Department of Computer Science

Submitted in part fulfilment for the degree of BEng

Enhancing Image Retrieval in Natural Language Processing Applications

Thomas Crispin Pollak

2/5/2023

Supervisor: Adrian Bors

Acknowledgements

I'd like to thank my supervisor Adrian Bors, for all his help and feedback when writing this dissertation.

Statement of Ethics

The dataset used in the paper is available for academic use. No personally identifiable data is present in the dataset. All authors who made significant contributions to this paper have been acknowledged.

Table of Contents

Table of Figures

Table of Tables

Executive Summary

The increasing amount of images and digital media available today has led to a growing demand for efficient and general methods to retrieve relevant images from large heterogeneous image databases.

This dissertation proposes a novel image retrieval approach combining natural language processing (NLP) and computer vision, aiming to deliver a powerful image retrieval system capable of searching image databases using any natural language query. I leverage OpenAI's Contrastive Language-Image Pretraining (CLIP) multimodal model to predict the similarity between an image and text query, and integrate this with an Approximate k-nearest neighbour (ANN) index to scale to tens of thousands of images. This approach aims to provide an intuitive and powerful search experience, not bound by the constraints of a predetermined set of labels or categories.

This research will assess the effectiveness of the image retrieval system through the merits and tradeoffs of different model architectures and ANN index algorithms, considering factors such as inference speed, model size, accuracy, recall, build time and query time. The significance of these factors may vary depending on the objectives of the production system. By exploring these aspects, this study aims to provide an image retrieval guide for different use cases.

1 Introduction

Conventional image search methods often rely on predetermined labels or metadata associated with images, which can limit their ability to capture the nuanced and diverse range of content. The growing number of large and varied image databases has required the development of a powerful and efficient image search method to enable users to retrieve relevant content that can not only handle the sheer volume of images, but also cater to the diverse queries users may have. Examples of such systems could include: video search on a CCTV camera, Google Image Search, Instagram recommendation algorithms and others.

In this dissertation, I propose an innovative approach to image retrieval that capitalises on the recent advancements in multimodal (image-text) models. In 2021, OpenAI published the multimodal CLIP model that could predict the similarity between images and text. The model demonstrated remarkable capability on a variety of tasks, most notably zero-shot classification, where it was able to predict classes and categories not present in the training dataset. This surpasses the capabilities of any previous architecture, and remains state-of-the-art (SOTA) today. Previous models focused on a narrow set of categories, and were ineffective at classifying images not represented in their datasets.

The success of the CLIP model can be attributed to the extensive range of images in CLIP's training dataset and the generality of its natural language processing (NLP) input. Consequently, this makes CLIP an ideal candidate for image retrieval, as further explored in this dissertation. To develop an efficient and effective NLP-based image retrieval system, I propose the integration of a CLIP model with an approximate k-nearest-neighbour (ANN) index, a well established method for efficiently searching for similar items in high-dimensional data.

Using these two components, I will show it is possible to create a fast and effective NLP-based image search system. I will compare different model architectures and assess the effectiveness of each one in terms of inference speed, model size and accuracy. Further, I will examine various ANN index algorithms to determine their respective strengths and weaknesses in terms of recall, build time, query time, and binary size.

The significance of these factors may vary depending on the specific production use case. The ImageNet DET object detection dataset will serve as the basis for evaluation of the proposed image retrieval system.

Chapter 2 will cover the history and research of vision models, the introduction of CLIP with zero-shot prompting. It will introduce a variety of ANN index algorithms and the dataset we will be using to evaluate our implementation.

My methodology in chapter 3 will explore the Vision Transformer and CLIP training process further, as well as the model and index factors that will affect the system's performance.

Chapter 4 introduces the implementation of the dual-stage image retrieval pipeline, consisting of building and querying stages. I will discuss the various hyperparameters that can be adjusted in the ANN indexes to give optimal performance for each of the metrics we are evaluating for. I will also discuss the integration of the dataset within the pipeline, and the generic implementation of multiple models and ANN index libraries using a wrapper and a public API interface.

Chapter 5 will statistically evaluate the model and index performance using the factors discussed in the methodology, comparing the performance between the Vision Transformer and ResNet architecture. I also assess the efficacy of each ANN index with respect to the image retrieval task.

Finally, chapter 6 will conclude the paper, and discuss potential improvements to my methodology and implementation, as well as limitations and potential areas of further research.

2 Literature Review

This literature review will provide a comprehensive overview of the history and SOTA methods for NLP-based image retrieval, examining the evolution of image classification methods, the introduction of the vision transformer and the development of multimodal image-text models. Further, it explores a variety of approximate nearest neighbour algorithms, and evaluates their effectiveness for this application. Finally, the literature review will provide an overview of the ImageNet DET Object Detection dataset, used to evaluate the performance of the system.

2.1 Development of Computer Vision Models

Convolutional Neural Networks (CNN) Previous to the invention of the transformer [1], SOTA computer vision models used CNNs. First introduced in 1998, Lenet-5 [2] utilised a mathematical operation called a “convolution” to detect different local features of an image, such as edges, corners and textures. This is the process of multiplying a small matrix (filter) with part of the image, and summing the result. The filter slides over the image producing a feature map. A neural network can use this These features are processed through a fully connected linear layer network to classify features into a set of classes.

In 2012, CNNs proved their effectiveness in image recognition tasks with AlexNet [3], the first CNN to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [4]. This competition was the benchmark for computer vision models at the time.

Residual Networks In a conventional CNN, layers are stacked sequentially, with each layer learning a representation of the input data. As the depth of the network increases, the vanishing gradient problem [32] arises, where the gradient of the loss function with respect to the network weights becomes very small, the network will only update the weights by a minimal increment, causing the network to not train effectively. Residual networks (ResNets) [34] mitigate the issue by adding “skip” connections that bypass certain layers. These skip connections effectively simplify the network, reducing the impact of vanishing gradients by allowing backpropagation to flow uninterrupted from the output to the input through these skip connections.

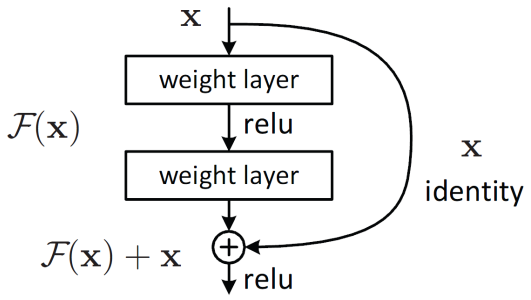

Figure 2.1: *Residual "skip" connection. Taken from [47]*

In this paper, ResNets are described in the form: RN50x4. This denotes that it is using the Resnet-50 architecture, 1 input layer, 48 convolution layers, and 1 output layer, scaled up 4x according to the EfficientNet [33] scaling rule.

Vision Transformer (ViT) [5] Vision transformers (ViT) have recently emerged as a powerful method to capture semantic information in images. By 2022, vanilla ViTs had outperformed SOTA ResNets in both accuracy [33] and efficiency [39]. One drawback of ViTs is the considerably longer training time compared to CNNs, as the convolution operation is specifically designed to capture local patterns, giving it an inherent inductive bias towards spatial locality in an image. ViTs lack this bias and must learn to recognise local patterns and hierarchical relationships from the data itself. As a consequence ViT models must be pre-trained on a significantly larger dataset than their CNN counterparts. The ViT architecture is further discussed in Section 3.1 of the Methodology.

Throughout this paper I will use the abbreviation ViT-B/32 to describe ViT models. This notation indicates that the model is using the ViT of base model size, referring to the number of parameters, with the /32 denoting the model uses 32 patches.

CLIP In 2021, OpenAI released the CLIP (Contrastive Language-Image Pre-Training) model [6], which changed how we thought of visual recognition. Prior to the advent of CLIP, a SOTA vision model would predict a fixed set of predetermined object categories. In contrast, the CLIP paper introduced a multimodal NLP approach that allowed the user to query an image with *anything* that can be described in a text box. This flexible approach performed well in a variety of tasks, most notably zero-shot classification, predicting a class seen zero times in training data. CLIP was designed to compare natural language directly with images by a dual encoder trained on 400,000,000 image-text pairs.

By predicting the most relevant caption associated with an image, CLIP learned to predict the similarity between an image and an arbitrary piece of text. CLIP has been trained with both the ResNet and ViT architecture, however in modern applications the ViT architecture has proven to be more effective. This dissertation aims to provide a comprehensive comparison of both architectures.

CLIP Retrieval The project clip-retrieval [7] is a production grade implementation of NLP search on an ANN database. It was developed by the LAION group and used to filter the Laion-5B dataset [8]. The project uses the autofaiss library [9] for indexing and a variety of CLIP language models from OpenCLIP [35].

Leaner and Faster [25] This paper described a CLIP image-text encoder in which the training was divided into a two-stage framework, training the image and text encoder separately, freezing the training of the text encoder while training the image encoder, and vice versa. The paper used a ViT-S/16 to generate an image embedding, with TinyBert [36] for text, using knowledge distillation [26] from the CLIP ViT-B/32 model to achieve better performance. This paper claimed a 61% reduction in model size, and up to 1.6x/2.9x image/text inference speedup over the original OpenAI ViT-B/32 model, with similar accuracy.

This technique allowed the model to be split into separate image and text encoder models, allowing the models to be loaded separately, limiting memory usage. Using the image encoder, we can generate image embeddings to insert into the ANN index, and query using the text encoder. For the duration of this paper, I will use the abbreviation “LaF” to refer to this model.

2.2 Development of k-Nearest-Neighbour Indexes

Approximate k-nearest-neighbour (ANN) indexes are a popular indexing method for vector retrieval that can be used to efficiently search large high-dimensional datasets. They address a pervasive challenge in high-dimensional data analysis known as the Curse of Dimensionality [11, 12]. First coined by Richard E. Bellman, when discussing the difficulties of using a brute force grid search to optimise a function with too many input variables, it describes the problem of analysing data with a large number of features. As dimensionality increases, distances between data points tend to converge, tending to all become equidistant from each other, complicating the process of clustering or analysing the data.

