A SYSTEM FOR LARGE-SCALE IMAGE AND VIDEO RETRIEVAL ON EVERYDAY SCENES

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

Arun George Zachariah

Dr. Praveen Rao, Thesis Supervisor

JUL 2022

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

A SYSTEM FOR LARGE-SCALE IMAGE AND VIDEO RETRIEVAL ON EVERYDAY SCENES

presented by Arun George Zachariah, a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Praveen Rao

Dr. Prasad Calyam

Dr. Kannappan Palaniappan

Dr. Eduardo J. Simoes

ACKNOWLEDGMENTS

First and foremost, I would like to express my most sincere gratitude and deepest appreciation to my advisor and committee chair, Dr. Praveen Rao, for his guidance, support, motivation, and encouragement. Dr. Rao, you were always there by my side. You could see right through me and know when something was wrong. You always cared for me, and I knew that, I could come to you with any of my problems. I was fortunate to have had you in my life. Thank you for inspiring me and for guiding me. You are an exceptional teacher and, beyond that, an exceptional person. You will always have a special place in my heart.

I would also like to thank my committee members Dr. Kannappan Palaniappan, Dr. Prasad Calyam, and Dr. Eduardo J Simoes for being a part of my doctoral committee and for their support and guidance.

I would like to acknowledge and thank the National Science Foundation (Grant nos CNS-1747751, CNS-2034247, and IIP-2024429), University of Missouri-Kansas City (UMKC), University of Missouri-Columbia (MU), and TripleBlind for funding my Ph.D.

I am glad to have had the opportunity to work with and be a part of the Big Data Lab at UMKC and the Scalable Data Science Lab at MU. I consider it a privilege to have worked alongside each one of you. My heartfelt thank you to each and every one of you. My fond memories with the lab will last a lifetime.

Ph.D. has been a test of endurance in many aspects, including life outside of research. I would like to take this opportunity to specially thank a few priceless friends I have had, without whom I could have never survived the program. Firstly,

to Aditya, for his continuous support throughout my Ph.D. journey, giving me a feeling of home, away from home. I would never be able to repay for all that you have done for me. Secondly to Mohamed, for all the lunch we had together and the advice you gave. They gave me the strength to push further. Thirdly, to Shivika for being there for me during some of the darkest times of my life. I will always turn to you for advice. Fourthly, to Kadir. I cannot thank you enough for your presence and the energy you gave me in all forms. Fifthly, to Ahmad for his kind support. And finally, to Gharib for the invaluable support and mentorship. I would also like to thank my church family, Rebes, the Bells, the Doddas, and the Arthurs, for their support, prayers, and fellowship.

Last but not the least, I would like to thank, from the bottom of my heart, my parents, for their love and sacrifice, my grandparents, for their prayers, and my teachers, for molding me and inspiring me. A heartfelt thank you!

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

There has been a growing amount of multimedia data generated on the web today- in terms of size and diversity. This has made accurate content retrieval with these large and complex collections of data a challenging problem. Motivated by the need for systems that can enable scalable and efficient search, we propose QIK (**Q**uerying **I**mages Using Contextual **K**nowledge). QIK leverages advances in deep learning (DL) and natural language processing (NLP) for scene understanding to enable large-scale multimedia retrieval on everyday scenes with common objects. The system consists of three major components: Indexer, Query Processor, and Video Processor. Given an image, the Indexer performs probabilistic image understanding (PIU). The PIU generated consists of the most probable captions, parsed and represented by tree structures using NLP techniques, and detected objects. The PIU's are stored and indexed in a database system. For a query image, the Query Processor generates the most probable caption and parses it into the corresponding tree structure. Then an optimized tree-pattern query is constructed and executed on the database to retrieve a set of candidate images. The candidate images fetched are ranked using the tree-edit distance metric computed on the tree structures. Given a video, the Video Processor extracts a sequence of key scenes that are posed to the Query Processor to retrieve a set of candidate scenes. The candidate scene parse trees corresponding to a video are extracted and are ranked based on the number of matching scenes. We evaluated the performance of our system for large-scale image and video retrieval tasks on datasets containing everyday scenes and observed that our system could outperform state-of-the-art techniques in terms of mean average precision.

Chapter 1

Introduction

We are in an era where there is an explosion in the amount of data generated on the web. The inventions of mobile technology like smartphones and tablets, along with advancements in mobile networks, have enabled creating and sharing of content with ease. Our digital and connected lifestyle combined with our love for social media has fueled data creation. The actual amount of data on the web today is difficult to calculate. The amount of multimedia data (a combination of diverse content such as text, images, animations, and videos) that we produce and consume in a day is truly mind-blowing. For example, Facebook[1], the largest social media platform, with 32 billion active users, sees 300 million photos uploaded every day[2]. Instagram[3], built around sharing photos and videos, has more than a billion active users a day. In 2021, there were 95 million photos and videos shared on Instagram in a single day. Since its

[1] https://www.facebook.com

[2] https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

[3] https://www.instagram.com

inception, over 40 billion photos and videos have been uploaded and shared[4]. Users on Snapchat[5], a camera company, see 527,760 photos shared every minute[6]. 4,146,600 videos are watched every minute on YouTube[7], a free video-sharing website[6]. This amount and the rate of data generation have necessitated the need for search engines. Search engines provide access to tons of data on the web that could be browsed by a user using keywords or phrases. They have now become a ubiquitous part of our life. For example, Google[8] processes 3.5 billion searches per day, which sums up to 1.2 trillion searches per year[9]. DuckDuckGo[10], a popular search engine emphasizing search privacy, serves an average of 98.79 million search queries a day. Storage and retrieval of these large and diverse collections of data have always been a challenge. Search engine optimization (SEO) is used by website maintainers to improve upon the visibility of a web page on a search engine. More recently, there has been a shift in users' interest from traditional text-based retrieval to content-based retrieval [1].

Content-based image retrieval (CBIR) has been a topic of research for many years. In CBIR, given a query image, the goal is to find images in the database that are similar to the query image. Typically, an image is mapped to a feature vector and a similarity metric is defined between feature vectors and used to identify images similar to the query image. CBIR has been particularly challenging due to the diversity of images on the web. This could include similar images with different backgrounds, occlusions in images (wherein two or more objects come too close to each other), view-

[4] https://www.wordstream.com/blog/ws/2017/04/20/instagram-statistics

[5] https://www.snapchat.com

[6] https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

[7] http://www.youtube.com

[8] http://www.google.com

[9] https://www.internetlivestats.com/google-search-statistics

[10] https://duckduckgo.com

point differences, etc. Advances in computer vision and deep learning have resulted in numerous research efforts and commercial solutions for CBIR. As image datasets continue to grow in volume on the Web and in domains such as healthcare, insurance, precision agriculture, remote sensing, astronomy, and defense, there continues to be great interest in efficient large-scale image retrieval.

Content-based video retrieval (CBVR) extends CBIR aiming to find videos in the database similar to the query video. Content in CBVR could refer to colors, shapes, textures, objects, faces, and audio in the frames of a given video. The most common first step for most content-based video retrieval involves segmenting the videos into frames. They are then mapped to a feature vector and aggregated. Post aggregating the frame-level feature vectors, a similarity metric is defined to identify videos similar to the query video. Video data possesses a multitude of information with applications in education and entertainment. But they are under-utilized unless systems are capable of retrieving relevant videos and at the same time while filtering out unwanted videos [2]. Hence, even today, video retrieval continues to be one of the most exciting and fastest-growing research areas in the field of multimedia technology [1].

Apart from web search, content-based retrieval finds its application in numerous domains. For example, content-based retrieval could help prevent video piracy and detect copyright violations by fetching all images and videos that are similar to a given image or video. CBIR could be used to obtain similar patterns of interest amongst satellite images. Terrapattern[11] has been building a CBIR system wherein a user could select any spot in a satellite image to be presented with satellite images containing

[11] `https://www.gislounge.com/terrapattern-search-engine-satellite-imagery/`

similar spots. CBIR also finds its application in Digital Pathology, primarily to obtain similar images/cases within large repositories. Lagotto[12], by Huron Digital Pathology, enables its users can find similar cases and read multiple pathology reports in real-time. With geospatial data warehouses becoming widespread, CBIR systems hold the key to efficient exploration. Video retrieval also finds its application in medical education (such as VuMedi[13]), providing fast access to specialized and relevant videos.

While CNNs have replaced descriptors built using local features, the primary limiting factor in web-scale information retrieval is feature dimensionality. In addition, models require larger memory capacity and access to Graphics Processing Units (GPUs), which limits their usage to specialized hardware. Google uses commodity hardware (composed of over 1000 computers)[14]. Striking the right balance of computation efficiency and retrieval performance is an active research area that could find its application in numerous domains. Moreover, to construct image representation, there also lies a semantic gap encountered in understanding the query. Understanding a query transcends the logical aspects of the query and the intellectual and emotional sides of the user posing the query. [3]

In this dissertation, we propose a system called `QIK` (Querying Images Using Contextual Knowledge) aimed at large-scale multimedia retrieval, i.e., videos and images comprising of everyday scenes with common objects in them. The system synergistically combines recent advances in Deep Learning (DL) and Natural Language Processing (NLP) to capture the context of everyday scenes and learn the relationships between objects in them to enable efficient retrieval. The key contributions of

[12] `https://www.hurondigitalpathology.com/lagotto/`

[13] `https://www.vumedi.com`

[14] `http://www.claytonstechnobabble.com/2011/08/how-does-google-do-it-meet-google-file.html`

our work are as follows:

- `QIK` is a generic framework that aims to capture the context of everyday scenes and learn the relationships between objects in them for efficient image retrieval. In order to facilitate this, `QIK` generates the PIU of an image using state-of-the-art deep learning models designed for image captioning and object detection tasks. The PIUs are processed at the time of retrieval.

- `QIK` leverages the advancements in NLP for linguistic analysis of the PIUs. For instance, image captions are transformed into tree structures which are widely used in computational linguistics [4]. These tree structures are queried during the filtering step of image retrieval and further used for ranking the candidate images retrieved.

- We conducted an evaluation of `QIK` against state-of-the-art instance retrieval techniques such as DELF [5], DIR [6], CroW [7], and FR-CNN [8] using the Microsoft COCO dataset [9], which is a well-known dataset containing complex everyday scenes with common objects. We computed the mean average precision (mAP) for top-k matches of a query to compare `QIK` and its competitors. First, we observed that using image captions for filtering and ranking, `QIK` performed better than when only the detected objects were used for image retrieval in terms of mAP. This asserts that leveraging relationships between objects during image retrieval yields better quality results. Furthermore, `QIK` using image captions outperformed the aforementioned competitors in terms of mAP.

- We leverage the modular nature of `QIK` and extend it to enable efficient video retrieval. We evaluated the video retrieval performance of `QIK` against state-of-

the-art techniques such as CSQ [10] and DnS [11] using the MSR-VTT [12]. `QIK` outperformed its competitors in terms of mAP for large-scale video retrieval.

The rest of the dissertation is organized as follows: Chapter 2 gives a brief background on the important concepts used in the dissertation; Chapter 3 discusses recent related work followed by our motivation in Chapter 4; Chapter 5 presents the design details; Chapter 6 describes the implementation details; Chapter 7 discusses the evaluation of QIK and comparison with state-of-the-art techniques for image and video retrieval tasks; and finally we present our conclusion and future work in Chapter 8.

Chapter 2

Background

2.1 eXtensible Markup Language

eXtensible Markup Language (XML) [13], first published as a W3C Recommendation on February 10, 1998, was primarily designed for efficient storage and transportation of a wide variety of data over the internet. XML facilitates data exchange between programs and platforms and is widely used in service-oriented architectures (SOA) where services (discrete units of functionality) are provided by separate components connected over the web. These components belong to different vendors and are implemented using different technologies. The format of the data exchanged with an is XML is standardised using an XML schema (XSD).

