy. 2022. *Probabilistic Machine Learning: An Introduction.* MIT Press. https://play.google.com/store/books/details?id=wrZNEAAAQBAJ

[34] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning - ICML '05* (Bonn, Germany). ACM Press, New York, New York, USA. https://doi.org/10.1145/1102351.1102430

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. (Dec. 2019). arXiv:1912.01703 [cs.LG] http://arxiv.org/abs/1912.01703

[36] John C Platt. 2000. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. 10, 3 (June 2000). http://dx.doi.org/

[37] Alec Radford, 1. Jong Wook Kim, 1. Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, and et al. [n.d.]. Learning transferable visual models from natural language supervision. https://cdn.openai.com/papers/Learning_Transferable_Visual_Models_From_Natural_Language_Supervision.pdf

[38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. (Feb. 2021). arXiv:2103.00020 [cs.CV] http://arxiv.org/abs/2103.00020

[39] Jean-Michel Renders. 2018. Active search for high recall: A non-stationary extension of Thompson sampling. In *Lecture Notes in Computer Science*. Springer International Publishing, Cham, 722–728.

[40] J. J. Rocchio. 1971. Relevance feedback in information retrieval. In *The SMART Retrieval System – Experiments in Automatic Document Processing*, Gerard Salton (Ed.). Prentice Hall, Englewood Cliffs, NJ, 313–323.

[41] Gerard Salton and Chris Buckley. 1990. Improving retrieval performance by relevance feedback. *J. Am. Soc. Inf. Sci.* 41, 4 (June 1990), 288–297. https://doi.org/10.1002/(sici)1097-4571(199006)41:4<288::aid-asi8>3.0.co;2-h

[42] Burr Settles. [n.d.]. *Active Learning*. Morgan Claypool.

[43] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press. https://doi.org/10.1017/CBO9781107298019

[44] E P Simoncelli and W T Freeman. 1995. The steerable pyramid: a flexible architecture for multi-scale derivative computation. In *Proceedings., International Conference on Image Processing*, Vol. 3. 444–447 vol.3. https://doi.org/10.1109/ICIP.1995.537667

[45] Fuwen Tan, Paola Cascante-Bonilla, Xiaoxiao Guo, Hui Wu, Song Feng, and Vicente Ordonez. 2019. Drill-down: Interactive Retrieval of Complex Scenes using Natural Language Queries. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 2647–2657.

[46] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. 2015. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Boston, MA, USA). IEEE. https://doi.org/10.1109/cvpr.2015.7298658

[47] Nuno Vasconcelos and Andrew Lippman. [n.d.]. Learning from user feedback in image retrieval systems. https://papers.nips.cc/paper/1999/file/7283518d47a05a09d33779a17adf1707-Paper.pdf. https://papers.nips.cc/paper/1999/file/7283518d47a05a09d33779a17adf1707-Paper.pdf Accessed: 2021-8-12.

[48] Leejay Wu, Christos Faloutsos, Katia Sycara, and Terry R Payne. [n.d.]. FALCON: Feedback adaptive loop for content-based retrieval. http://www.cs.cmu.edu/~christos/PUBLICATIONS/vldb2k-falcon.pdf. http://www.cs.cmu.edu/~christos/PUBLICATIONS/vldb2k-falcon.pdf Accessed: 2022-5-30.

[49] Zhepeng Yan, Nan Zheng, Zachary G Ives, Partha Pratim Talukdar, and Cong Yu. 2013. Actively soliciting feedback for query answers in keyword search-based data integration. *Proceedings VLDB Endowment* 6, 3 (Jan. 2013), 205–216. https://doi.org/10.14778/2535569.2448954

[50] Stuart Young, Johanna Rode-Margono, and Rajan Amin. 2018. Software to facilitate and streamline camera trap data management: A review. *Ecol. Evol.* 8, 19 (Oct. 2018), 9947–9957. https://doi.org/10.1002/ece3.4464

[51] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. 2018. BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *CoRR* abs/1805.04687 (2018). arXiv:1805.04687 http://arxiv.org/abs/1805.04687

[52] Zhe Yu and Tim Menzies. 2018. Total Recall, Language Processing, and Software Engineering. (Aug. 2018). arXiv:1809.00039 [cs.SE]

[53] Xiang Sean Zhou and Thomas S Huang. 2003. Relevance feedback in image retrieval: A comprehensive review. *Multimed. Syst.* 8, 6 (April 2003), 536–544. https://doi.org/10.1007/s00530-002-0070-3

[54] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation. (2002). https://www.semanticscholar.org/paper/2a4ca461fa847e8433bab67e7bfe4620371c1f77

[55] Xiaojin Zhu and Andrew B Goldberg. 2009. *Introduction to Semi-supervised Learning*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00196ED1V01Y200906AIM006

[56] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. 2003. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, Vol. 3. http://mlg.eng.cam.ac.uk/zoubin/papers/zglactive.pdf

Received April 2023; revised July 2023; accepted August 2023