In the context of NLP-based image retrieval, ANN indexes can be used to index image embeddings, providing a fast and efficient method for retrieving relevant images based on NLP queries. ANN algorithms employ a variety of techniques to search for data points in high-dimensional data:

Linear Search [13] Comparing the query vector to every other vector in the index. This is an exhaustive k-nearest-neighbour search and has perfect recall, but requires linear query time over the size of the dataset. This is also known as a "Flat index".

Grid Search [13] Subdivide the search space into a grid, where each cell c_i in the grid representing the space closest to point i. This requires exponential space and time in the dimensionality of the dataset, and so is unviable for our high-dimensional use case.

Locality Sensitive Hashing [11] This involves applying multiple hash functions to map data points into buckets, with the objective of ensuring that data points in close proximity are placed in the same bucket, effectively maximising hash collisions [37] of similar vectors. To search the index, the query is hashed to locate the bucket containing the closest points. This gives theoretical guarantees of sub-linear query time, but comes at the cost of high memory usage.

Spotify Annoy Library [16] A tree based algorithm, that constructs a binary tree using random projection [17], subdividing the search space in two randomly, recursively for a given number of trees. Random projection can be used to approximate cosine distance [18, 48]. This is repeated multiple times to create a forest of many trees, mitigating some of the randomness splitting the search space randomly may cause. When querying the index, each of the trees in the forest undergoes binary search to its leaf, and the distances from the query point to each point in the index space is calculated exhaustively.

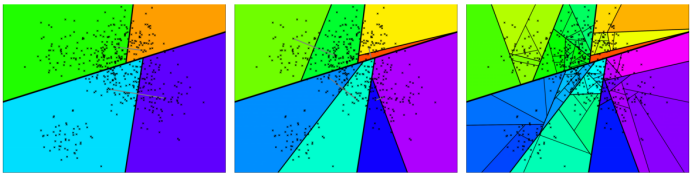

Figure 2.3: *Illustration of random projections in two-dimensional space. Taken from [48]*

Annoy has several notable features: it uses static files as indexes, allowing the creation of indexes to be decoupled from loading them. However, because of the static nature, once an index has been built no more data can be added to it. The Annoy indexes can be rapidly loaded and mapped into memory, only requiring a mmap [20] operation on the CPU. Additionally the index claims to have a smaller memory footprint compared to ANNs discussed later in this paper.

Annoy is designed for dense, low dimensional data (<100 dimensions) in contrast to other ANNs, which cater to sparse data and are fairly agnostic to dimensionality. This may prove to be a disadvantage for the index, as our models use a minimum of 512 dimensions, and could lead to Annoy using more memory and incur a larger runtime cost.

Hierarchical Navigable Small World Graphs (HNSW) [21, 22] is a graph-based algorithm, where vertices are linked based on proximity with each other. This approximation is built around a probability skip list [23], a sorted linked list that allows for efficient insertion and querying. A skip list constructs layers, where the top layers contain few vertices far apart from each other to allow fast navigation across the graph. When we reach the first node larger than the query on one layer, we move to the layer below. As we descend the layers, the precision of the list increases until the bottom layer, which is the entire linked list. Each node is given a random number which dictates how many layers the node will occupy. This makes the list a probabilistic data structure, with an average time complexity of $O(log\ n)$ for search and insertion.

The HNSW graph uses this concept, but instead of a sorted linked list, uses a proximity graph. At the top layer, the graph is very low-degree. At the bottom layer, the graph has a very high-degree. We use a greedy search algorithm [24] to search the HNSW graph. We begin at an entry point, and choose the vertex closest to the query vector, until we find a vertex with no closer vertices. When no closer vertices are found, we descend to the next layer.

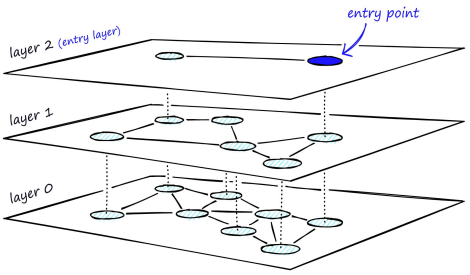

Figure 2.3: *Illustration of layers of a HNSW index. Taken from [50]*

HNSW has the advantage of batch querying, where multiple queries can be run at the same time. However, HNSW graphs generally have high memory usage. This type of index is advantageous in highly clustered embeddings, a characteristic of many real world datasets.

2.3 ImageNet DET Object Detection Dataset

I use the ImageNet DET Object Detection dataset [4] to evaluate the performance of the CLIP models and indexes. The dataset consists of a diverse collection of 200 basic-level categories, such as "dog" or "plane." The dataset labels are constructed upon the WordNet synset [31], a lexical database for English, where related concepts are arranged in a hierarchical tree fashion. The DET Object Detection dataset is distinct from the more conventional ImageNet 21K dataset [38], containing over 21,000 categories. I chose this dataset as my initial intention was to use an object detector paired with the CLIP model, however this no longer in the scope of this paper

2.4 Summary

This literature review provided an overview of the history and SOTA methods for NLP-based image retrieval. We observed that the development of image classification models and the introduction of the vision transformer allowed for more efficient and effective processing of images, outperforming ResNets, the previous SOTA. Additionally, the release of the CLIP model marked a turning point in zero-shot visual recognition, and proved more flexible through a multimodal natural language approach. The development of the clip-retrieval project by the LAION group shows the practical application of NLP search on an ANN database and its potential for large-scale image retrieval.

Finally, this research evaluated multiple ANN algorithms to explore the feasibility of employing ANN indexes with a CLIP model to enhance image retrieval in NLP applications

3 Methodology

In this paper I will evaluate the efficacy of using pretrained CLIP models and ANN indexes in the development of a robust image retrieval system. The evaluation will involve a comparative analysis of the ResNet and ViT with respect to inference time and accuracy, and use precision, recall with the F1 score to determine an optimal binary classification threshold.

Furthermore, I will examine the performance of Annoy [16], HNSW [21, 22] and Flat L2 [13] indexes in terms of recall, build time, query time and binary size. By statistically adjusting the hyperparameters of each index, I hope to achieve an optimal configuration for each metric. Additionally, I will assess the suitability of these indexes for the efficiency and scalability required for an image retrieval system. I will use the PyTorch [52] library to evaluate the models in this paper, and the Annoy and Faiss [51] libraries to evaluate the indexes.

A limitation of this study is that I will only use a single dataset for evaluation, ImageNet DET Object detection. The performance of the models may vary on the dataset used, and index implementations may be effective on datasets of different sizes. We will discuss the implications of this further in the conclusion.

3.1 Vision Transformer Architecture

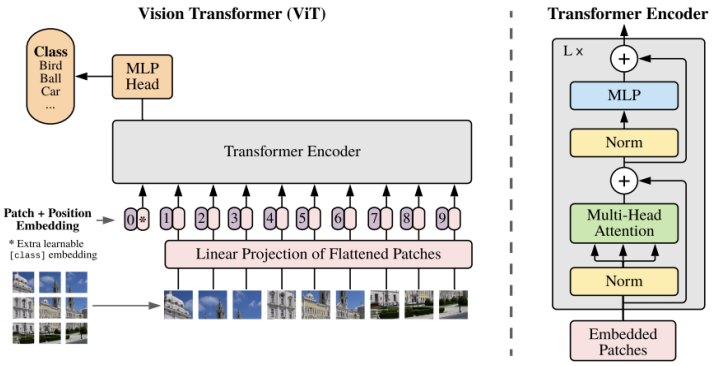

Figure 3.1: *Vision Transformer model overview. Diagram from [4]*

The ViT model transforms an image into a sequence of non-overlapping patches and linearly embed these into a flat vector. The patch embeddings are fed into the transformer architecture [1] with a positional embedding index [1], which captures long-range dependencies between the patches using multi-head self-attention.

3.1.1 Multi-Head Self-Attention

$$Attention(Q,\ K,\ V)\ = softmax(\frac{QK^T}{\sqrt{d_k}})V \qquad \text{(Equation 1)}$$

Self-attention leverages three learnable matrices: Query (Q), Key (K) and Value (V), derived by multiplying the input tokens with the respective weight matrices W_Q, W_K, W_V. The self-attention mechanism is shown in Equation 1. QK^T computes the compatibility between each pair of tokens in the input sequence. The higher the dot product, the more related the tokens are. Softmax converts compatibility into attention weights, summing to 1, where $\sqrt{d_k}$ scales the values to prevent large values from dominating the softmax and causing vanishing gradients [32]. V aggregates the information from each of the input tokens with the relative importance from the softmax. The Transformer further enhances self-attention with multi-head attention, consisting of multiple self-attention layers using distinct weight matrices. The outputs are concatenated and linearly transformed to produce the final multi-head attention output. The Multi-Head attention can be seen in the Transformer Encoder in Figure 3.1.

The title of the paper introducing ViTs, “An Image is Worth 16x16 Words” [5], is referencing the idea that an image can be divided into 16x16 patches, where each patch creates a token embedding in the same way a word would be used in a text-based transformer.