Figure 2.1 shows a simple XML document. The key components of any XML documents are:

- **XML declaration.** Every XML document begins with an XML declaration,

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <car category="sedan">
    <make>Honda</make>
    <model>Accord</model>
    <year>2021</year>
    <transmission>manual</transmission>
    <price>24970</price>
  </car>
  <car category="SUV">
    <make>Toyota</make>
    <model>Land Cruiser</model>
    <year>2020</year>
    <transmission>auto</transmission>
    <price>85415</price>
  </car>
</cars>
```

Figure 2.1: Example of an XML

which defines the XML version and character encoding.

- **Tag.** A *tag* is a construct that begins with a `<` and ends with a `>` The three types of tags are:
  - *start-tag represented* as `<tag>`
  - *end-tag represented* as `</tag>`
  - *empty-element-tag* represented as `<tag />`
- **Element.** An *element* is the basic construct in an XML document. It begins with a start-tag and ends with a matching end-tag of consists only of an empty-element-tag. The XML tree shown in Figure 2.1 starts with the *root element* `cars` and branches to the *child elements* `car`. `make`, `model`, `year`,

`transmission`, and `price` are the *child elements* to `car`.

- **Attribute.** An *attribute* consists of a name-value pair that exists in an *element*. `category` is the *attribute* to the *element* `car`.

2.2 XPath

XPath (XML Path Language) is the query language for selecting nodes in an XML document. It leverages the tree representation of XML documents and uses path expressions to navigate and select nodes. A simple XPath query can be written as $/A_1::N_1[p_1]/\cdots/A_i::N_i[p_i]/\cdots/A_n::N_n[p_n]$, where A_i denotes an XPath axis, N_i denotes an XML element name, and p_i denotes a predicate of that node. A predicate may be empty for a node. Using / selects from the root node, while // selects nodes in the document from the current node, no matter where they are. :: is used to separate an axis name from a node test in an XPath expression. Although there are 13 XPath axes [14], we only use 4 of them in our subsequent chapters:

- `child` to indicate a parent-child relationship.
- `descendant` to indicate an ancestor-descendant relationship.
- `following` to indicate that a node follows the other in document order (a.k.a. preorder).
- `following-sibling` to indicate that two nodes share a common parent.

Example 1. *To select all the* `model` *nodes, the XPath expression would be* `/cars/car/model`.

Example 2. *To obtain the price for* `Land Cruiser`*, the XPath expression would be* `/cars/car[model[text()="Land Cruiser"]]/price/text()`.

Example 3. *To obtain the* `model` *nodes following* `Accord`*, the XPath expression would be* `/cars/car[model[text()="Accord"]]/following::car`.

2.3 Tree Edit Distance

Tree edit distance [15, 16, 17] (TED) is the minimum cost, in terms of edit operations, required to transform one ordered labeled tree to another. The edit operations performed on the tree nodes include:

- Insertion
- Deletion
- Replacement

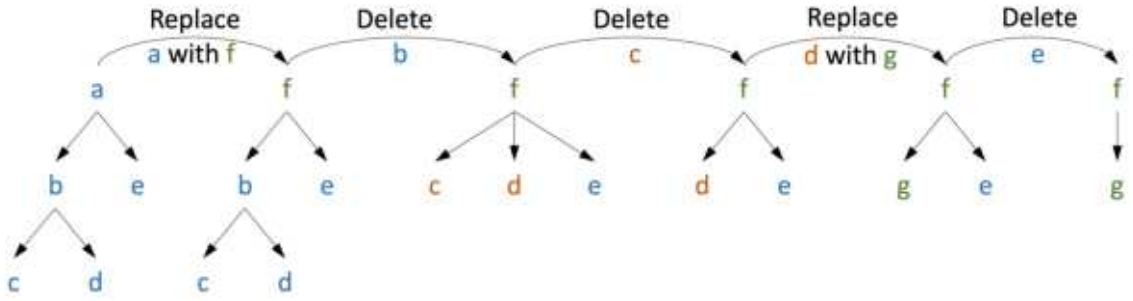


Figure 2.2: Example of tree edit distance operations

Figure 2.2 shows an example of the set of operations to be performed to transform tree A (represented as `a(b(c, d), e)`) to tree B (represented as `f(g)`). In 1989,

Zhang et al. [18] developed a recursive algorithm that involved finding an optimal matching solution to a given pair of trees. The algorithm involves finding the keyroots, which are the nodes that have the common leftmost leaf. The tree distance is then calculated for the combination of each of the keyroot nodes of one tree to the keyroot nodes of the other tree. The algorithm has a time complexity of $O(n^4)$ time and $O(n^2)$ space for trees with n nodes. Over the years, various optimizations were proposed that brought down the time and space complexity [19, 20, 21, 22]. Today, TED has found a wide range of diverse applications like software engineering, natural language processing, and bioinformatics [23].

2.4 Bloom Filter

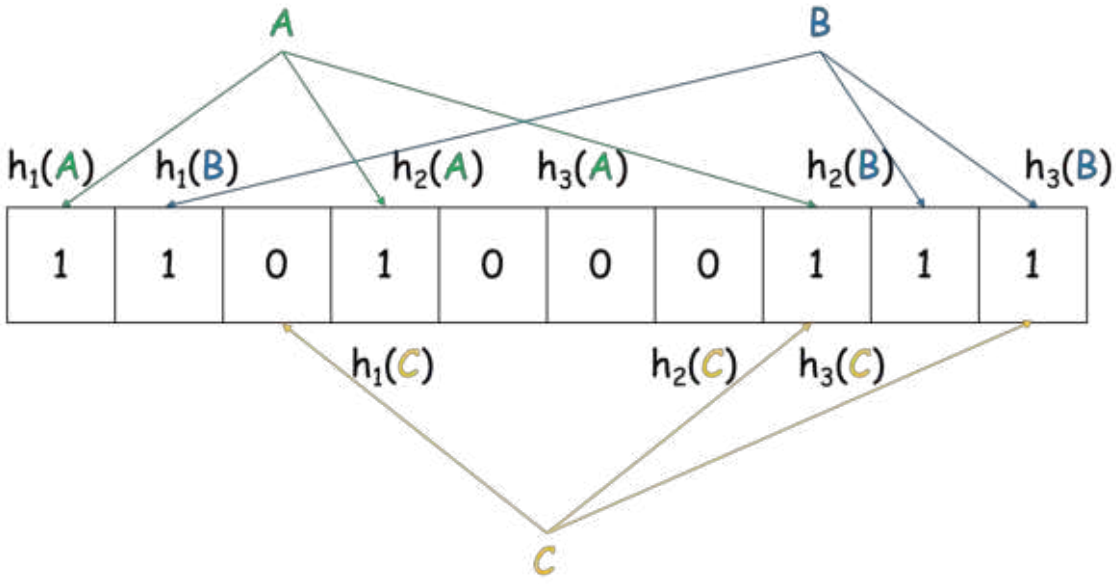


Figure 2.3: Bloom filter

Bloom filter [24] is a randomized and memory efficient probabilistic data structure

designed to test if a given element is a part of a set. A Bloom filter maintains an array of n bits, initialized to 0. There are m hash functions defined, that maps the set of elements to one of the n positions in the array. The hash functions defined must ensure independent and uniform distribution of hash codes. To insert an element to a Bloom filter, the m hash functions are applied on it to get the m bit positions. These bits are set to 1. To test if an elements exists in the set, the same m hash functions are applied and the corresponding bit positions are check if the value is set to 1. If any of the bit positions are 0, the element is definitely not present in the set. If all of the bit positions are 1, the element is either present in the set, or the the bits were set to 1, during the insertion of other elements, resulting in false positives.

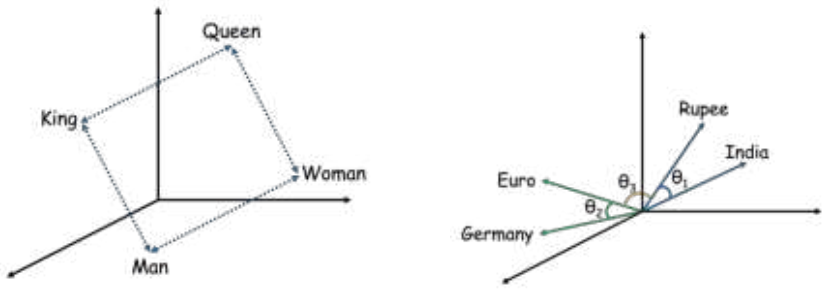
2.5 Convolutional Neural Networks

Images are represented as an array (or a matrix) of pixels arranged in columns and rows. The dimensions of an image posed considerable challenges for artificial neural networks. To overcome this Convolutional Neural Networks (CNNs) were proposed. CNNs aim at reducing images to a form that is earlier to process while at the same time without losing critical features. Using a filter (a set of learnable weights learned using backpropagation), a CNN can capture the spatial and temporal dependencies in an image. During a convolution, a filter shifts based on the Stride (the number of pixels to be shifted) length defined, performing matrix multiplications between filter elements and the portion of the image under the filter. Pooling layers are required to reduce the size of the convolved features. Reducing the dimensions helps reduce the number of parameters to be learned and the compute needed. The two most

common signs of pooling are Max Pooling and Average Pooling. Max Pooling returns to maximum value under the portion covered by the Kernel. On the other hand, Average Pooling returns the average of all the values under the portion covered by the Kernel. Convolution of an image with different forms filters can capture low-level features such as edges, color, gradient orientation, etc. Adding multiple convolutional layers help capture the high-level features to enable a wholesome understanding of images. The convolutional features are passed through a fully connected layer to learn non-linear functions mapping these features. Over the years, there have been various variants of CNN architectures developed and deployed (such as AlexNet [25], VGG [26], ResNet [27], etc.) leading to astonishing advances in the field of deep learning.

2.6 Word Embeddings

Since computers (and Machine Learning algorithms) operate on numbers, there was a need to represent linguistic entities as mathematical entities. Hence word embeddings were introduced. Word embeddings are learned representations (represented by a real-valued vector in a latent space) of a word such that words with similar meanings have similar representations. Tomas Mikolov et al. proposed Word2Vec [28] for learning high-quality word embeddings efficiently from large corpora of text (with billions of words). Two different methods were proposed: the Continuous Bag-of-Words (CBOW) model and the Continuous Skip-Gram model. The CBOW model learns the embedding by predicting the current word based on the context. On the other hand, the Continuous Skip-Gram model predicts the context for the current word by



(a) (b)

Figure 2.4: Word embedding examples

predicting words within a defined range before and after it.

Representing words as vectors allows us to perform vector arithmetic on them like the famous example: example, $\vec{king}$ - $\vec{man}$ + $\vec{woman}$ = $\vec{queen}$ (seen in Figure 2.4a).In the case of Word2Vec, the similarity between words is measured in terms of the cosine distance between their vector representations. The cosine distance between two vectors $\vec{A} = (A_1, A_2, ..., A_n)$ and $\vec{B} = (B_1, B_2, ..., B_n)$ (equal to the cosine of the angle between the vectors) is computed as:

$$cosine\ distance = 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2}\sqrt{\sum_{i=1}^{n} {B_i}^2}}$$

Similar words would have a smaller cosine distance. For example, in figure 2.4b, the words Rupee and India; and Euro and Germany are closer to each other as one represents the current of the other. Hence they have a low cosine distance compared to the distance between Rupee and Euro, the two different currencies (i.e $\theta_1 < \theta_3$ and $\theta_2 < \theta_3$, while $\theta_1 \approx \theta_2$).

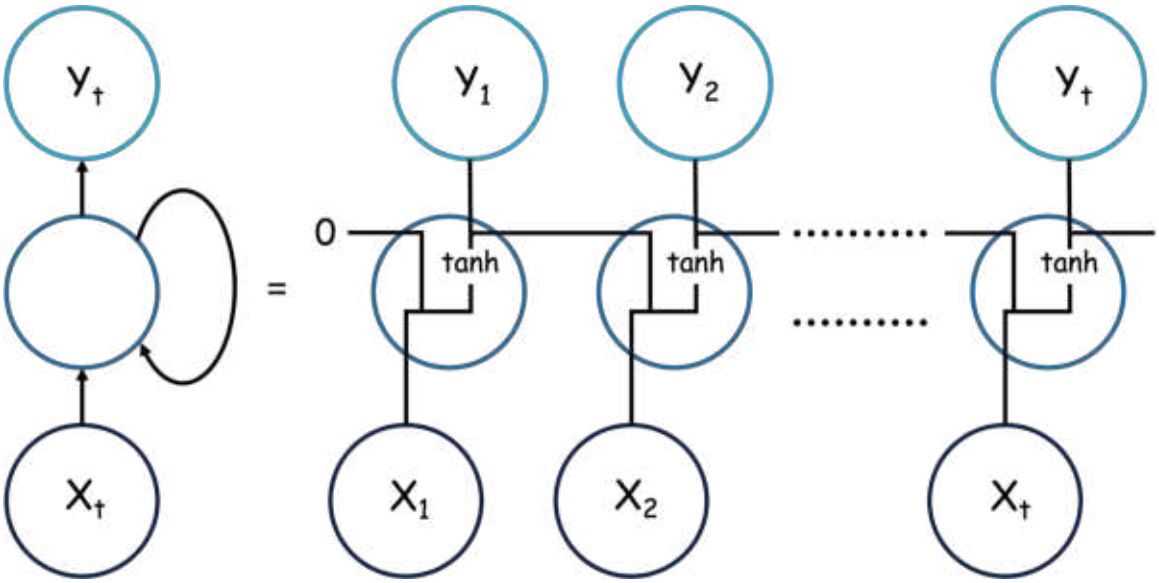


Figure 2.5: Recurrent neural network

2.7 Recurrent Neural Networks

Feedforward neural networks were limited to learning representations among data points independent of each other. Hence, to learn representations of sequential data, such as time series, speech, text, audio, video, etc., where one data point depends upon the previous data points, Recurrent Neural Networks (RNNs) were introduced. RNNs are known for their concept of "memory" (by virtue of feedback loops) that help them retain past information to predict the next in the sequence. As seen in Figure 2.5, the inputs to an RNN include the present and the recent past data. Weights are assigned to them to output a prediction. The weights for the RNNs are learned using gradient descent and backpropagation through time. To overcome the vanishing gradient problem encountered while training artificial neural networks with gradient-based learning methods and backpropagation, Hochreiter et al. [29] introduced Long Short Term Memory (LSTM). Figure 2.6 shows the LSTM cell contents. Information

from the previous hidden state and information from the current input are passed through the sigmoid function. This is called the forget gate. The output from the sigmoid function is in the range of 0-1, with values closer to 0 being forgotten. The information from the previous hidden state and information from the current input are passed through sigmoid and tanh functions. Their outputs are multiplied together to form the input gate. The output from the forget and input gates are aggregated to obtain a new cell state that is carried over to the next time step. Meanwhile, this output is also passed through a tanh function and multiplied with the information from the previous hidden state and information from the current input, passed through the sigmoid function, to form the hidden state carried over to the next time step.

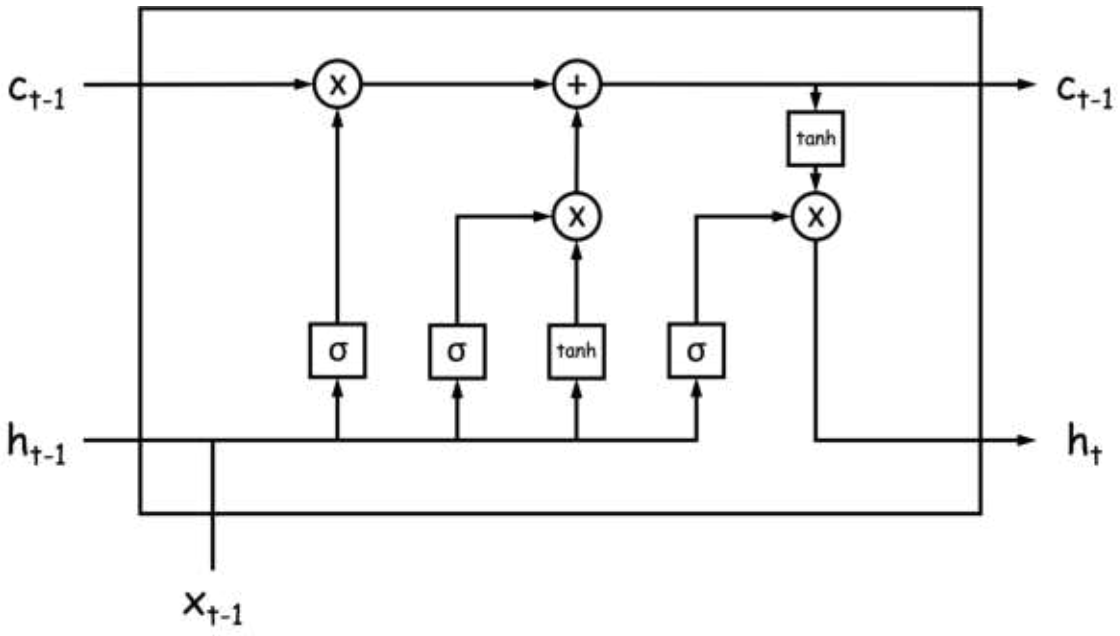


Figure 2.6: Long short-term memory [29]

2.8 Image Captioning

Image captioning involves describing the content of an image in a textual form. It combines computer vision to understand an image and natural language processing for generating textual descriptions that are syntactically and semantically correct. It has been an important task in Deep Learning, with numerous impactful applications.

Vinyals et. al. proposed Show and Tell [30, 31], a neural and probabilistic framework for generating textual image descriptions. The model draws inspiration from advances in machine translation that involve using a RNN to encode variable-length input into a fixed dimensional vector that can be decoded to the desired output sentence. In line with traditional machine translation approaches, the Show and Tell model architecture comprises an encoder model and a decoder model. A CNN is used as the encoder to convert an image to a vector representation. This vector forms the input to a RNN that acts as the decoder. The CNN model comprised the Inception V1 image classification model [32] (a state-of-the-art at the time). The RNN model used was LSTM [29] due to its state-of-the-art performance on *sequence* tasks. Unique words in the captions were used to create a vocabulary to map words to a corresponding index. The words were one-hot encoded and embedded to the same latent space as the images using word embedding. Special start and stop words are added to captions since not all captions are of the same length. When the number of words in the caption is less than than the LSTM units, the remaining spaces were padded using another special word. For obtaining the best captions (or predictions), the model uses Beam Search. In this approach, k best sentences are considered candidates at each time step t. In the next time step, $t + 1$, k best sentences are chosen from the sentences constructed from k most probable words with k sentences. The

process is repeated to attain k best captions for a given image.

Mokandy et al. proposed ClipCap [33], that fused CLIP [34] with GPT-2 [35] to generate image captions. During training, the images are passed through CLIP's image encoder network. The output of this network is mapped to the embedding space of GPT-2 by a mapping network. The mapping network comprises of a trained lightweight transformer network. The mapping network produces a prefix for each image which is appended to the embedding of its annotated caption. This combined embedding is passed through GPT-2 to predict caption tokens conditioned on the prefix. At the time of inference, the image encodings from CLIP are passed through the mapping network to generate a prefix. The GPT-2 then outputs the caption starting from the prefix.

Chapter 3

Related Work

As we aim at content-based multimedia retrieval, we dissect our literature review into advances in content-based image and video retrieval.

3.1 Image Retrieval

Current image retrieval systems typically have two stages: the filtering stage to identify a set of candidate images and a re-ranking stage, where a small number of similar candidates are re-ranked based on specific criteria. Several approaches used hand-crafted local features, composed of a feature keypoint and a feature descriptor. Lowe [36] proposed Scale Invariant Feature Transform (SIFT) that used hand-crafted local features based on the appearance of the objects at points of interest. Motivated by SIFT, Bay et al. [37] proposed speeded up robust features (SURF) to extract points of interest. Inspired by the bag-of-words representation for text categorization, Csurka et al. [38] proposed a bag-of-keypoints approach to feature vector

construction and clustering. To enable web-scale search, Philbin et al. [39, 40] built a scalable image-feature vocabulary that mapped each visual region to a weighted set of words.

Advances in CNNs have resulted in new methods for image understanding including, image recognition and object detection. Deep CNNs have achieved remarkable accuracy for object recognition on standard benchmarks (e.g., GoogLeNet [32], Inception v3 [41], ResNet [27]). Several techniques have recently explored the use of CNNs for large-scale image retrieval. They rely on CNN features for global image representations enabling fast filtering. Babenko et al. [42] demonstrated this by using the features obtained from a CNN, trained to recognize Image-Net [43] classes, to the task of image retrieval. They also explored further by aggregating local deep features to produce compact descriptors for image retrieval [44]. Radenovic et al. [45] introduced unsupervised fine-tuning of CNNs for image retrieval using information extracted from a Structure-from-Motion pipeline trained to construct 3D models from 2D images. Jegou et al. [46] proposed VLAD (Vector of Locally Aggregated Descriptors) that enabled large-scale image search using CNN features. VLAD aggregated local image descriptors into a vector of limited dimensions, allowing descriptors for large image datasets could fit in memory. Arandjelovic et al. [47, 48] improved further on VLAD descriptors for accurate image retrieval. Salvador et al. [8] used image-wise and region-wise representations pooled from an object detection CNN such as Faster R-CNN [49]. The image-wise representations served as global features and were used during the filtering step. The region-wise representations served as local features and were used for re-ranking. Gordo et al. [6] produced a global and compact fixed-length representation of each image by aggregating many region-wise descriptors so that it

is robust to scale and transformation. The regions to be pooled were predicted using a region proposal network. Noh et al. [5] used local feature descriptor that employed an attention-based keypoint selection mechanism to identify semantically useful local features there were needed for image retrieval. More recent work by Teichman et al. [50] used a novel region aggregation method for image retrieval. It extended the idea of aggregated selective math kernels proposed by Tolias et al. [51] by extracting object regions from an image and its local features, proposed by Noh et al. [5]. The region aggregation method helped in re-balancing the visual information in the image. Kalantidis et al. [7] proposed an efficient non-parametric weighting and aggregation scheme to transform convolutional image features to a compact global image feature. The output of the convolutional layers is aggregated before the fully connected layers. For re-ranking, local image representations from CNNs have been employed through spatial matching and geometric verification [52, 5, 53]. Chum et al. [54, 55] used query expansion approaches to increase retrieval performance significantly but at the cost of higher query times. To improve on this limitation, Tolias et al. [56] proposed to use CNN activations of convolutional layers for filtering and re-ranking.

Another interesting success was the development of image captioning models like Show and Tell [30, 31] and Show, Attend and Tell [57] that used a combination of convolutional layers/networks and recurrent neural networks. Thus, it is now more feasible than before to extract useful knowledge from images and understand their context on a large scale. Gordo et al. [58] used human-annotated region-level captions during training time to generate global visual representation of images for semantic retrieval. At query time, only the query image is used without any captions. VistaNet [59] combines text with images but for sentiment analysis. Hodosh

et al. [60] applied sentence-based image descriptions as a ranking task in text-based image search. Socher et al. [61] proposed a dependency tree RNN (DT-RNN) that used dependency trees for sentences into the same vector space as the outputs of CNNs applied to images allowing querying images using a textual description.

We now describe in detail some of the techniques that use hand-crafted features as well as features extracted from CNN's for retrieval. They are considered to be state-of-the-art in the area of image retrieval and we later use them for evaluation in Chapter 7.

- **Deep Image Retrieval:** Gordo et al [6]. proposed an effective and scalable approach for instance-level image retrieval that involves finding in a database of images a match for an object in a query image. The approach implements a three-stream siamese network. At the time of training, the input triplet comprises the query, a relevant image, and a non-relevant image. In each stream of the siamese network, the images are passed through a pre-trained VGG16 CNN to extract their local features. The features are max-polled in different regions of the images using a region proposal network (RPN). RPN trains on the local features extracted by the CNN and bounding boxes (obtained from the ground truth) depicting the region of interest (ROI). A binary class label is assigned to each candidate box depending on how much it overlaps with the ROI. The loss function is a combination of a classification loss and a regression loss, which is optimized through backpropagation and stochastic gradient descent. The pooled features are l2-normalized, whitened with PCA (using a shifting and fully connected layer), and l2-normalized again. The features are then sum aggregated and l2-normalized. Learning is done using triplet loss comparing

the query image against the relevant and non-relevant images. The gradient is backpropagated through the three streams of the siamese network, including the CNN and PCA layers, to update the weights. In short, the relevant images act as regularizers, while the non-relevant images force learning in the network. Hence the approach produces compact representations for each image by aggregating many region-wise descriptors. During testing, the query image is passed through the query stream network of the trained model to output a compact vector representation. It can then be compared against the dataset image representations using a simple dot product.

- **Large-Scale Image Retrieval with Attentive Deep Local Features:** Noh et al.[5] propose an attentive local feature descriptor, trained over a landmark image dataset, for large-scale image retrieval. Dense features are extracted from an image by passing it through a fully convolutional network (FCN). The FCN here is the `conv4_x` (convolutional) layer from the ResNet50 model trained on the ImageNet dataset. The output of the FCN is a dense grid of local feature descriptors. For images that are of a different dimension, image pyramids are constructed. Each layer in the pyramid is passed through the FCN to construct a dense grid of local feature descriptors. They are passed through an attention layer (2 layer CNN with a softplus activation) that measures the relevance of the local features descriptors. For optimizing training and accuracy, a two-step training strategy is applied. First, the dense feature extraction model is trained over the dataset for image classification using cross entropy loss. Then, the attention-based model is trained by pooling the features obtained by a weighted sum of the features, where the weights are predicted by the attention network.

The gradients obtained from the cross entropy loss function computed over the weighted sum of the features are backpropagated to fine-tune the weights. The output is an embedding for given image input. They are then L_2 normalized and passed through a PCA layer to reduce the dimensions and further L_2 normalized. To enable efficient image retrieval, a large-scale index is constructed. For all the images in a dataset, local descriptors are extracted and indexed using a combination of KD-tree and Product Quantization. At the time of retrieval, given a query image, the local descriptors are extracted. An approximate nearest neighbor search over the index returns the top-k closest local descriptors. The final set of matches are returned after performing geometric verification to remove distractors.