3.2 CLIP Training Process

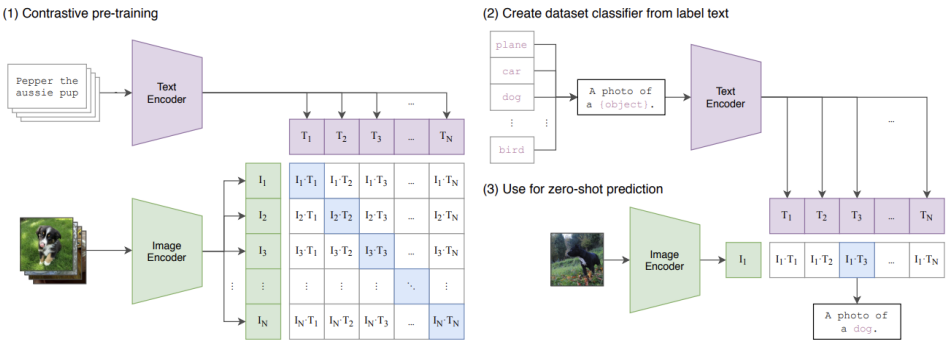

Figure 3.2: *Summary of CLIP training. Diagram from [6]*

I compare pretrained CLIP models in this paper, trained using a contrastive loss function, this maximises the similarity between representations of image-text pairs, while minimising the similarity with non-corresponding pairs. The models were trained using the Adam optimizer for 32 epochs on over 400,000,000 image-text pairs.

Parts 2 & 3 of Figure 3.2 show the evaluation of the CLIP models with zero-shot prediction. I will be using the same method to evaluate the models, but with a conjoining ANN index. For each image, I will compare the embedding with every label in the dataset, and pick the top k elements with the smallest distance.

While the original CLIP paper uses cosine similarity to measure the similarity of text and the image embeddings, I will be using angular distance for comparison for the duration of this paper. These are fundamentally equivalent metrics ($angdist = arccos(cossim)/\boldsymbol{\pi}$). I use angular distance primarily as it is a default distance metric in Annoy, and used it for subsequent indexes for comparison.

3.3 CLIP Model Factors

This study tests variants of CLIP ViT and ResNet models, including a smaller derivative "Leaner and Faster" (LaF), distilled from the original CLIP ViT-B/32 model. I compare the CLIP ViT-B/16 and /32 model to test the impact of reducing the number of patches. Further I evaluate the ResNet models RN50 and RN50x4, to assess the effect of EfficientNet scaling laws [33].

I anticipate that the CLIP ViT-L/14 model, with a larger number of parameters and a smaller patch size to outperform the other models in accuracy, but also run the slowest. Factors that may affect the models are as follows:

Inference time The time required to create image embeddings primarily depends on the number of parameters used in the model, and the patch size for ViT. The LaF model, being a ViT-S/16, claims a 1.6x faster image inference time compared to the original CLIP ViT-B/32 model. Conversely, the CLIP ViT-L/14 model, with a larger number of parameters and smaller patch size, may require a longer inference time due to increased complexity. From literature review research, ViT's should be more computationally efficient and accurate than ResNets [33, 39], so I expect the base ViTs to outperform the ResNet models.

Memory usage The memory consumption of the models is influenced by the number of parameters and patch size for ViT models. I expect the LaF model, with its reduced size, to consume less memory compared to the ViT-B/32 model. The ViT-L/14 model may require more memory due to the increased number of parameters and smaller patch size.

ResNets require less memory than the ViT models, as ViTs are required to store all pairs of self-attention tokens during inference, while the ResNets need only to store localised areas of the image in convolutions. Further, I expect the ViT-B/32 to require more memory than ViT-B/16, as the increased number of patches requires an increased number of inputs to the transformers self-attention, which scales memory quadratically with number of inputs. Unfortunately due to time constraints I will not evaluate memory usage in this study.

Accuracy Generally a trade-off with inference time and memory usage. The LaF model may exhibit a lower accuracy compared to the CLIP ViT-B/32 model due to its smaller size. By comparing ViT-B/32 and ViT/16, we will be able to quantify the difference of a smaller patch size.

The CLIP ViT-L/14 model, with its larger number of parameters and smaller patch size, is expected to provide a higher accuracy, representing the upper bound on the performance of the retrieval project. I expect that ViTs in general to outperform ResNets, although a larger ResNet model may be able to achieve higher accuracy than a smaller ViT model.

Dimensionality A model's dimensionality can affect the recall of the ANN index. For instance, the Annoy index prefers lower dimensional dense embeddings, while the HNSW index performs better with embeddings that can be clustered easily. The LaF and ViT-B model, with its lower dimensionality, may be more suitable for the Annoy index.

In contrast, the CLIP ViT-L/14 model has a higher dimensional embedding which may be more easily clustered, benefiting the HNSW index. Additionally, some index implementations store only non-zero features, meaning that sparse high-dimensional data can result in smaller index sizes.

3.4 Approximate Nearest Neighbour Index Factors

Various ANN algorithms have been developed to address a range of factors. Choosing an appropriate index requires evaluating the significance of each factor for the specified use case and determining trade-offs one is willing to make. Factors include:

Build time Index creation and build time are crucial if there are few queries compared to the size of the index. If this is the case, many of the ANN algorithms will take longer to build the index than to query.

Load time The speed in which an index can be loaded from disk to RAM, important if it is necessary to split a dataset into multiple smaller indexes.

Search time For a use case with a large volume of queries, search time could be essential. This often comes as a penalty to build time and recall.

Hardware resources Considerations include memory usage during both the build and querying process, as well as whether the index can be split up or otherwise lazily loaded. The size of the index when saved to disk could also be taken into account. Finally, some libraries can take advantage of GPUs, while others are designed for single or multi-core CPUs

Recall requirements Many of the algorithms used in this paper provide approximations of distances. If perfect recall is essential for an index, a Flat index should be considered.

Access patterns Some indexes support batch querying, while others may be able to share the index across processes. An important consideration should be to determine if an index can be incrementally added to, or remain fixed once built, which may suit different use cases. Finally, some indexes have preferences for different data dimensionalities. Our CLIP embeddings are high dimensional, which may be a disadvantage to certain indexes such as Annoy.

4 Implementation

My implementation consists of two stages. The first uses a CLIP model to generate vector embeddings for each image in the dataset, with each image embedding being inserted into the ANN index and a database for efficient retrieval. In the second stage, a text query is encoded into a vector embedding, which is used to search for similar embeddings in the index.

4.1 Retrieval Pipeline

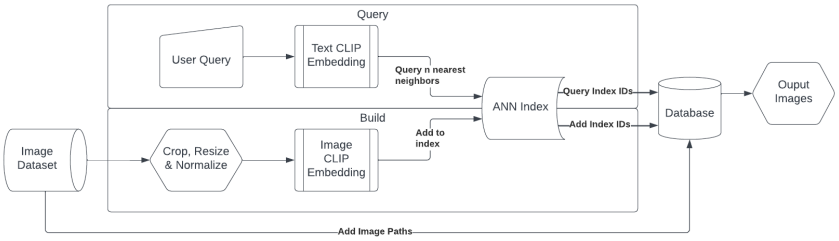

Figure 4.1: *Illustration of image retrieval pipeline*

The process of image retrieval comprises two distinct stages: index construction and querying. Although these stages are executed independently, optimising one often comes at the expense of the other. For example, an L2 Flat Index can be quickly constructed and provides exact matches, but has a slow query time as it requires linear scanning. On the other hand, HNSW delivers accurate results and enables for fast queries, but has a longer index construction time.

4.1.1 Index Construction

To prepare each image in the dataset for retrieval, they undergo cropping, resizing, and normalisation according to image encoder's requirements. The CLIP model transforms the processed image into a high-dimensional vector embedding, which is inserted into the ANN index. The image paths and IDs are stored in an SQLite database, enabling the retrieval of the original image when the ANN IDs are returned from the query.

4.1.2 Querying

During the query stage, the user's search query is encoded and utilised to retrieve the n most similar items from the index. The returned elements' index IDs are used to locate the original images from the SQLite database.

4.2 Model & Index Implementation

To compare multiple models and ANN libraries, I developed a wrapper around each library and model, with a common public interface. This approach enables the encapsulation of specific library features in the wrapper, taking a config object to adjust the hyperparameters of the model and index accordingly. This enables the image retrieval code to be written once while retaining specific implementations for various libraries.

An SQLite database is used to store complimentary image and index data. The indexes are stored externally, with references to the index paths in the database. The index construction and querying processes are performed as described in Sections 3.3.1 and 3.3.2. The SQL architecture is further discussed in Appendix 1.

4.3 ImageNet DET Object Detection Dataset Integration

I utilise the ImageNet DET Object Detection validation dataset to test the image retrieval system, which contains 20120 images of 200 different classes. I developed the "ImagenetDETDataset" class to parse the dataset, linking images to their respective labels. Each label is used to query the dataset with the following prompt: “[label]”.

This can be modified, if instead the dataset consisted of bird images, the prompt could instead be “A photo of a [label], a type of bird”. The integration of the ImageNet DET dataset into the retrieval implementation shows the system’s capability of processing large-scale, real-world images, serving as a robust benchmark for evaluating the performance of the ANN libraries and models.

As the CLIP image encoder is trained on images of size 244x244, I am required to resize and normalise the images into the correct format for the model.

The images can be sorted by angular distance, which is a measure of the predicted similarity of the model. Following this we can now evaluate top 1 and top 5 accuracy of the model, suitable angular distance threshold values, index recall and more.

4.4 Approximate Nearest Neighbour Implementations

In this project, I will evaluate the CLIP model embeddings across a variety of ANN implementations and libraries. My research will primarily focus on Spotify's Annoy library [16] and Faiss' [51] Flat L2 and HNSW index. I will attempt to adjust the hyperparameters for these indexes to achieve satisfactory results with respect to the factors discussed in Section 3.4 of the Methodology.

It is important to note that many of the factors are conflicting; therefore my aim is to find a balance supported by empirical evidence. Additionally, this analysis is not exhaustive, as numerous other index implementations and enhancements may be considered, such as vector encoding and quantization [40], Google's Scann library [41], Locality Sensitive Hashing [11], or using an Inverted File Index [42].

4.4.1 Annoy Index Hyperparameters

Annoy employs a number of specific hyperparameters that can impact its performance and recall. These include:

Number of trees Provided at build time, influences the recall, build time and the index size. A larger number of trees will yield more accurate results but also result in larger index sizes.

Search k The number of nodes to inspect. *Search k* offers a runtime trade-off between better recall and speed. A larger *search k* will produce more accurate results but will take longer to run.

Max results When including distances, the Annoy index must calculate distances to all the closest nodes. This parameter can affect the overall recall and search time.

Number of indexes The number of indexes used to store the dataset can have an impact on the performance of the Annoy index. A higher number of indexes may increase recall, as each index contains fewer elements, but can also slow down search speeds due to the switching between indexes and multiple tree searches. Conversely, a single large index containing all elements may increase search times but decrease the overall recall. This hyperparameter must be included as a design decision when creating the retrieval architecture. I will compare multiple sizes in my results along with the other hyperparameters.

4.4.2 Flat L2 Index Hyperparameters

This is a linear scan through the dataset. It has only a single hyperparameter:

Max results Increasing the max results leads to increase an in search time, as the flat index maintains a priority queue [43] for the closet k elements. When replacing an element, it must reorganise the queue, where a larger queue requires more computation.

4.4.3 HNSW Index Hyperparameters

Max results Maximum number of elements to return. In some instances there may not be enough elements to fill the results list, resulting in fewer elements than max results specified, unlike Annoy and Flat index. Max results increase search time due to the increased number of distance calculations and the priority queue management, similar to the Annoy and Flat indexes.

M Maximum number of neighbours per layer. Larger values of *M* lead to a denser graph, resulting in faster search times and higher recall, but increased memory usage.

efConstruction Size of dynamic list used to store best candidates during index construction. Larger values lead to a more accurate and connected graph, increasing recall, but comes at the cost of build times and memory usage.

efSearch Determines size of dynamic list to store best candidates found so far. Larger values increase recall, but at the cost of search time.

5 Results

5.1 Evaluating CLIP Inference

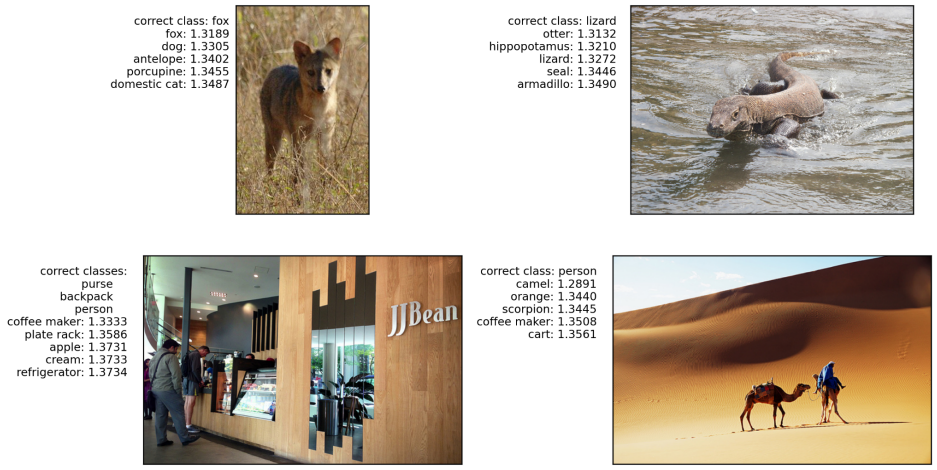

Figure 5.1: *Top 5 Angular Distances of ViT-B/16 Image embeddings to each of the 200 ImageNet labels.*

The top left image of Figure 5.1 demonstrates the correct classification of a fox. The other classifications are reasonable, as they all represent mammals that might inhabit the same environment. However, the alternate classifications have significantly larger angular distances than the correct classification.

The top right image showcases the contrastive similarity capabilities of CLIP. CLIP is trained to bring together similar images and text, while distancing those that are dissimilar. In this example, a mammal in the water is considered similar to aquatic animals: “otter”, “hippopotamus” and “seal” were all positively identified. The correct class “lizard” was also included in the top 5.

The bottom left image presents an incorrect classification. However, it is important to note that 4 of the top 5 embeddings had angular distances 1.35 and above, which would exceed the classification threshold. While no coffee maker is in the scene, the subjects are queuing in a coffee shop (JJBean is a coffee chain Vancouver). CLIP can sometimes infer contextual information about the scene, even when it is not visually apparent. This is further demonstrated in Appendix 6. This image has another phenomenon of the CLIP embeddings. They very rarely give a high similarity with the “person” label, I believe because of the vague wording.

Upon manual inspection of the misclassified images, I found a considerable number of them were inadequately labelled. In the bottom right image, the only label is “person”. CLIP correctly identifies the camel in the image with high similarity, which is classed as an incorrect classification.

5.2 Evaluating CLIP Model Performance

Model	Top 1	Top 5	Inference time (s)	Model size (MB)	Emb Dim
LaF	0.5662	0.7710	1370	229	512
RN-50	0.5982	0.8057	826	244	1024
RN-50x4	0.6333	0.8203	2623	402	640
ViT-B/32	0.6248	0.8180	705	338	512
ViT-B/16	0.6515	0.8355	2117	335	512
ViT-L/14	0.6657	0.8450	14230	890	714

Table 5.1: *CLIP model results on ImageNet DET Object Detection Validation Dataset.*

All results run on the CPU of a M1 Macbook Air. GPU results may vary. I consider factors from Section 3.3 of the Methodology.

Baseline Accuracy Top 1 and top 5 accuracy was assessed image-by-image, where the closest 1 & 5 images were selected for evaluation, irrespective of the relative distances. For images with more than 1 label, positively identifying one label is deemed correct.

ViT-L/14 demonstrated the best performance, with a top 1 accuracy of 66.57% and a top 5 of 84.50%. I anticipated this outcome as the model has the largest number of parameters and smallest patch size. Interestingly, ViT-B/16, performed significantly better than ViT-B/32, despite having a similar number of parameters, with results comparable to ViT-L/14.

Inference speed I used the total inference time to embed all 20,120 images in the ImageNet validation dataset. While ViT-L/14 demonstrates the highest accuracy, it also had the longest inference time. Comparing ViT-B/16 with ViT-B/32, we can evaluate the effect

patch size, given all other parameters are equivalent. The smaller patch size has a higher accuracy, but with 3x the runtime.

The LaF model was the most disappointing, claiming a 1.6x speedup on image embeddings. In my testing however, LaF underperformed ViT-B/32 by 1.94x in inference, while also giving a worse accuracy.

LaF utilises a ViT-S/16 mode and a smaller patch size which may have a larger performance hit on CPU than if run on a GPU. Unfortunately, LaF uses operations not yet supported by MPS, the MacOS GPU framework, so I cannot evaluate the inference time using a GPU.

Model size Model size serves as a useful heuristic for the number of parameters. It exhibits a linear relationship with the models in terms of inference speed and accuracy. As we expect, ViT-L is larger than ViT-B, and LaF is smallest at 229 MB, using a ViT-S and TinyBert [36].

Embedding Dimension This can be crucial for some use cases; for example, Annoy is optimised for low-dimensional data. For other indexes it is not a significant factor. The dimensionality of the embedding will also affect the optimal angular distance threshold.

5.2.1 ViT vs ResNet

ViT-B/32 and RN-50 have comparable inference speed, but ViT-B/32 is slightly faster and more accurate, albeit with a larger model size. The same holds true for ViT-B/16 and RN-50x4; ViT-B/16 is slightly faster and more accurate, while also having a smaller model size than RN-50x4. This supports the conclusion by other papers researched in the literature review, ViTs outperform ResNets in both accuracy and computational efficiency [33, 39].

Memory usage for each model was not evaluated in this paper, but may have been in ResNets favour, as explained in my preliminary analysis in 3.1 Method. This is an area for further research.

5.2.2 Angular Distance Threshold

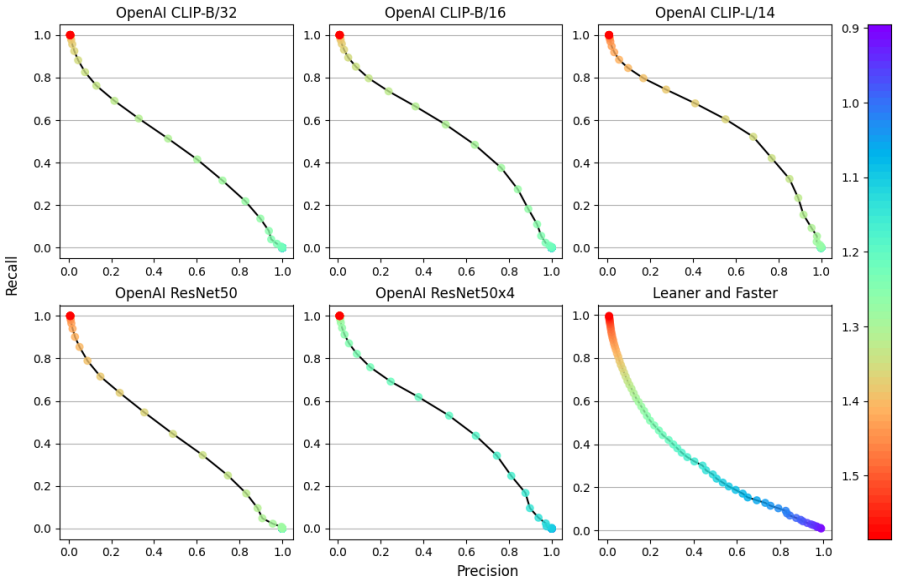

Figure 5.2: *ROC Graph w.r.t angular distance of CLIP models (up and to the right is better)*

Figure 5.2 displays the Receiver Operating Characteristic (ROC) graph of each of the CLIP models, illustrating the trade-off between precision and recall in the context of binary classification for determining the presence of a particular label within an image. Precision and recall represent conflicting objectives; optimising one comes at the cost of the other. ROC evaluates the model's ability to classify images *over a range of thresholds*.