- **Cross-Dimensional Weighting for Aggregated Deep Convolutional Features:** Kalantidis et al. [7] propose a framework for extracting image representations by cross-dimensional weighting and aggregation of deep convolutional neural network layer outputs. The authors refer to the features extracted through this framework as Cross-dimensional Weighted or CroW features. Sum-pooling or max-pooling is applied over the deep convolutional features obtained from the last spatial layer of a CNN (A pre-trained VGG16 model was used in the implementation). For each location in the pooled feature map, a weight derived from the spatial activation of the output layer is applied across each channel. This is called spatial weighting. Spatial weighting boosts features at locations with important visual content. For each channel, another weight, based on the sparsity of the feature maps, is assigned at each location in that channel. This is called channel weighting. Channel weighting, thus, helps regulate the effect

of channel burstiness. (Burstiness is a phenomenon in which a particular visual element appears more times in an image than what a model could predict hence corrupting the visual similarity measured). A weighted sum aggregation is performed over the location-wise and channel-wise weights. As spatial dimensions vary per image depending on the size, the original image size is retained. This is another factor that contributes to the overall performance. The resulting vector is then normalized and power transformed. Whitening and PCA are applied to the output vector to reduce its dimensions. A final normalization on this yields CroW features. To improve image retrieval performance, the query expansion technique proposed by Chum et al. [54] is applied to the CroW features. The features of the top results are summed and L2 normalized before re-querying over the ranked list of database images.

- **Faster R-CNN Features for Instance Search:** Salvador et al. [8] proposed using features extracted from an object detection CNN for instance retrieval. A Faster R-CNN [49] was used to extract both global and local features. The Faster R-CNN network comprised a Region Proposal Network (RPN) that learns a set of window locations and a classifier that identifies it as one of the classes in the training set. Global features were extracted from the last convolutional layer. To extract local features, the output of the convolutional layers is passed through a region pooling layer, to extract the convolutional activations for each object proposal learned by the RPN. The aggregated features are L2-normalized, whitened, and further L2-normalized. To create better representations and help improve retrieval performance fine-tuning and spatial reranking were performed. Best results were obtained when the weights of all

convolutional layers were updated. Updating the weights of all convolutional layers enabled convolutional features, RPN proposals, and fully connected layers to learn from the query instances.

- **Central Similarity Quantization for Efficient Image and Video Retrieval:** Yuan et al. [10] proposed Central Similarity Quantization (CSQ), a "global similarity metric" known as "central similarity" by which the embeddings of similar data pairs would converge to a common center while dissimilar data pairs would converge to different centers in an embedding space. The technique is generic and could be applied to both image and video retrieval tasks. The high dimensional data is first mapped to K-bit binary hash codes in Hamming space, comprising of the 2K, using a hash function. Then, a hash center, defined as a set of points in Hamming Space, comprising of all combinations of K bits with sufficient distance from each other, is generated. Two approaches were proposed for generating the hash centers. The first approach leverages Hadamard matrix properties, to construct hash centers with maximum mutual Hamming distance. A Hadamard matrix is a square matrix of size K with values +1 or -1 and whose rows are mutually orthogonal. The limitation of this approach is that K needs to be a power of 2. To overcome this limitation the second approach proposed to generate hash centers by randomly sampling each center vector from a Bernoulli distribution. The training data samples and labels are then associated with hash centers generated to get the semantic hash centers. For single-label data, one hash center is associated with a single category. For multi-label data, a centroid of the hash centers associated with the category is considered as the semantic hash center. CSQ then optimizes the

central similarity objective (obtained by maximizing the logarithm posterior of the hash codes with respect to the semantic hash centers) and the quantization loss (using the bi-modal Laplacian prior for quantization proposed by Zhu et. al [62]).

- **Lucene Image Retrieval:** Lucene Image Retrieval (LIRE) [63] is an open-source lightweight Java library for content-based image retrieval. LIRE extracts Color histograms, Fuzzy color and texture histograms [4], correlograms [64], MPEG-7 descriptors [65], textural features corresponding to human visual perception [66], and Color and Edge Directivity Descriptor features [67]. The image features extracted are indexed using Lucene[1], an open-source search engine library written in Java. LIRE leverages the efficient and fast disk access features of Lucene for quick retrieval. The image features extracted from a query image are sequentially compared against the indexed database image features to return the top-k matches.

3.2 Video Retrieval

Early video retrieval techniques, similar to image retrieval techniques, extracted handcrafted features from video frames to derive a representation over which a similarity metric could be applied to retrieve relevant videos. Wu et al. [68] used signatures derived from color histograms. Huang et al. [69] used color histograms, textures, color and shape signatures, and reference points to construct a one-dimensional representation. Motivated by the performance of CNN-based descriptors for image clas-

[1] `https://lucene.apache.org`

sification and retrieval, there have been numerous techniques proposed that relied on CNN for extracting video frame features which were then aggregated. Revaud et al. [70] used properties of circulant matrices on features extracted using a CNN to encode frames. Gao et al. [71] proposed a framework to create compact video representations extracted from CNNs applied on image features to capture temporal correlations. Kordopatis-Zilos et al. [72, 73, 74, 11] aggregated the keyframe features from intermediate CNN layers to learn the video representation.

More recently, self-supervised video representation learning has garnered a lot of interest. Han et al. [75] proposed the Dense Predictive Coding framework that learns encoding video blocks of an equal number of frames by predicting the next set of video blocks. Kuang et al. [76] explored contrastive learning, a self-supervised machine learning technique that involves using pairs of similar and augmented dissimilar images to learn high-level features with videos. They proposed a video-level contrastive learning framework to formulate positive and negative pairs.

With image hashing methods have shown promising results in content-based image retrieval, video hashing techniques have been extensively studied. Several efficient hashing techniques have since then been proposed. Song et al. [77] proposed Multiple Feature Hashing that learns a group of hash functions for extracting global and local features that map the vital video frames into the Hamming space. Cao et al. [78] proposed a hashing framework that combined hashing and feature pooling for powerful search. The hashes were obtained from heterogeneous hash codes and stored in a hash table to speed up the search. While these techniques looked at videos as a set of image frames, Ye et al. [79] proposed a method to transform high-dimensional data into compact binary hash codes by accounting for the spatiotemporal information

embedded in a video. Liong et al. [80] fused the temporal information across different frames within a video to learn its feature representation. Gu et al. [81] proposed Supervised Recurrent Hashing (SRH). Raw frame representations obtained from a CNN were fed to an LSTM, max-pooling, and fully connected layer to attain the fixed-length hash codes. For reducing the feature size to support massive video databases, Zhuang et al. [82] proposed using a differential LSTM (DLSTM) [83] for modeling videos. They extract one video segment to generate a highly compact fixed-length representation of the original video. Qin et al. [84] proposed Disaggregation Hashing. A high-dimensional feature vector is split into different groups by applying a form of k-means clustering. Different hash functions are then applied over the groups to obtain a binary code. Yuan et al. [10] proposed a new global similarity metric termed central similarity that leverages Hadamard matrix and Bernoulli distributions to ground similar video pairs. Zhang et al. [85] proposed Self-Supervised Temporal Hashing aimed at hashing videos into short binary codes for efficient search and retrieval.

RNNs have also been widely used to model relationships among the frames of a video. Li et al. [86] used RNNs to generate binary codes that captured the spatial-temporal structure of a video. Srivastava et al. [87] used LSTMs to learn representations of video sequences to achieve promising results. Feng et al. [88] used LSTMs to find tubelets in reference videos that semantically corresponded to a query video.

We now describe in detail some of the most recent techniques for video retrieval. We later use them for evaluation in Section 7.

- **Distill-and-Select:** Kordopatis-Zilos proposed a Knowledge Distillation framework called Distill-and-Select (DnS) [11]. Knowledge Distillation refers to the

idea of model compression. It generalizes the knowledge learned from a larger neural network (called a teacher) to train another smaller neural network (called a student). Unlike traditional content-based video retrieval techniques, by using Knowledge Distillation, DnS can attain high retrieval performance while maintaining computational efficiency. The DnS framework consists of three networks: a coarse-grained student network, a fine-grained student network, and a selection network. The coarse-grained student network provides fast retrieval but has low performance. The fine-grained student network offers high performance but is computationally expensive. The selection network balances performance and efficiency by computing the fine-grained student network feature similarity of only a required set of video pairs. At the time of indexing, given a video database, the features from the three networks are obtained and stored. During retrieval, given a query video, the features from the three networks are first extracted. The dot product of the coarse-grained features of the query and database videos are then computed. The selector network then decides whether the query-target feature pairs need to be re-ranked by the fine-grained student networks. The tensor dot of the query-target feature pairs is computed followed by the Chamfer distance. The output frame-frame similarity matrix is passed through a Video comparator (a CNN module capable of capturing temporal patterns) to calculate video-video similarity. Finally, Chamfer distance is computed to obtain the resultant query-target video-video similarity. Two fine-grained student networks were proposed that used the ViSil Architecture [74]. The first add more trainable parameters to the original architecture for better performance, while the second uses a Binarization function to hash

features into Hamming space to attain space efficiency.

- **Central Similarity Quantization for Efficient Image and Video Retrieval:** Yuan et. al proposed Central Similarity Quantization (CSQ) [10], described in detail in section 3.1. For video retrieval, to retain temporal information videos are hashed using 3D CNNs [89]. The training data videos and labels are then associated with hash centers generated to further obtain the semantic hash centers. Videos having the same center as the query video are considered to be a match.

Chapter 4

Motivation

We see in Chapter 3 that, with the advances in CNNs, various techniques have been proposed over the years that leverage features from CNNs for accurate image retrieval. While many techniques achieve state-of-the-art results, they were tested on datasets that contain objects such as buildings, scenic views, and landmarks, such as the Oxford Dataset [39][1], the Paris Dataset [40][2], the INRIA Dataset [90][3], the Google Landmarks Dataset [91][4], etc. Images containing everyday scenes with common objects are quite different from the images used previously as they contain objects both in the foreground and in the background. In such an image, certain objects become the main focus of the image and dominate its context. Based on our experiments using the MS COCO dataset, we observed that the techniques based on CNN-based features failed to precisely capture the main aspect of such an image leading to false

[1] https://www.robots.ox.ac.uk/~vgg/data/oxbuildings/
[2] https://www.robots.ox.ac.uk/~vgg/data/parisbuildings/
[3] https://project.inria.fr/aerialimagelabeling/
[4] https://github.com/cvdfoundation/google-landmark

(a) Query image for DELF [5]

(b) A false positive by DELF

(c) Query image for FR-CNN [8]

(d) A false positive by FR-CNN

(e) Query image for CroW [7]

(f) A false positive by CroW

(g) Query image for DIR [6]

(h) A false positive by DIR

Figure 4.1: Query images and false positives output by CNN based techniques

positives in many cases. For example, for the query image in Figure 4.1(a), while DELF returned an image of a kitchen (Figure 4.1(b)), neither was there anyone cooking nor were any people in it. For the query image in Figure 4.1(c), FR-CNN returned an image (Figure 4.1(d)) that contained a person posed in a manner similar to the query image bust was a baseball player instead of a skateboarder. For the query image in Figure 4.1(e), CroW returned an image (Figure 4.1(f)) of a room with daylight but missed out on the people that were present in the query image. Finally, for the query image in Figure 4.1(g), DIR returned an image (Figure 4.1(h)) of a restaurant kiosk similar to the query image but misses out on a person with a tie.

We emphasize that human cognition can capture the key essence of an image and describe it aptly via a caption. Through a caption, certain objects and regions in an image that do not contribute to understanding the main context of the image could be ignored. We posit that through an accurate image captioning system, we can aptly describe images containing everyday scenes with common objects using a caption. Hence an image retrieval system based on automatically generated captions of images could outperform CNN-based features for accurately retrieving these images. This motivated us to design a new approach called `QIK` that can provide superior image retrieval performance than its competitors for images containing everyday scenes. `QIK` moves away from traditional CNN-based features and instead uses predictions made by deep learning networks designed for image understanding tasks. It uses the predictions made by these models to probabilistically understand an image in a novel fashion to generate its PIU. It synergistically combines this understanding with modern NLP techniques to enable efficient and accurate large-scale image retrieval.

A video is a sequence of images called frames. We also posit that an image retrieval

system, that could accurately fetch images with everyday scenes, could be leveraged to retrieve frames of a video that is relevant to the query. To enable accurate video retrieval, the temporal nature of a video can be leveraged by the system by ensuring an ordering in the sequence of frames retrieved. Hence, through this dissertation, we propose a system called `QIK` (**Q**uerying **I**mages Using Contextual **K**nowledge) for large-scale image and video retrieval, with everyday scenes containing common objects, by synergistically combining PIUs and NLP.

Chapter 5

Design

In this chapter, we describe our proposed design for QIK. Our design is motivated by the fact that techniques that use CNN-based features fail to fetch precise matches when images comprise of everyday scenes with common objects.

5.1 `QIK` Architecture

Figure 5.1 shows the architecture of `QIK`. The three main components of the `QIK` architecture are **Indexer**, **Query Processor**, and **Video Processor**. In sections 5.1.1 through 5.1.3, we provide an overview of each of these components.

5.1.1 Indexer

Given an image repository, the Indexer performs a linguistic analysis using state-of-the-art deep learning models for image captioning and object detection to generate

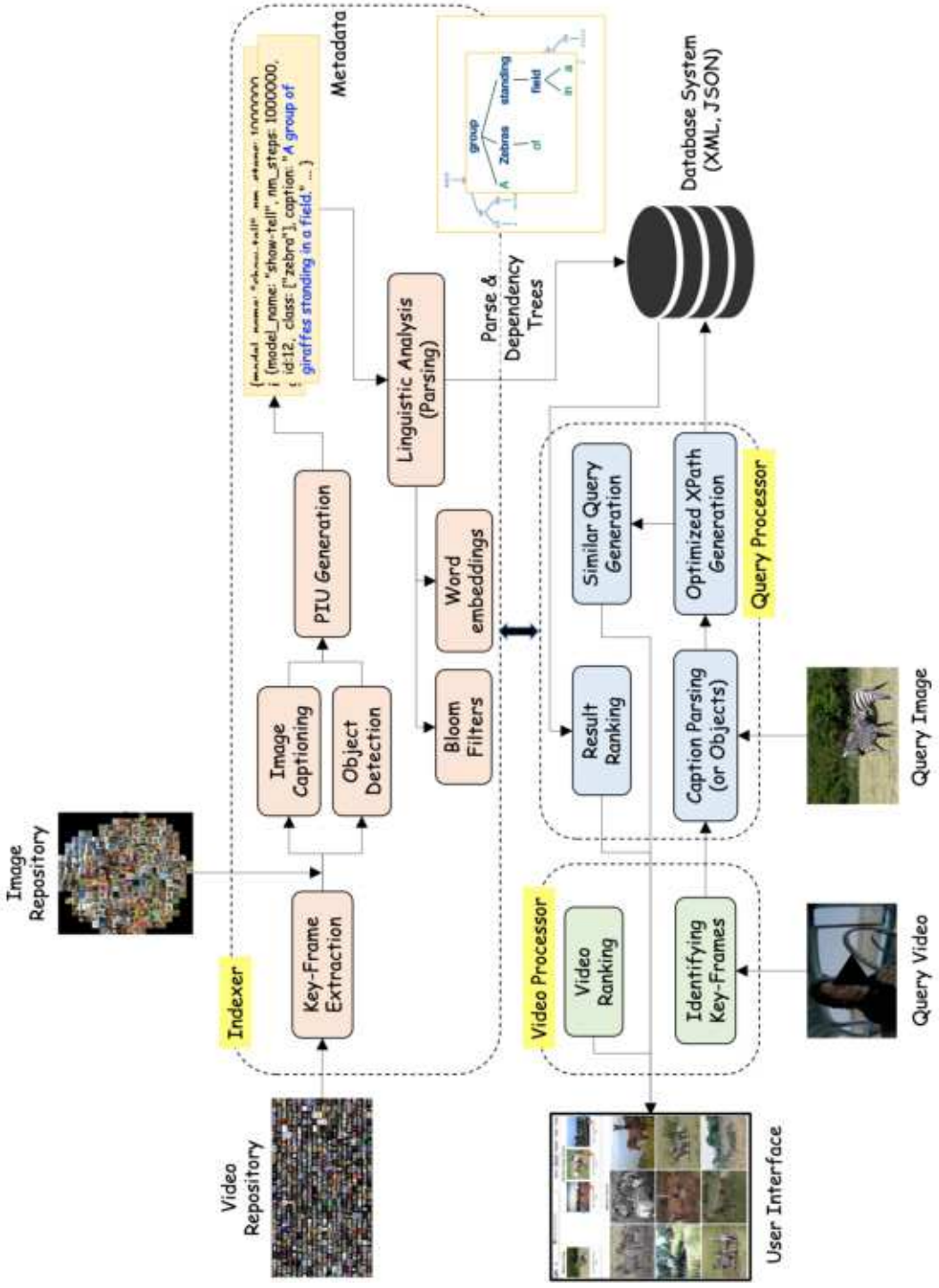


Figure 5.1: Architecture of QIK

PIUs for each image. The PIU constructed for an image contains the most probable captions and objects detected in that image. With the help of the captions and objects detected, we can accurately capture the context of everyday scenes and learn the relationships between objects in them. For each image caption, the Indexer constructs a sentence parse tree and a dependency tree [4]. The entire collection of trees, which are ordered trees, are represented in XML [13]. XML is the appropriate choice as it preserves the ordering. The XML documents are stored and indexed using an XML database system. The other contents of the PIUs (*i.e.*, detected objects, probability scores for captions generated and objects detected, etc.) are also indexed. To enable fast generation of similar image queries, the Indexer constructs word embeddings by training on the image captions using word2vec [28]. The Indexer maintains Bloom filters for each POS tags such as `VBG`, `NN`, `JJ`, and others in order to quickly test which words appear under these tags for all the image captions in the repository. Using Bloom filters allows the Indexer to search an element within the collection in constant time.

5.1.2 Query Processor

The Query Processor uses image captions or detected objects to retrieve and present the user with a precise set of images. In the first case, given a query image, the Query Processor generates the most probable caption (using the same image captioning model) and the associated parse tree and dependency tree. After removing the prepositions, determiners, conjunctions, etc., in the parse tree, the Query Processor processes it to generate an optimized XPath query [14] containing only essential keywords in the caption while at the same time, preserving the ordering between these

keywords and their relationships. After executing the XPath query on the XML documents, a set of candidate images are retrieved. As the database system does not incorporate a ranking mechanism while returning a set of candidate images, the next step performed by the Query Processor is ranking. In order to rank and return the top-k relevant images, the Query Processor leverages the tree edit distance [17] metric. The tree edit distances between a candidate image caption's parse tree (or dependency tree) and the parse tree (or dependency tree) of the query image's caption is computed. The candidate images are then ranked in increasing order of the computed tree edit distance, and the top-k matches are returned.

In the second case, given a query image, the Query Processor detects objects in an image (using an object detection model) with a probability greater than a user-configured threshold. A set of candidate images that contain any of the objects that were detected in the query image are retrieved. A score is assigned to each candidate image based on the combined probability scores of those objects that are detected in the candidate image which are also present in the query image. The candidate images are then ranked in decreasing order of this score, and the top-k matches are returned.

To suggest similar image queries for a query image, the Query Processor generates different XPath queries, using the optimized XPath query, by replacing the XPath text nodes with similar words. To obtain words similar to a given word, its nearest neighbors are computed using word embeddings constructed on image captions. This can lead to an exponential number of possibilities. Hence, we limit them to a user-configured threshold. Bloom filters are then checked based on the XML element name enclosing the word to ensure that replacing a word with a similar word will yield an XPath query that produces a non-empty result. Note that this process uses only the

filtering step but not the ranking step.

5.1.3 Video Processor

Given a video repository, the Video Processor extracts the key scenes contained in a video. For each scene extracted by the Video Processor, PIUs are generated by the Indexer (as described in section 5.1.1). The collection of PIUs generated are represented as XML documents and stored and indexed using an XML database system. For a query video, the Video Processor identifies the key scenes and sends them over to the Query Processor for further processing. The Query Processor uses the image captions generated for each scene and performs linguistic analysis to return a set of candidate scenes and videos (as described in section 5.1.2). The Video Processor ranks the candidate videos returned by the Query Processor based on the number of candidate scenes that match the query scenes. The candidate videos are ranked in an increasing order of scene matches, and the top-k matches are returned.

5.2 Image PIU Generation

`QIK` generates PIUs for each image in the image repository by leveraging the predictions made by deep neural networks developed for image understanding tasks. For generating image captions, each image is passed through a pre-trained image captioning model to output the most probable captions. The captions predicted depict the relationship between objects present in an image and accurately capture the context of that image. In addition to it, each image is also passed through a pre-trained object detection model to identify the most probable objects in the image (with a

probability score exceeding a user-configured threshold). Along with the other attributes, the bounding boxes obtained from the object detection model, probability scores output by the models, etc. can also be stored as part of the image's PIU. The PIU generated from different models are queried together during image retrieval.

Algorithm 1: IndexPIU(*img*)

Input: *img* denotes an image in the database

```
1  Predict the most probable captions C of img
2  Predict the most probable objects O in img with probability greater than a
     user-defined threshold
3  for each caption c ∈ C do
4  |  Generate the parse tree p for c
5  |  Generate the dependency tree d for c
6  |  Represent p in XML
7  |  Represent d in XML
8  |  Store and index the XML documents in the database system
9  end for
10 for each object o ∈ O do
11 |  Represent o as a JSON record containing the probability of o
12 |  Store and index the JSON record in the database system
13 end for
```

Algorithm 1 outlines the steps involved in generating, analyzing, and indexing the PIUs by the `QIK` Indexer. For each image in the repository, the most probable captions are generated and their parse and dependency trees are constructed and stored in an XML database. The objects detected in the image with a probability

greater than a user-specified threshold are also indexed.

Figure 5.2: An example image

Let us now look in detail at how `QIK` generates the PIU an image and performs linguistic analysis of the PIU for indexing using an example. Consider the image shown in Figure 5.2. We first pass the image through an image captioning model. Out of the three captions produced by an image captioning model, we choose the caption with the highest probability score (a.k.a the most probable caption). The most probable caption is "a young boy kicking a soccer ball on a field." Suppose an object detection model identifies two objects with probabilities greater than 0.9. Suppose these two objects are "soccer ball" and "person". Together, they constitute the PIU of the image.

Linguistic analysis of the image PIUs are performed by the `QIK` indexer using state-of-the-art NLP techniques. Each image captions are analyzed linguistically to constructs its sentence parse tree and dependency tree. A *parse tree* represents the structure of the sentence/phrase based on the grammar rules by identifying tokens denoting noun phrases, nouns, verb phrases, verbs, adjectives, determiners, preposi-

tions, etc., in the sentence/phrase. These tokens are based on parts-of-speech (POS) tagging in computational linguistics [4]. A *dependency tree* on the other hand provides a representation of how words in a sentence/phrase are connected by syntactic dependencies [4].

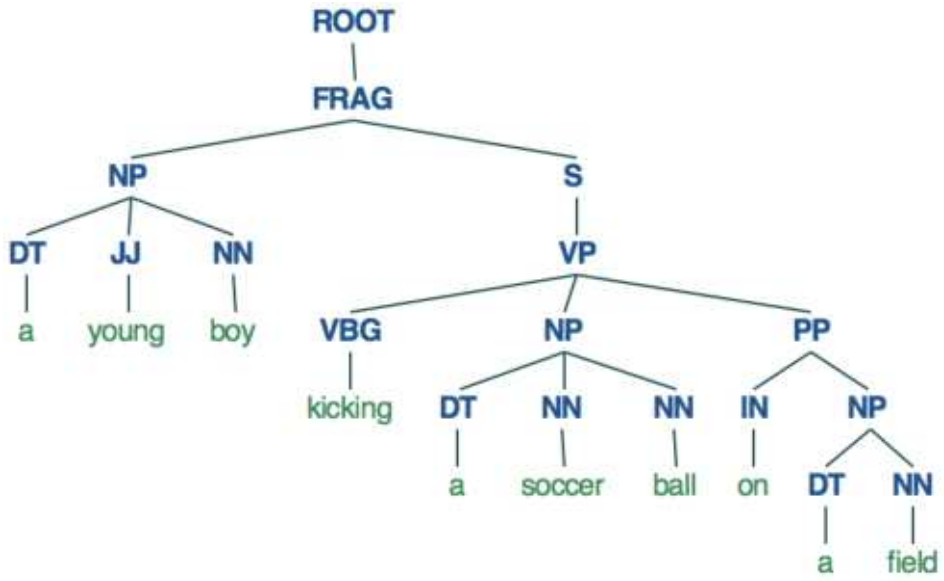


Figure 5.3: Parse tree