Model	**AUC-ROC**
LaF	0.3001
RN50	0.4380
RN50x4	0.5053
ViT-B/32	0.4778
ViT-B/16	0.5443
ViT-L/14	0.5840

Table 5.2: *AUC-ROC of CLIP models*

Area Under the Curve of the ROC graph (AUC-ROC) can be used as a comprehensive performance metric, quantifying the performance across all threshold values. A higher AUC-ROC value signifies that the model exhibits superior performance without the need to select a specific threshold. The distribution follows the Top 1 & 5 accuracy, where a larger model size and fewer patches yield a higher score.

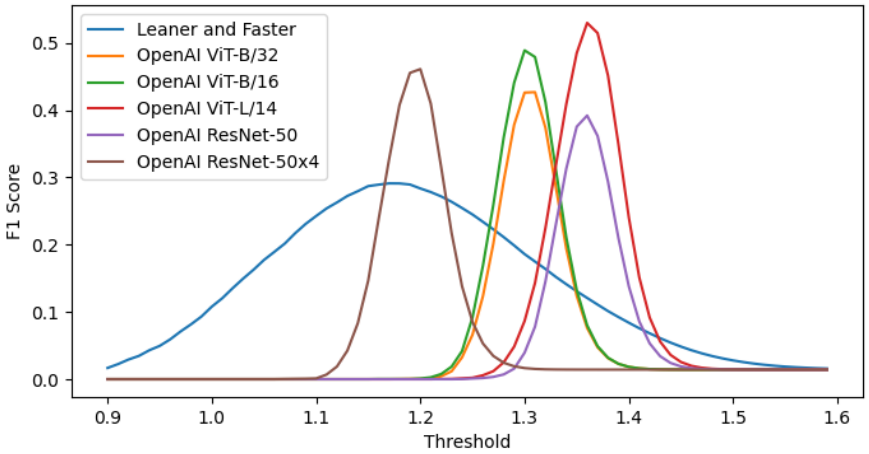

Figure 5.3: *F1 Score of CLIP models over angular distance threshold*

A related metric is the F1 score [45], which is the harmonic mean of precision and recall. This is closely related to ROC, but the goal of the F1 is to obtain an optimal threshold value that *strikes a balance between these two competing metrics*.

The OpenAI models have a tighter F1 distribution, meaning that the angular distances are dispersed over a smaller range. This can be advantageous as it provides a clear optimal classification threshold. I believe this is why LaF performed worse than expected on AUC-ROC (Table 5.2), with a convex curve on the ROC graph (Figure 5.2). Images distributed over a larger range offer more opportunity for incorrect classifications, while the OpenAI models can reject anything outside of the narrow range.

From the F1 graph, we can see how the models are distributed with their embedding dimension. Embeddings in a higher dimension will generally be farther away from each other due to the curse of dimensionality [11, 12]. We can see that the ViT-B models with their 512 dimensions have a peak angular distance threshold at a closer distance than ViT-L, with 712 dimensions. Conversely, RN50x4 has a closer distance threshold than RN50, with dimensions 640 and 1024 respectively.

5.3 Evaluating ANN Indexes

I used LaF's 512 dimensional image embeddings to evaluate the ANN indexes in this study. At the time I was under the impression it was a faster and more efficient implementation, but I now believe the distribution of embeddings over a larger angular distance range (Figure 5.3) may have a negative impact on the approximation algorithms of Annoy and HNSW indexes.

The performance of the indexes was evaluated on the following factors: recall, query times, load times and binary size, as specified in Section 3.4 of the Methodology. Query time refers to the cumulative time taken to perform all 200 class queries on the dataset of 20120 images.

5.3.1 Annoy Index Results

Annoy indexes are optimised for dense, low-dimensional vectors. 512 dimensional embeddings are relatively high-dimensional, however they are the lowest dimension of the embeddings I evaluated. I will evaluate the above factors according to the Annoy hyperparameters: max results and number of indexes. Unless specified otherwise, the number of trees is set to 1024. Search k defaults to the number of trees (1024) multiplied by the maximum results. To simplify evaluation, I will use maximum results to evaluate the effect of both parameters.

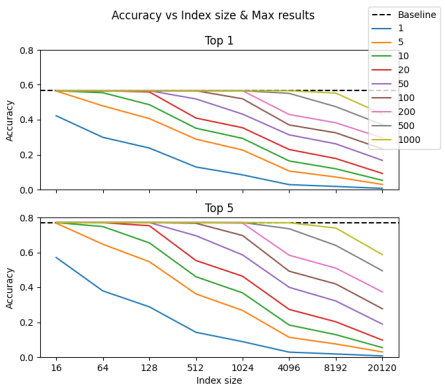

Figure 5.4: *Accuracy vs index size and max results.*

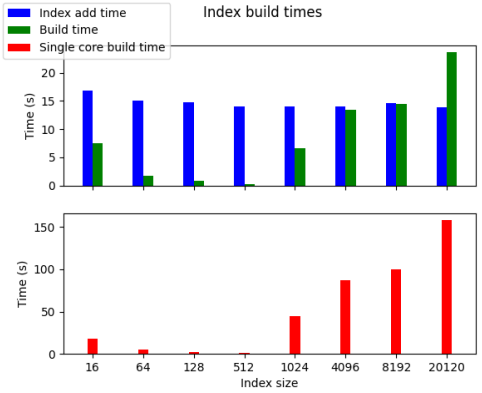

Figure 5.5: *Annoy build and vector add times vs index size.*

Recall Figure 5.4 demonstrates the effect of index size and max results have on accuracy. We can see that index size exhibits a negative correlation with recall, counteracted by max results, which

also increases search k. Post query each image is filtered for the top n closest embeddings.

When the number of max results is larger than the number of items in the index, the distance to every item in the index is calculated, giving it perfect recall, emulating a Flat index. This can be seen in Figure 5.4, where the maximum number of results is greater than the index size, the index has baseline accuracy.

Build Time Figure 5.5 shows the combined build times of all indexes for each index size. For an index size of 16, the total build time is longer than an index size of 64 as 16 requires more indexes. From this plot we can see that the index size that optimises build time is 512, possibly because the entire index can be stored in memory rather than lazily loaded (shown in Appendix 7).

I believe this is why indexes larger than 512 take significantly longer to build. We can also see that index size has no effect on the time to add each vector to the index. For an index size of 512, index add time dominates index build time.

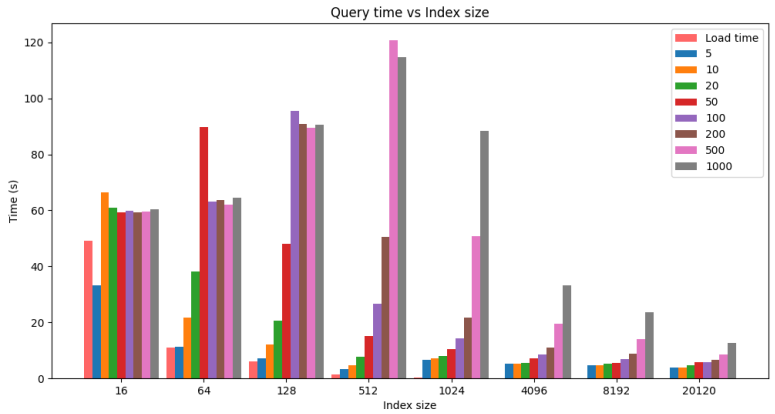

Figure 5.6: *Annoy query vs index size & max results*

Query Time Figure 5.6 shows that in general, a fewer number of large indexes query quicker than a large number of small indexes. As we expect, a larger number of max results slows down query time, as the index must calculate the distances for more elements in the index, due to the increase in search k.

The fastest query times were obtained from a single index of 20,120 elements, with query times from 5 to 12 seconds. The load times do not impact the performance at any but the smallest indexes, due to the accumulation of load times of many indexes.

A notable observation is that when the max results are close *but not quite* the same as index size, the query time is significantly longer. This is because it must return almost the entire index, but still sort some items out with binary tree search. When the max result is larger than the index size, it can return distances of every item in the index.

This is evident by the longest query time for an index size of 16 being a maximum result of 10, 64 being 50, 128 being 100, 512 being 500 and all subsequent sizes being 1000. I further explore single index query times in Appendix 2

In summary, an index size of 512 appears to be a suitable choice for balancing query and build times, using a maximum results in range of 20-50, to balance the tradeoff between speed and recall. These results may vary with the dimensionality of the data. A considerable gap in this study is that I did not evaluate the effects of the *number of trees* hyperparameter, which could further improve performance. Appendix 4 contains a preliminary test of the number of trees on query time, but unfortunately I did not have time to explore this further in this paper.

5.3.2 HNSW Index Results

In this section I evaluate the HNSW index and the impact of the hyperparameters *M* and the max results to return, to compare similarities with the Annoy index. I used the default hyperparameters given by Faiss [51] for *efConstruction* (40), and *efSearch* (16). Investigating the effects of adjusting these parameters could be a suitable area of further research.