```
<ROOT><FRAG><NP><DT>a</DT><JJ>young</JJ><NN>boy</NN></NP><S><VP>
<VBG>kicking</VBG><NP><DT>a</DT><NN>soccer</NN><NN>ball</NN></NP>
<PP><IN>on</IN><NP><DT>a</DT><NN>field</NN></NP></PP></VP></S>
</FRAG></ROOT>
```

Figure 5.4: Parse tree XML representation

Consider the most probable caption obtained for the image shown in Figure 5.2: "a young boy kicking a soccer ball on a field." Figure 5.3 shows the parse tree of the caption output by the Stanford Parser [92]. Figure 5.5 shows the corresponding dependency tree output by the Stanford Parser [93]. Figure 5.4 shows the XML

document for the parse tree in Figure 5.3. The leaf nodes of the parse tree are represented as text nodes in XML. The dependency tree is also represented similarly in Figure 5.6. However, the leaf nodes are treated as XML elements.

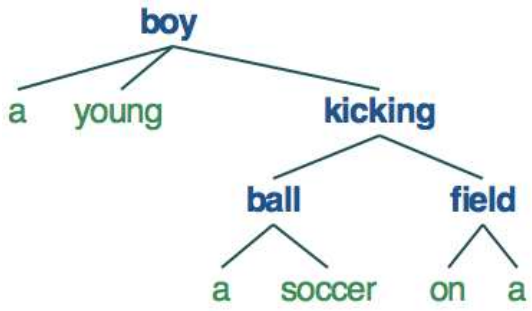


Figure 5.5: Dependency tree