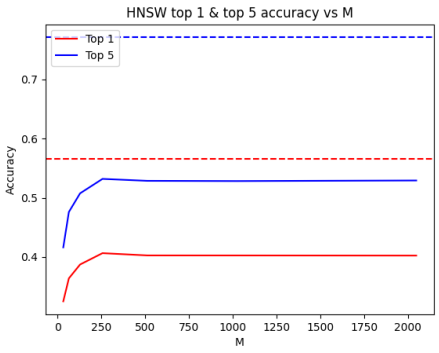

Figure 5.7: *M vs Accuracy*

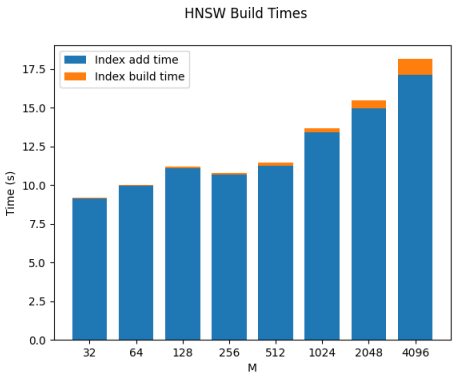

Figure 5.8: *HNSW Build Times*

Accuracy Figure 5.7 shows the recall of the HNSW index increases with *M* up to *M=250*, after which there is no significant improvement. Notably, the HNSW index exhibits remarkably low recall. Increasing *efConstruction* and *efSearch* from their defaults may help improve the results.

Build Time Figure 5.8 illustrates how the *M* hyperparameter affects build time. As *M* increases, the number of connections in each layer increases, leading to longer index add times. This also causes an increase in the final binary size (Appendix 3), which may have led to the increase in build times at larger values of *M*.

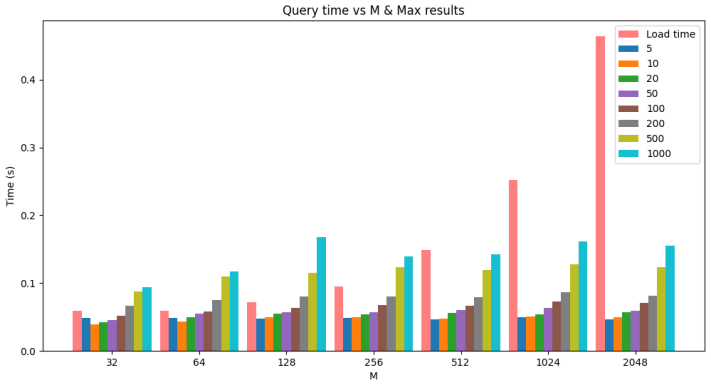

Figure 5.9: *HNSW query times vs M & Max results*

Query Time Figure 5.9 shows the impact of the *M* parameter on query times and the number of results to return. *M* has little effect on query time, but a large effect on load time, which could be due to the increased index binary size (Appendix 3). The maximum number of results to return increases query time, due to the larger priority queue, as explained in section 4.4.2.

My experiments showed that HNSW searches approximately 60 times faster than a single Annoy index, with somewhat comparable build times. The HNSW loads slightly slower, with a single Annoy index lazily loaded in 0.02 seconds, but this is not a significant factor due to Annoy's extremely long query time.

When running these tests it is clear that the *efConstruction* and *efSearch* hyperparameters must be further adjusted in order to achieve acceptable recall. This index may be useful for larger datasets but its current state renders it unviable.

5.3.3 Flat L2 Index Results

Statistic	Time (s)
Index add time	1.0630
Build time	0.0538

Table 5.3: *Flat L2 Index build times*

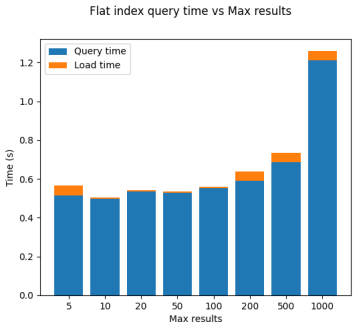

Figure 5.10: *Flat L2 index query times vs max results.*

I used a single index for the entire dataset, as it has perfect recall there would be no effect on changing the size of the index. A Flat index is equivalent to a list of tensors, and has over 10 times faster build times compared to Annoy and HNSW, shown in Table 5.3.

Figure 5.10 shows that a linear scan is performant over our dataset of 20,120 images. The index searches the dataset between 0.5 and 1.25 seconds, considerably faster than the Annoy index, but ~10x slower than the HNSW index. I expect the search times to scale linearly with the size of the dataset. Further research could explore the impact of an even larger dataset and how it affects the query performance.

5.4 Summary

My results demonstrate that ViTs perform better than ResNets on both inference speed and accuracy. I find ViT-B/32 to be a suitable tradeoff between the two factors. For a higher accuracy to speed tradeoff, ViT-B/16 can be used, which is 3 times slower, but has comparable Top-1 and Top-5 accuracy (1.42% and 0.95% respectively) to ViT-L/14 while being 6.7 times faster. These results were obtained on a CPU, so may vary when using a GPU or different hardware.

I found that CLIP embeddings under-classified certain labels, such as “person”. This could be improved with engineering more detailed prompts, such as “a photo of a [label]”.

When evaluating the ANN index performances the Annoy index was found to have significantly inferior performance to both HNSW and

Flat indexes. This outcome may be due to the high-dimensionality of the CLIP embeddings, which may not be suitable for tree-based search due to the curse of dimensionality [11, 12]. Increasing max results (and therefore search k) was found to improve recall, likely because distances to more elements were calculated, thereby approximating a Flat index. Alternatively, further finetuning of the hyperparameters *number of trees* may result in a better outcome. The unsuitability of the Annoy index is illustrated in the fact that the Flat index linear scan outperformed Annoy across build, load and query times, as well as binary size (Appendix 5) while providing perfect recall.

Index	**Query Time (s)**
Flat	0.00615
HNSW	0.0015
Annoy	0.06325

Table 5.4: *Single index time query time for 1000 results*

HNSW demonstrates the fastest search time, 4x faster than the Flat index. Annoy demonstrated a much slower search time, at 10x slower than the Flat linear scan, while also not having the same guarantees of perfect recall as the flat index.

For the 20,120 image dataset, I found the Flat index to be the most suitable choice. Although the HNSW index searches 4x faster than the Flat index, the build times are 10x slower. This tradeoff may be acceptable; however the HNSW index's recall performance is significantly worse compared to the perfect recall of the Flat index.

The *efConstruction* and *efSearch* hyperparameters may need to be further adjusted for the index to become viable. I believe the Flat index also performed well due to its simplicity and distinct lack of hyperparameters, which allowed it to perform well without any finetuning. Further research could include a range of dataset sizes to test how the indexes scale with the number of elements.

6 Conclusion

My experiments indicate that the LaF model did not live up to the expectations set in the literature review. The “Leaner and Faster” paper claimed 1.6x faster image inference over ViT-B/32, while my results show it to be 1.94x *slower*, while also giving worse accuracy.

For all my ANN index experiments I used the LaF embeddings. Unfortunately when evaluating LaF I found that the embeddings had a wider distribution of angular distances (Figure 5.3) compared to other models I evaluated. I believe this may have caused an unrealised impediment on the accuracy of the Annoy and HNSW indexes. If I were to do this again, I would use one of the official OpenAI model embeddings to test the indexes.

Due to time constraints, I could not comprehensively evaluate the *number of trees* hyperparameters for the Annoy index, instead choosing to focus on comparison with other libraries, namely Faiss with the HNSW and Flat index. Additionally, I did not fully evaluate the *efConstruction* and *efSearch* hyperparameters of the HNSW index, which may have contributed to why the index performed so poorly in recall. This research could be continued to further evaluate the effects of these parameters.

In my experiments the Flat index had the best overall performance. However, for larger datasets, I believe a Flat index may become unviable, necessitating the use of approximation algorithms such as HNSW. It was surprising that the Flat index remained effective with over 20,000 images. Further research would require a larger dataset, such as the training data for ImageNet DET which contains over 1,000,000 images. Evaluating the performance of each index over a range of dataset sizes could also lead to interesting results.

While I evaluated model and index binary size, a good heuristic for the number of parameters in a model, I did not assess the memory usage. Memory usage as a performance metric may serve as a valuable factor in evaluating performance. Annoy claims it has low memory usage compared to other indexes.

My analysis does not consider the impact of prompt engineering. Currently for each label the prompt is given as: “[label]”. A potential avenue of further research could involve whether different prompts lead to better results. In the original CLIP paper, the prompt “A photo

of [label]" was used, which may have yielded better results, particularly for under-classified prompts such as "person".

6.1 Architectural improvements

CLIP models are trained with both a variety of ViTs and ResNets, however the training method is generic and could be applied to any architecture capable of processing images and text. A notable new architecture is SWIN [44], which has been demonstrated to be more efficient by dynamically adapting to the input data using shifted windows. However, no publicly released model to date has applied the SWIN architecture to the CLIP training process. This could be an area of further research

An improvement to the indexing method could be using a vector database such as Weaviate [45]. These databases have recently become popular for storing vectors in a HNSW index with complimentary data, potentially offering more efficiency than my solution of an external ANN index paired with an SQLite database. However, my current approach enabled the comparison of different potential indexes and their hyperparameters, which may be more informative than a "black box" vector database.

6.2 Further research in other mediums

My implementation generalises well to other vision tasks, such as natural language search on video. Video retrieval essentially involves image retrieval over multiple frames. A potential implementation could use an object detector such as SSDLite [27, 28] RetinaNet [29] or the Yolov7 backbone [30], on the video stream to identify specific regions of a frame to create embeddings.

This would be a useful application for devices such as CCTV cameras, enabling dynamic searches over hours of video. These devices would likely be compute and memory constrained, so fast and small models and indexes are essential. A suitable choice would be the ViT-B/32 with a Flat index. The Flat index has extremely low build times, allowing multiple indexes to cover different time periods and enabling users to search only a subset of the embeddings. The ViT-B/32 had the fastest inference speed of any model, at 28.5 embeddings / second. This may be able to run real time in future applications. However it is important to note that inference was ran with more powerful hardware than that that would be available on an IoT device such as a CCTV camera.

7 Appendices

Appendix 1 SQL Schema

I first created this schema with video also in mind, but to use just on the image dataset we can safely ignore the `video` and `detection` tables.

When an index is created, a corresponding row is added to the `index` table. When an image is added to this index, it is first inserted into the `image` table, with its ID serving as a reference for future index-related operations.

The addition of an image to the index requires invoking the `add_item` function of the index's public API discussed in 4.2. This function appends the item to the index and returns a distinct local ID for the item within the index, known as the `index_ref_id`. It is crucial to note that this local ID may differ from the `image_id`. While some indexes may use these identifiers interchangeably, doing so may cause significant issues in others. For instance, when adding an item to an Annoy index, memory is allocated for `max(id) + 1` items. If an image with ID 1023 is added to an index of size 16, memory allocation for 1024 items will occur, thereby causing a substantial increase in the index size.

The `index_image` table serves as a bridge, connecting the `index_id` and an `index_ref_id` to an `image_id`. When saving an index, its filename is set to `index_id`, enabling the retrieval of the original `image_id` using the `index_id` and the `index_ref_id` returned by the query.

When the pipeline processes a video, the overall procedure remains mostly unchanged. The object detector identifies “interesting” sections of the video frames and adds these "detections" to the detection table. This table stores the `video_id`, the bounding box of the detection, and the detection timestamp. These detections are then added to the index and linked in the `index_image` table using the `detection_id`.

The final constraint to impose in this design is to ensure that either `image_id` or `detection_id` is set in the `index_image` table, but not both simultaneously. This constraint can be incorporated directly into the SQL schema using the following expression: `CHECK`

`(image_id IS NOT NULL XOR detection_id IS NOT NULL)`.

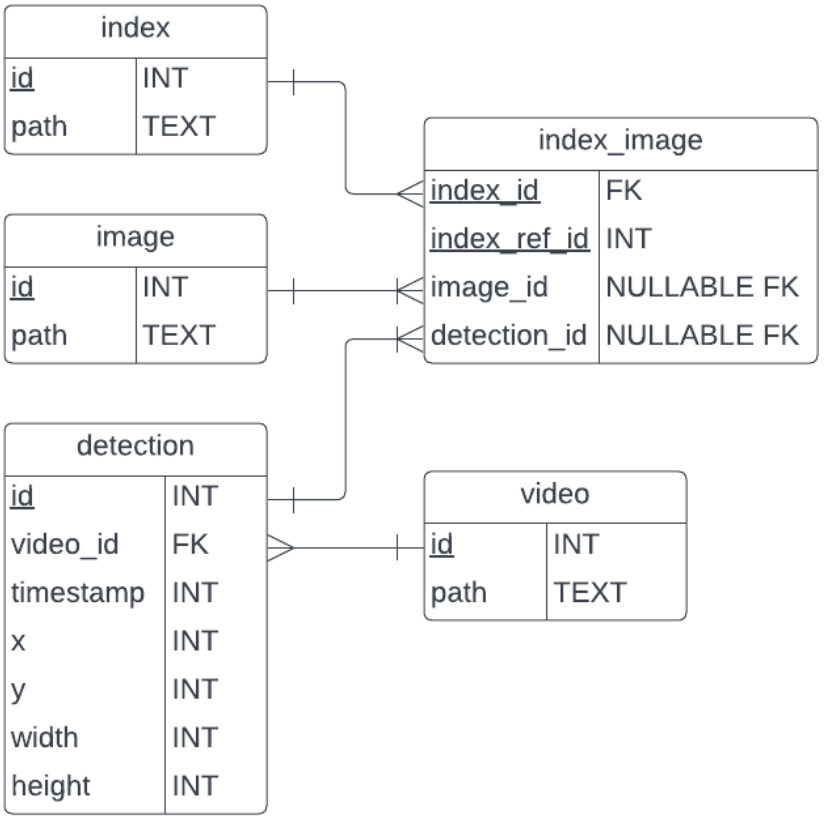

Figure 7.1: *Image retrieval SQL schema*

By adhering to this design principle, the retrieval pipeline can efficiently handle both image and video data while maintaining the integrity and consistency of the indexing process.

Appendix 2 Annoy Single Index Query Time

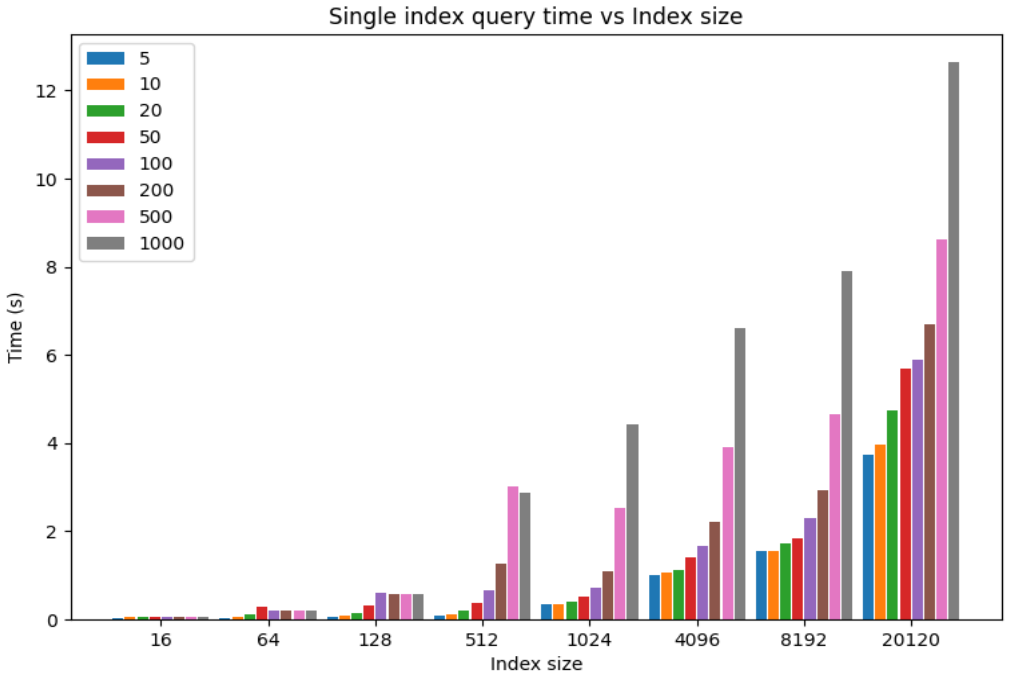

Figure 7.2: *Single index query times vs index size & max results*

While Figure 5.6 showed that a larger index resulted in a shorter query time, Figure 7.2 shows that a single large index takes longer to search than a smaller index. Therefore the smaller indexes take longer to search because of the larger number of them

Appendix 3 HNSW Binary Size

M	Index size (MB)
32	45
64	49
128	59
256	79
512	118
1024	197
2048	354

Table 7.1: *Effect of M on HNSW index size*

Table 7.1 shows how *M* affects the final index size in HNSW indexes. *M* is a hyperparameter that refers to the maximum number of neighbours per layer. As Table 6.1 shows, a larger value of *M* results in a larger index binary size.

Appendix 4 Number of Trees Preliminary Analysis

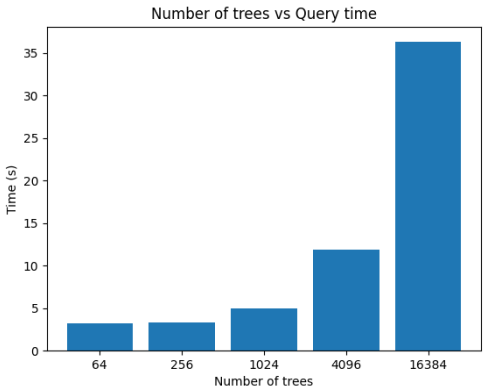

Figure 7.3: *Effect of number of trees on search time.*

Figure 7.3 shows the effect of the number of trees on search time. A larger number of trees requires more binary tree searches for each one, so increases the search time of the index.

Appendix 5 Comparing Binary size of the indexes

Index	**Size (MB)**
Annoy	299
Flat	39
HNSW (256)	79

Table 7.2: *Built index binary size*

Figure 7.2 shows the binary size of each index. As we expect the Flat index is the smallest due to its simple design. The Annoy index is a single index of all 20120 images, and is over 3x the size of the HNSW index.

Appendix 6 Analysing CLIP Embeddings

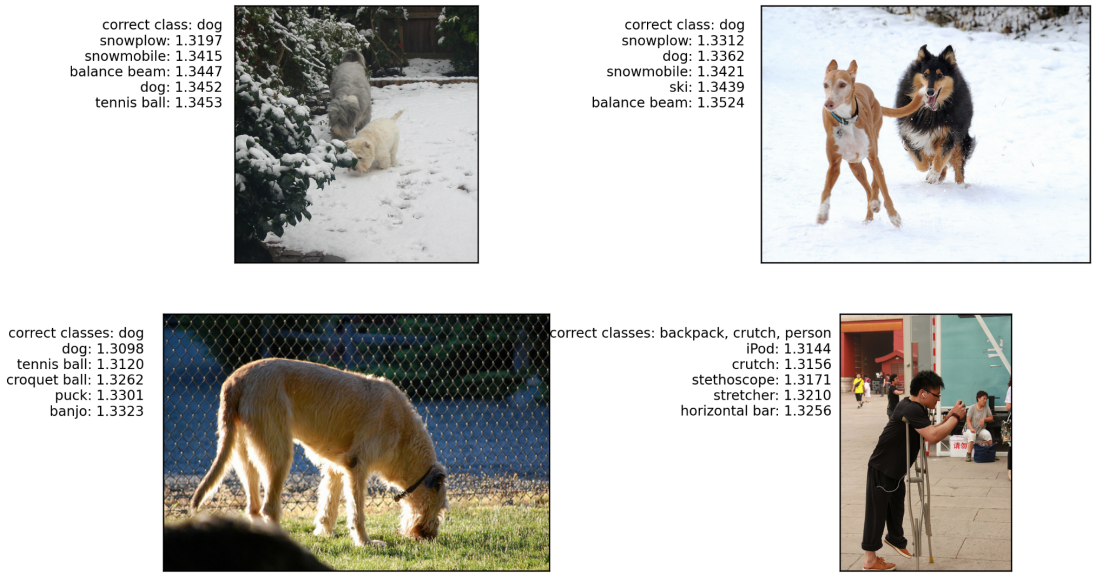

Figure 7.4: *Top 5 labels for each of the images (ViT-B/32), illustrating the effect of NLP in image retrieval.*

986/2777 (35.5%), of correctly classified dog images incorrectly classify the image as also containing a tennis ball in the top 5. This is because CLIP largely associates tennis balls with dogs, so even if the image does not contain a ball, the similarity of a tennis ball embedding to an image of a dog is enough to get it in the top 5 images. Figure 6.4 illustrates this in the bottom left image.