```
<boy><a/><young/><kicking><ball><a/><soccer/></ball><field><on/>
<a/></field></kicking></boy>
```

Figure 5.6: Dependency tree XML representation

XML is the appropriate choice for persisting and querying over the parse and dependency trees constructed as an XML representation preserves the ordering between the nodes of the tree nodes. While the other contents of the PIU (detected objects, their bounding boxes, etc.) could be stored, indexed, and retrieved using an XML database, it is beneficial to store as a JSON and queried using a JSON database. This is because storing them as key-value pairs would consume less storage space compared to storing them as an XML. Thus, `QIK` is designed to be a generic framework that can accommodate multiple image understanding models for many information retrieval tasks. Both the JSON and XML documents representing the PIUs are stored and indexed to enable fast query processing.

5.3 Retrieval Using Captions in PIUs

In this section, we see how the `QIK` Query Processor processes a query image using image captions and returns the top-k relevant matches to the user.

Algorithm 2: RetrieveImages(k, img_q)

Input: k denotes the maximum number of matches to return

Input: img_q denotes the query image

1 Predict the most probable caption C for img_q

2 Predict the most probable objects O in img_q

3 $q \leftarrow GenerateBasicXPath(C)$

4 $q' \leftarrow GenerateOptimizedXPath(q)$

5 Generate a query q'' for O

6 Execute q' on the XML database to obtain set of image IDs

7 Execute q'' on the JSON database to obtain set of image IDs

8 Compute the intersection of the above two sets to obtain the candidate images

9 **for** *each candidate image* img_c **do**

10 Compute tree edit distance between the parse tree (or dependency tree) of the caption of img_c and the parse tree (or dependency tree) of the caption C

11 **end for**

12 Sort (in ascending order) the candidate images based on the computed tree edit distance values

13 **return** *top-k matches*

Algorithm 2 outlines the overall steps during image retrieval. Given a query image, its captions are first predicted using the same pre-trained image captioning model that was used during indexing. The most probable objects with probability greater than a user-defined threshold are also predicted. A basic XPath query is constructed for the predicted caption by invoking Algorithm 3. This query is optimized further by invoking Algorithm 4 to generate an optimized XPath query. The optimized XPath query is executed on the XML database to obtain an initial set of candidate images. In addition, a query comprising of the objects detected is generated and executed on a JSON database to retrieve another set of candidate images. An intersection of the two sets gives us the final set of candidate images. These images would contain all objects that are present in the query image as well as match the criteria specified by the XPath query constructed from the captions. The tree edit distance between the parse trees (or dependency trees) of the candidate and query image pairs is computed. The candidate images are then sorted in ascending order of the computed tree edit distance values to return the top-k matches to the user.

Next, we discuss how a basic XPath query is constructed for a given caption (Algorithm 3) and how it is optimized (Algorithm 4) for efficient retrieval. In Algorithm 3, a query caption is first parsed into its parse tree. The tree is pruned to remove non-essential keywords such as "on", "a", "in", and others (Line 2). By clearing the tree of prepositions, determiners, conjunctions, etc., we reduce the size of the tree without losing out on the relationship between the objects, hence preserving the knowledge of the context of that image. The pruned tree is traversed in a preorder fashion (Lines 3-16) to output the basic XPath query containing XPath axes such as `child`, `following`, and `following-sibling`.

Algorithm 3: GenerateBasicXPath(C)

Input : C denotes the image caption

Output: An XPath expression

```
1  Let D denote the parse tree of C
2  Prune D by removing subtrees rooted at POS tags such as DT, IN, and other
     non-essential keywords
3  for each node n in preorder traversal of D do
4      Let t denote node label of n
5      if n is the root node of D then
6          q ← /child::t
7      else if n is child of the previous node (in preorder) then
8          Append /child::t to q
9      else if n is a sibling of the previous node (in preorder) then
10         Append /following-sibling::t to q
11     else if n is a leaf node then
12         Append [text()="t"] to q
13     else
14         Append /following::t to q
15     end if
16 end for
```

Let us consider the caption predicated for image in Figure 5.2 "a young boy kicking a soccer ball on a field." A parse tree is constructed as shown in Figure 5.3. Algorithm 3 produces the basic XPath query as shown in Figure 5.7.

From the basic XPath query, an optimized query is generated following Algo-

```
/child::ROOT/child::FRAG/child::NP/child::JJ[text()="young"]/
following-sibling::NN[text()="boy"]/following::S/
child::VP/child::VBG[text()="kicking"]/
following-sibling::NP/child::NN[text()="soccer"]/
following-sibling::NN[text()="ball"]/following::PP/
child::NP/child::NN[text()="field"]
```

Figure 5.7: Basic XPath query

rithm 4. A variable `leadingAxis` is defined that tracks the axis encountered. The XPath query is traversed from left-to-right, one axis-node pair at a time. When the axis is one of `following` or `following-sibling`, the `leadingAxis` is replaced with that axis (Lines 14 - 15). Each time a node containing a predicate is encountered, an XPath axis and node name are appended to the optimized query (Lines 3 - 11). The axis type is `following`, if the `leadingAxis` is `following` or `following-sibling` (i.e The axis type depends on the sequence of axes encountered since the previous node with a predicate) (Lines 4 - 7). When only one axis appears (*e.g.*, `child`, `following`, `following-sibling`) since the previous node with a predicate (*e.g.*, an adjective and its noun), it is replaced by that axis (Line 9). The child axes without a predicate are ignored (Lines 12 - 13). The first axis of the generated query (`child`) is replaced finally by `descendant` (Line 21). The main idea behind generating an optimized XPath query is to replace a sequence of XPath axes with a single axis that still specifies the same constraint on the keywords as the original query while reducing the length of the XPath query in terms of the number of nodes and axes. Algorithm 4 transforms the basic XPath query shown in Figure 5.7 into an optimized XPath query (Figure 5.8).

To summarize, Algorithm 2 comprises of a filtering step (Lines 1 - 8) and a re-

Algorithm 4: GenerateOptimizedXPath(q)

Input : q denotes an input XPath expression

Output: Optimized XPath expression

```
1  q' ← NULL; leadingAxis ← NULL
2  for each axis x and node n in q (from left-to-right) do
3      if n has predicate p then
4          if leadingAxis is following then
5              Append /following::n[p] to q'
6          else if leadingAxis is following-sibling then
7              Append /following::n[p] to q'
8          else
9              Append /x::n[p] to q'
10         end if
11         leadingAxis ← NULL
12     else if x is child then
13         continue
14     else if x is following::sibling or following then
15         leadingAxis ← x
16     else
17         print("Invalid axis")
18         return NULL
19     end if
20 end for
21 Replace the first axis in q' with descendant
22 return q'
```

```
/descendant::JJ[text()="young"]/following-sibling::NN[text()="boy"]/
following::VBG[text()="kicking"]/following::NN[text()="soccer"]/
following-sibling::NN[text()="ball"]/following::NN[text()="field"]
```

Figure 5.8: Optimized XPath query

ranking step (Lines 9 - 12). During the filtering step, linguistic analysis is performed on the query images caption to construct an optimized XPath query which is further executed on the XML database to obtain a set of candidates. During the ranking step, the candidate images obtained are ranked in an increasing order of the tree edit distance between the parse tree of the candidate image's caption and the parse tree of the query image's caption to return the top-k matches.

Figure 5.9 illustrates the execution plan following the above steps for another query image.

5.4 Retrieval Using Detected Objects

In this section, we show how `QIK` processes a query image using the objects detected in them to return the top-k relevant matches to a user.

Algorithm 5 sketches the overall steps for retrieving objects using detected objects. For a given query image, all objects contained in them are detected using an object detection model. Then a Boolean AND query is constructed using those objects that have a probability value greater than a user-specified threshold. The JSON database is then queried to fetch all the candidate images that contain every object in the query. Candidate images are ranked based on the sum of the product of the probabilities of the selected objects in the candidate image and the query image. Finally, the images

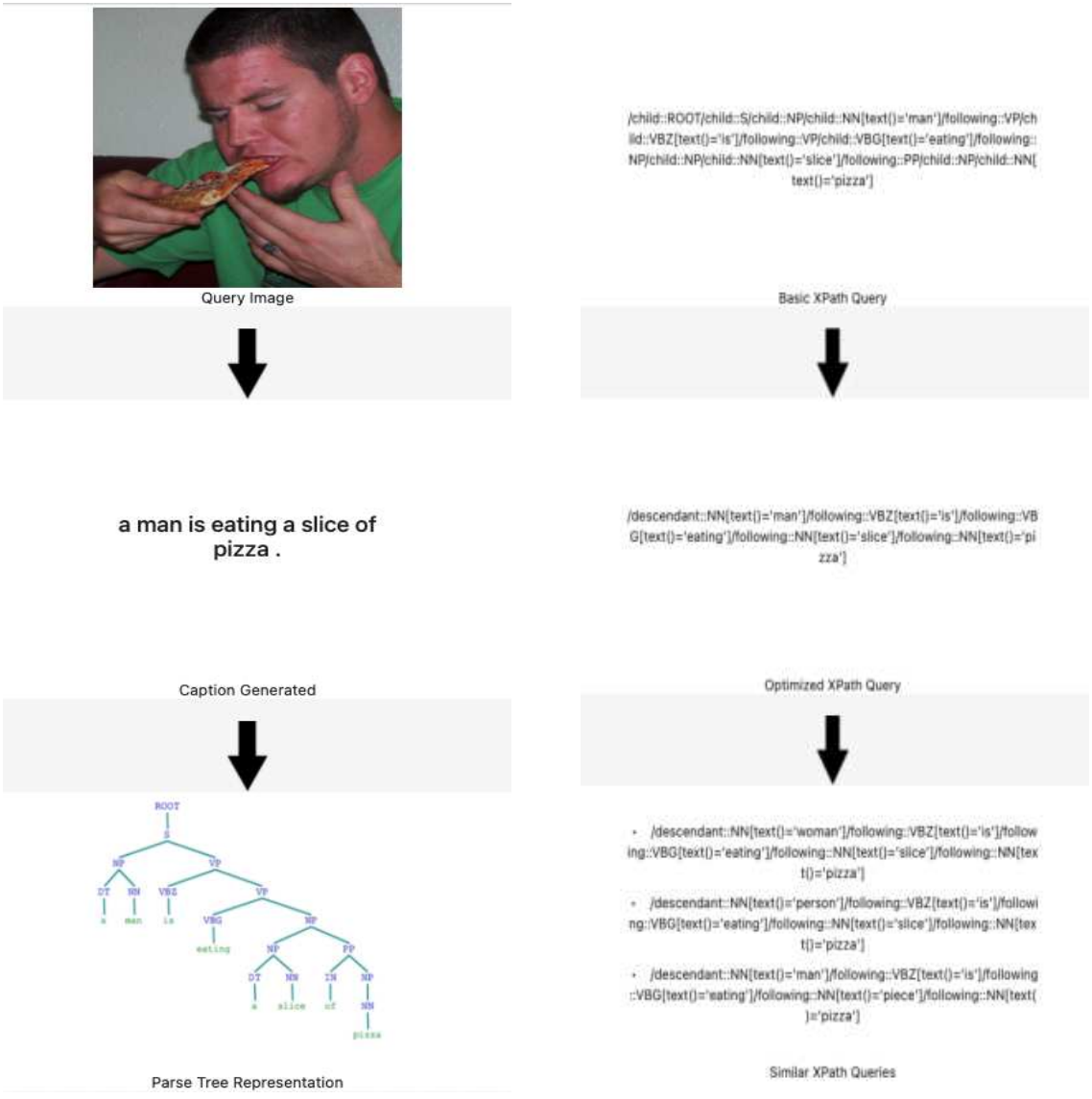


Figure 5.9: Execution plan

Algorithm 5: RetrieveImagesObj(k, img_q)

Input : k denotes the maximum number of matches to return

Output: img_q denotes the query image

1 Let $O = \{(o_1, p_1), (o_2, p_2), \ldots, (o_n, p_n)\}$ denote the most probable objects and their probabilities in img_q such that p_i is greater than a user-specified threshold

2 Generate a query q to specify a Boolean AND of the objects $o_1, o_2, \ldots, o_n$

3 Execute q on the JSON database to obtain set of image IDs along with the probabilities of the objects in each candidate image

4 **for** *each candidate image* img_c **do**

5 Let $O_c = \{(o_1, r_1), (o_2, r_2), \ldots, (o_n, r_n)\}$ denote the matched objects and their probabilities in the candidate image

6 score$(img_c) = \sum_{i=1}^{n} p_i \times r_i$

7 **end for**

8 Sort (in descending order) the candidate images based on their scores

9 **return** *top-k matches*

are sorted in descending order by their scores, and the top-k results are returned. Algorithm 5 also comprises of the filtering step (Lines 1 - 3) wherein objects are detected to form a query in order to obtain a set of candidate images and a ranking step (Lines 4 - 8) where the candidate images are ranked based on the score computed from the object detection probability values.

Algorithm 6: IndexPIU(*video*)

Input: *video* denotes a video in the database

```
Extract the key scenes scenes_video of video
for each scene ∈ scenes_video do
    Predict the most probable caption c of scene
    Generate the parse tree p for c
    Generate the dependency tree d for c
    Represent p in XML
    Represent d in XML
    Store and index the XML documents along with in the database system
end for
```

5.5 Video PIU Generation

QIK extends PIUs generation to videos by identifying and extracting key frames from them. Given a video repository, key frames are identified for each video, and PIUs are generated by leveraging the predictions made by deep neural networks developed for image understanding tasks. The most probable captions are predicted for each frame using an image captioning model that constitutes the PIUs for that frame. The key frame PIUs together form the PIU of the video. The PIUs of the key frames are further analyzed linguistically by constructing the parse trees and dependency trees of the captions. The parse trees and dependency trees obtained are represented as an XML. The XML documents are stored and indexed in a database system. The sets of steps involved in generating a PIU for a video is listed under Algorithm 6.

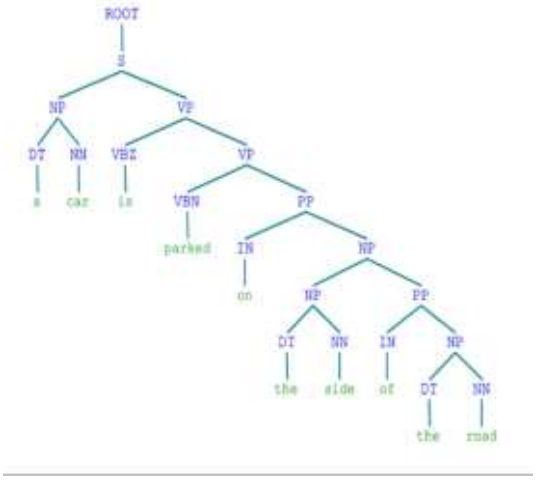
For example, consider a video represented as a sequence of frames as shown in Fig-

Figure 5.10: An example video

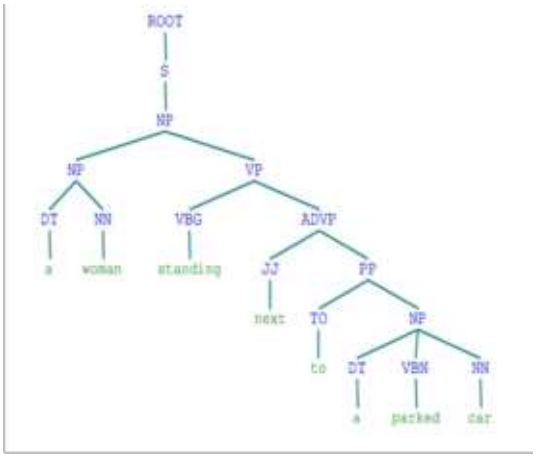
(a)

(b)

Figure 5.11: Example key frames

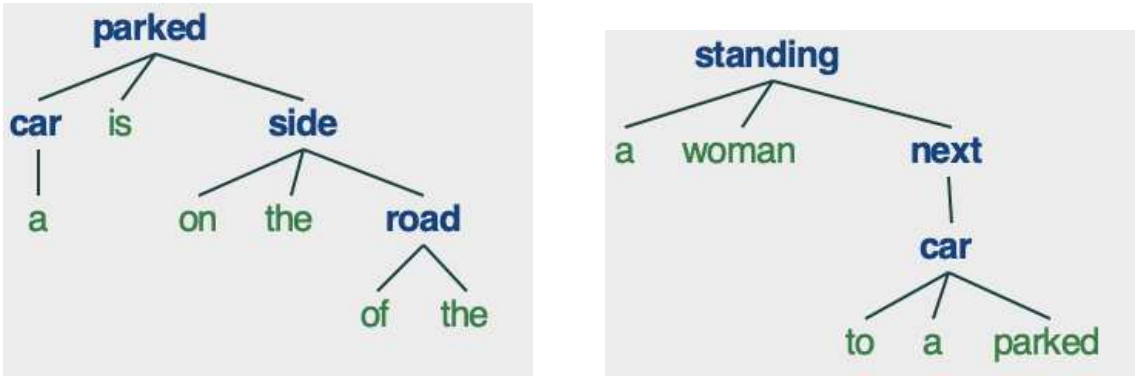


(a)



(b)

Figure 5.12: Example parse tree representations



(a) (b)

Figure 5.13: Example dependency tree representations

ure 5.10. The key frames are identified and and most probable captions are obtained. The key frames extracted for the query video are shown in Figures 5.11a and 5.11b. The captions produced for these frames by an image captioning model are "a car is parked on the side of the road" and "a woman standing next to a parked car". The captions constitute the PIUs of key frames. Together they form the PIU of the given video. The most probable captions for each frame are analysed linguistically analysed by the Query processor. For each captions a parse trees (seen in Figures 5.12a and 5.12b) and dependency trees (seen in Figures 5.13a and 5.13a) are constructed. The constructed parse trees and dependency trees are represented as as XML (seen in Figures 5.14 and 5.15) and stored and indexed in an XML database system.

5.6 Video Retrieval Using PIUs

In this section, we show how `QIK` processes a query video to return the top-k relevant matches to the user.

Algorithm 7 outlines the overall steps involved in video retrieval. Given a query

```
<ROOT><S><NP><DT>a</DT><NN>car</NN></NP><VP><VBZ>is</VBZ><VP>
<VBN>parked</VBN><PP><IN>on</IN><NP><NP><DT>the</DT><NN>side</NN>
</NP><PP><IN>of</IN><NP><DT>the</DT><NN>road</NN></NP></PP></NP>
</PP></VP></VP></S></ROOT>
```

(a)

```
<ROOT><S><NP><NP><DT>a</DT><NN>woman</NN></NP><VP>
<VBG>standing</VBG><ADVP><JJ>next</JJ><PP><TO>to</TO><NP>
<DT>a</DT><VBN>parked</VBN><NN>car</NN></NP></PP></ADVP>
</VP></NP></S></ROOT>
```

(b)

Figure 5.14: Parse tree XML representations

video, the key frames are first identified (Line 1). For each of the identified key frames, a caption is predicted using the same pre-trained image captioning model that was used during indexing. A basic XPath query is generated using Algorithm 3. The query is optimized further by invoking Algorithm 4. The optimized XPath query is executed on the database to obtain a set of frames of qualifying videos IDs (Lines 2 - 8).

```
<parked><car><a/></car><is/><side><on/><the/>
<road><on/><the/></road></side></parked>
```

(a)

```
<standing><a/><woman/><next><car><to/><a/>
<parked/></car></next></standing>
```

(b)

Figure 5.15: Dependency tree XML representations

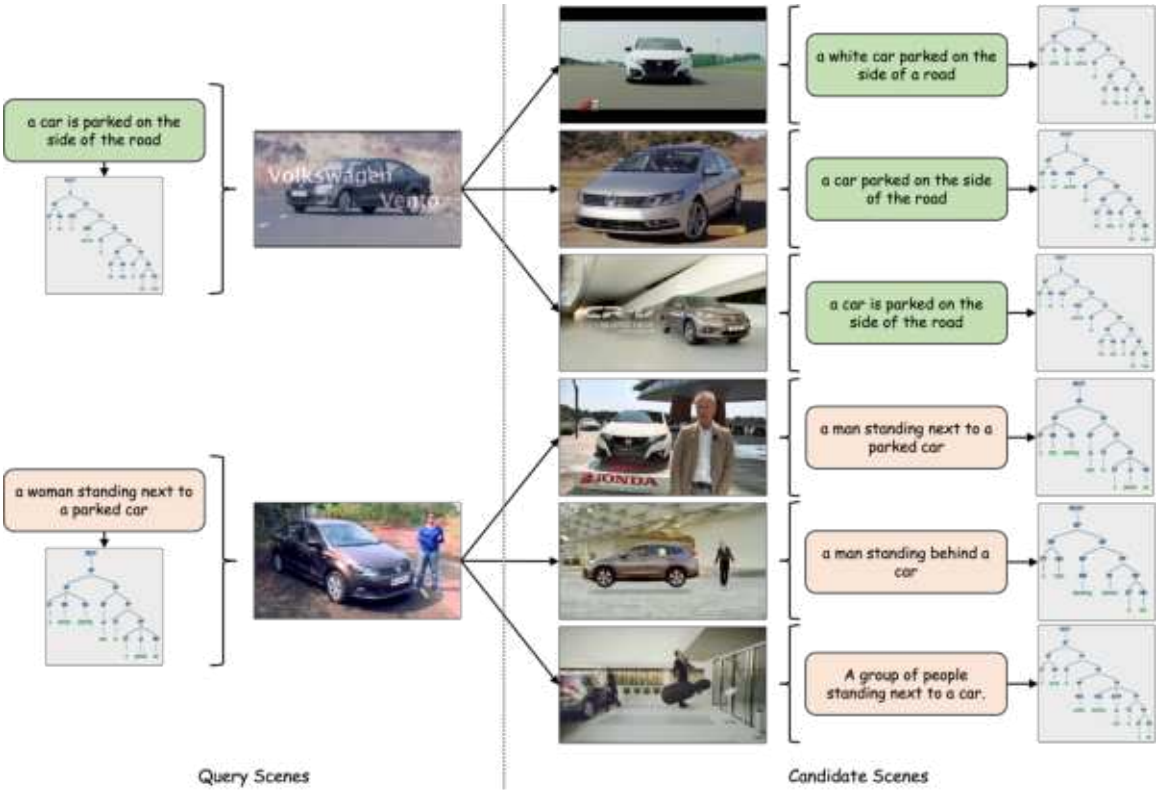

a car is parked on the side of the road
a woman standing next to a parked car
a white car parked on the side of a road
a car parked on the side of the road
a car is parked on the side of the road
a man standing next to a parked car
a man standing behind a car
A group of people standing next to a car.
Query Scenes
Candidate Scenes

Figure 5.16

Figure 5.17

Algorithm 7: RetrieveVideos(k, $video_q$)

Input: k denotes the maximum number of matches to return

Input: $video_q$ denotes the query video