The top 2 images of Figure 7.4 further show this phenomenon. Dogs in the snow identifies highly with “snowplow” and “snowmobile”, even though there’s clearly neither in the photo, because these labels are similar to the snow in the picture.

Finally as you can see in the bottom right image, the model finds a high similarity with iPod, even though it is not visible in the scene, but the model infers this from the device being hidden in his hand and the headphones he has on. Further, the stethoscope and stretcher are very similar in this image, while again clearly not in the scene, as they have a high similarity to the crutches. Again the “person” tag is not identified, even though *clearly* in the image.

This illustrates the importance of prompt engineering, CLIP performs much better with specific descriptions.

Appendix 7 Annoy Single Index Load Time

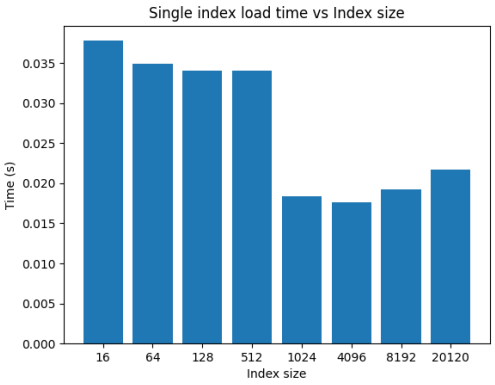

Figure 7.5: *Single index load time*

Figure 7.5 shows that the index load time is constant from index sizes 16-512 and 1024-20120. I believe this is because at 1024 the index is lazily loaded, which further reduces the initial load times.

Bibliography

[1] A. Vaswani et al., ‘Attention Is All You Need’. arXiv preprint arXiv:1706.0376. 2017

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ‘ImageNet Classification with Deep Convolutional Neural Networks’, in Advances in Neural Information Processing Systems, 2012, vol. 25.

[4] O. Russakovsky *et al.*, ‘ImageNet Large Scale Visual Recognition Challenge’. arXiv preprint arXiv:1409.0575, 2014.

[5] A. Dosovitskiy et al., ‘An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale’. arXiv preprint arXiv:2010.11929, 2020.

[6] A. Radford et al., ‘Learning Transferable Visual Models From Natural Language Supervision’. arXiv preprint arXiv:2103.00020, 2021.

[7] R. Beaumont, ‘Clip Retrieval: Easily compute clip embeddings and build a clip retrieval system with them’, GitHub repository. GitHub, 2022.

[8] C. Schuhmann et al., ‘LAION-5B: An open large-scale dataset for training next generation image-text models’. arXiv preprin: arXiv:2210.08402, 2022.

[9] Criteo, ‘autofaiss: Automatically create Faiss knn indices with the most optimal similarity search parameters.’, GitHub repository. GitHub, 2023

[11] P. Indyk and R. Motwani, ‘Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality’, Conference Proceedings of the Annual ACM Symposium on Theory of Computing, pp. 604–613, 2000.

[12] R. Bellman, Dynamic Programming, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[13] P. D. Cunningham and S. J. Delany, ‘k-Nearest Neighbour Classifiers - A Tutorial’, ACM Computing Surveys, vol. 54, no. 6, pp. 1–25, Jul. 2021.

[16] E. Bernardsson, ‘Approximate Nearest Neighbors Oh Yeah’, GitHub repository. GitHub, 2023.

[17] D. Yan, Y. Wang, J. Wang, H. Wang, and Z. Li, ‘K-nearest Neighbor Search by Random Projection Forests’, in 2018 IEEE International Conference on Big Data (Big Data), 2018.

[20] M. Kerrisk, "The Linux Programming Interface," No Starch Press, 2010.

[21] Y. A. Malkov and D. A. Yashunin, ‘Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs’. arXiv preprint arXiv:1603.09320, 2016.
[22] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, ‘Scalable Distributed Algorithm for Approximate Nearest Neighbor Search Problem in High Dimensional General Metric Spaces’, 08 2012, pp. 132–147.
[23] W. Pugh, ‘Skip Lists: A Probabilistic Alternative to Balanced Trees’, Commun. ACM, vol. 33, no. 6, pp. 668–676, Jun. 1990.
[24] Paul E. Black, "greedy algorithm", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 2 February 2005. (accessed 2/5/2023) Available from: https://www.nist.gov/dads/HTML/greedyalgo.html
[25] S. Ren and K. Q. Zhu, ‘Leaner and Faster: Two-Stage Model Compression for Lightweight Text-Image Retrieval’. arXiv preprint arXiv:2204.13913, 2022.
[26] G. Hinton, O. Vinyals, and J. Dean, ‘Distilling the Knowledge in a Neural Network’. arXiv preprint arXiv:1503.02531, 2015.
[27] W. Liu et al., ‘SSD: Single Shot MultiBox Detector’, in Computer Vision - ECCV 2016, Springer International Publishing, 2016, pp. 21–37.
[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’, 2018.
[29] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, ‘Focal Loss for Dense Object Detection’. arXiv preprint arXiv:1708.02002, 2017.
[30] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, ‘YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors’. arXiv preprint arXiv:2207.02696, 2022.
[31] Princeton University, “WordNet | A Lexical Database for English,” Princeton.edu, 2019. https://wordnet.princeton.edu/
[32] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, ‘Deep Networks with Stochastic Depth’, Lecture Notes in Computer Science, pp. 646–661, 2016.
[33] M. Tan and Q. V. Le, ‘EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks’, arXiv preprint arXiv:1905.11946. 2020.
[32] K. He, X. Zhang, S. Ren, and J. Sun, ‘Deep Residual Learning for Image Recognition’, arXiv preprint arXiv:1512.03385 . 2015.
[33] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, ‘Scaling Vision Transformers’, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 12104–12113.

[34] J. Deng, R. Socher, L. Fei-Fei, W. Dong, K. Li, and L.-J. Li, ‘ImageNet: A large-scale hierarchical image database’, in 2009 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), June 2009, pp. 248–255.
[35] G. Ilharco et al., OpenCLIP. Zenodo, 2021.
[36] X. Jiao et al., ‘TinyBERT: Distilling BERT for Natural Language Understanding’, arXiv preprint arXiv:1909.10351 . 2020.
[37] T. Cormen ‘Introduction to Algorithms’, MIT Press, p. 253, 2009.
[38] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, ‘ImageNet-21K Pretraining for the Masses’, arXiv preprint arXiv:2104.10972. 2021.
[39] X. Chen, C.-J. Hsieh, and B. Gong, ‘When Vision Transformers Outperform ResNets without Pre-training or Strong Data Augmentations’, arXiv preprint arXiv:2106.01548. 2022.
[40] S. Liu and H. Lu, ‘Learning Better Encoding for Approximate Nearest Neighbor Search with Dictionary Annealing’, arXiv preprint arXiv:1507.01442. 2015.
[41] R. Guo et al., ‘Accelerating Large-Scale Inference with Anisotropic Vector Quantization’, in International Conference on Machine Learning, 2020.
[42] E. S. de Moura and M. A. Cristo, ‘Inverted Files’, in Encyclopedia of Database Systems, L. Liu and M. T. Özsu, Eds. Boston, MA: Springer US, 2009, pp. 1571–1574.
[43] G. Brodal, ‘A Survey on Priority Queues’, Jan 2013.
[44] Z. Liu et al., ‘Swin Transformer: Hierarchical Vision Transformer using Shifted Windows’, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021.
[45] E. Dilocker et al. ‘Weaviate’. Github Repository, Github. 2023
[44] A. Li, A. Jabri, A. Joulin, and L. van der Maaten, ‘Learning Visual N-Grams from Web Data’, arXiv preprint arXiv:1612.09161. 2017.
[45] G. Forman and M. Scholz, ‘Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement ABSTRACT’, SIGKDD Explorations, vol. 12, pp. 49–57, 01 2010.
[46] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, ‘Johnson-Lindenstrauss Lemma, Linear and Nonlinear Random Projections, Random Fourier Features, and Random Kitchen Sinks: Tutorial and Survey’, arXiv preprint arXiv:2108.04172. 2021.
[47] “What is Residual Connection?” Towards Data Science. (accessed 5/2/23) https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

[48] “Nearest neighbors and vector models – part 2 – algorithms and data structures.” erikbern.com. https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html (accessed 5/2/23)

[49] W. Muła, Wikimedia Commons. https://en.wikipedia.org/wiki/Skip_list#/media/File:Skip_list.svg (accessed 5/2/23)

[50] “Hierarchical Navigable Small Worlds (HNSW) | Faiss: The Missing Manual.” pinecone.io. https://www.pinecone.io/learn/hnsw/ (accessed 5/2/23)

[51] J. Johnson, M. Douze, and H. Jégou, ‘Billion-scale similarity search with GPUs’, IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2019.

[52] A. Paszke et al., ‘PyTorch: An Imperative Style, High-Performance Deep Learning Library’, in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.