```
Extract the key frames from video_q
for each query frame frame_q do
    Predict the most probable caption C for frame_q
    q ← GenerateBasicXPath(C)
    q' ← GenerateOptimizedXPath(q)
    Execute q' on the XML database to obtain candidate frames
    Execute q'' on the JSON database to obtain candidate frames
end for
sceneDictionary ← NULL
for each frame ID frameID do
    for each video ID videoID do
        if videoID is present in sceneDictionary then
            Append frameID to sceneDictionary[videoID]
        else
            sceneDictionary[videoID] ← frameID
        end
    end for
end for
RankCandidateVideos(video_q, sceneDictionary)
return top-k matches
```

Consider the video represented shown in Figure 5.10. The key frames are extracted and passed through the Query Processor to return a set of candidate frames. The set of candidate frames for each query frame is shown in Figure 5.16. From the frames returned by the Query Processor, the frames corresponding to the same video are grouped together (Lines 9-18), as seen in Figure 5.17.

After retrieving all sets of frames IDs and videos IDs, they are grouped based on query frame IDs (Lines 9-18). The candidate videos are further ranked to return the top-k matches to the user.

5.7 Candidate Video Ranking

Next, we discuss the details of the ranking step performed during video retrieval (Line 19, Algorithm 7). We propose four different ranking procedures detailed below that illustrates the significance of frame ordering and preserving the ordering between essential keywords in captions and their relationships.

5.7.1 Ranking Based on Matching Frames

In this approach, listed in Algorithm 8, we score each candidate video based on the number of frames that match the query video frames. The candidate videos are ranked in decreasing order of their scores to return the top-k matches to the user.

For example, in Figure 5.17, the number of matching frames for the candidate videos 1, 2, and 3 are 2, 1, and 3, respectively. Hence the scores assigned to the candidate videos are 2, 1, and 2, respectively. Sorting the candidate videos in decreasing order of the scores ranks candidate videos 3 the first, followed by candidate video 1

Algorithm 8: RankCandidateVideos($video_q$, $sceneDictionary$)

Input: $video_q$ denotes the query video

Input: $sceneDictionary$ denotes the candidate frame grouping order

for *each candidate video* $video_c$ **do**

  $score \leftarrow len(sceneDictionary[video_c])$

end for

Sort the candidate videos based on *score* (in descending order)

and candidate video 2.

5.7.2 Ranking based on the Longest Common Subsequence of Frames

Algorithm 9: RankCandidateVideos($video_q$, $sceneDictionary$)

Input: $video_q$ denotes the query video

Input: $sceneDictionary$ denotes the candidate frame grouping order

for *each candidate video* $video_c$ **do**

  Let $scenes_{lcs}$ denote the Longest Common Subsequence of scenes within the candidate and query scene sequences

  $score_{lcs} \leftarrow length(scenes_{lcs})$

end for

Sort the candidate videos based on $score_{lcs}$ (in descending order)

We now take into account ordering of the frames. For each candidate video, we first compute the Longest Common Subsequence of frames within the candidate frame sequences that match the query frame sequences. They are then assigned a score based

on the length of the Longest Common Subsequence. Finally, the candidate videos are ranked in decreasing order of their scores to return the top-k matches to the user. Algorithm 9 lists the steps involved in this approach.

For example, in Figure 5.17, the length of the Longest Common Subsequence of frames for the candidate videos 1, 2, and 3 are 2, 1, and 1, respectively. Sorting the candidate videos in decreasing order of the scores ranks candidate video 1 the first, followed by candidate videos 2 and 3.

5.7.3 Ranking based on the Longest Common Subsequence of Frames and Caption Tree Edit Distances

In this approach, in addition to taking into account the ordering of the frames, we also consider the ordering between essential keywords in captions and their relationships. Algorithm 10 lists the steps involved in this approach. For each candidate video, we first compute the Longest Common Subsequence of frames within the candidate frame sequences that match the query frame sequences. Let this length be represented as $score_{lcs}$ (Lines 3-4). Next, from amongst the longest subsequence, we compute the tree edit distance between the parse tree (or dependency tree) representation of the frame caption and the query caption to obtain $score_{ted}$ (Lines 5-7). We rank the candidate videos in decreasing order of the length of the longest common subsequences of frames and increasing order of the tree edit distance computed to return the top-k matches to the user.

For example, in Figure 5.17, the length of the longest common subsequences of frames ($score_{lcs}$) for the candidate videos 1, 2, and 3 in Figure 5.17 are 2, 1, and 1, respectively. The tree edit distances computed with the parse tree representation

Algorithm 10: RankCandidateVideos($video_q$, $sceneDictionary$)

Input: $video_q$ denotes the query video

Input: $sceneDictionary$ denotes the candidate frame grouping order

for *each candidate video* $video_c$ **do**
  $score_{ted} \leftarrow 0$
  Let $scenes_{lcs}$ denote the Longest Common Subsequence of scenes within the candidate and query scene sequences
  $score_{lcs} \leftarrow length(scenes_{lcs})$
  for *each candidate scene* $scene_{cand}$ *and query scene* $scene_q$ *in* $scenes_{lcs}$ **do**
    Compute tree edit distance between the parse tree (or dependency tree) of the caption of $scene_{cand}$ and the parse tree (or dependency tree) of the caption $scene_q$
    $score_{ted} \leftarrow \frac{1}{1+candidate_{ted}}$
  end for
end for
Sort the candidate videos based on $score_{lcs}$ and $score_{ted}$ (in descending order)

of the frame captions for the candidate videos 1, 2, and 3 (seen in Figure 5.16) are 0.2, 0.2, and 0.1, respectively. Ranking the candidate videos in decreasing order of the length of the longest common subsequences of frames makes candidate video 1, with $score_{lcs}$ as 2 to be ranked first, followed by candidate videos 2 and 3 each with $score_{lcs}$ 1. And, ranking based on the $score_{ted}$ computed re-ranks candidate videos 2 and 3.

5.7.4 Ranking based on Scene Matching, Longest Common Subsequence of Frames and Caption Tree Edit Distances

Algorithm 11: RankCandidateVideos($video_q$, $sceneDictionary$)

Input: $video_q$ denotes the query video

Input: $sceneDictionary$ denotes the candidate frame grouping order

1 **for** *each candidate video* $video_c$ **do**

2 $score \leftarrow len(sceneDictionary[video_c])$

3 $score_{ted} \leftarrow 0$

4 Let $scenes_{lcs}$ denote the Longest Common Subsequence of scenes within the candidate and query scene sequences

5 **for** *each candidate scene* $scene_{cand}$ *and query scene* $scene_q$ *in* $scenes_{lcs}$ **do**

6 Compute tree edit distance between the parse tree (or dependency tree) of the caption of $scene_{cand}$ and the parse tree (or dependency tree) of the caption $scene_q$

7 $score_{ted} \leftarrow score_{ted} + \frac{1}{1+candidate_{ted}}$

8 **end for**

9 **end for**

10 Sort the candidate videos based on $score_{ted}$ and $score$ (in descending order)

In this approach, each candidate videos are scored based on the number of frames that match the query video frames, represented (Line 2). Further, the Longest Common Subsequence of frames within the candidate frame sequences that match the query frame sequences are computed. Let this length be represented as $score_{lcs}$ (Lines 4). From amongst the longest subsequence, we compute the tree edit distance

between the parse tree (or dependency tree) representation of the frame caption and the query caption to obtain $score_{ted}$ (Lines 5-7). Finally, we rank the candidate videos in increasing order of the tree edit distance computed and in decreasing order of the number of matching frames to return the top-k matches to the user.

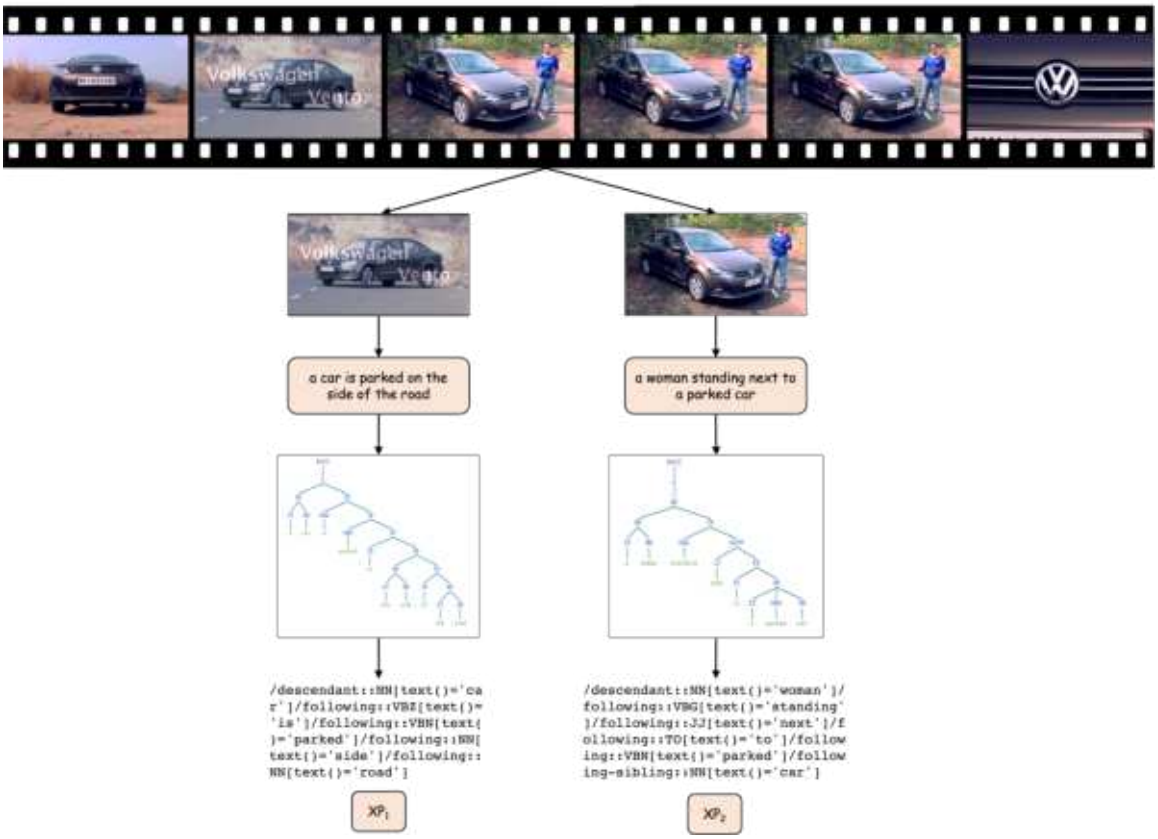


Figure 5.18: Video retrieval using PIUs - 1

For example, in Figure 5.17, as seen earlier, the number of matching frames for the candidate videos 1, 2, and 3 are 2, 1, and 2, respectively. The length of the longest common subsequences of frames ($score_{lcs}$) for the candidate videos 1, 2, and 3 in Figure 5.17 are 2, 1, and 1, respectively. The tree edit distance computed with the parse tree representation of the frame captions for the candidate videos 1, 2, and

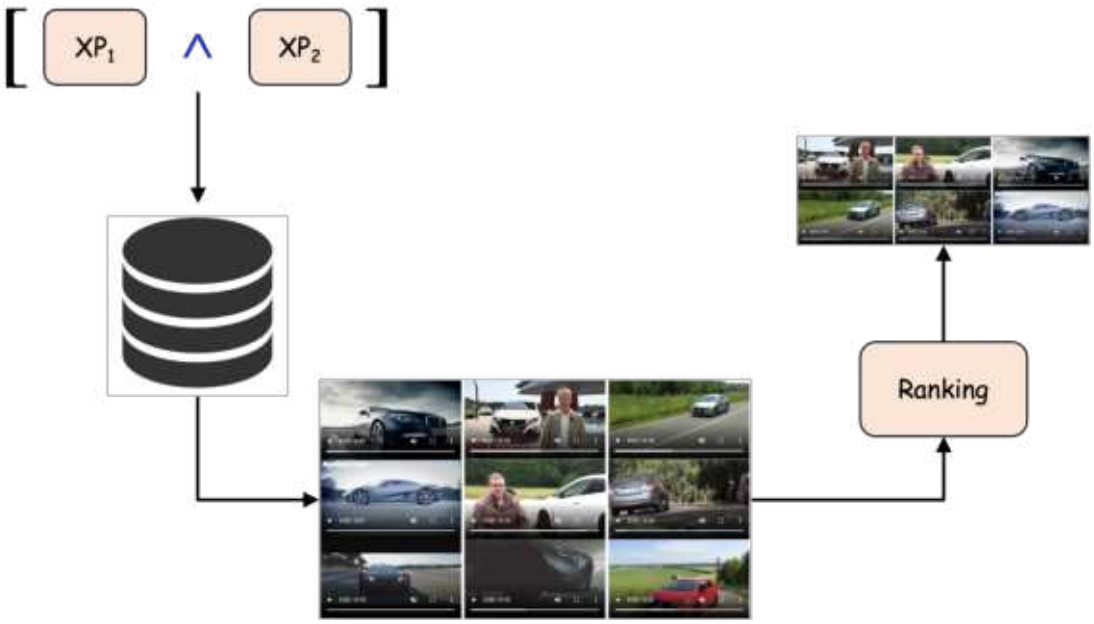


Figure 5.19: Video retrieval using PIUs - 2

3 are 0.2, 0.2, and 0.1, respectively. Ranking the candidate videos in decreasing order of $score_{ted}$ ranks candidate videos 1 and 2 the first, followed by candidate video 3. Candidate videos 1 and 2, having the same ranks, are further re-ranked on *score* to rank Candidate video 1 over 2.

The key idea is to rank those candidate videos that match the order of events in the query video higher up amongst the other candidates. And, amongst the candidate videos having the same order of events, we re-rank them based on the similarity of the query frame captions, essentially ranking those videos with frames that are the most similar to the query frame higher.

To summarize, given a query video, key frames are extracted. The most probable captions for each frame are analysed linguistically analysed by the Query processor to construct parse tree representations. The parse trees are then transformed to a

basic XPath query and further pruned to obtain an optimized XPath query. The optimized XPath queries are aggregated and executed on the database to retrieve a set of frames, corresponding to a set of candidate videos. Finally the candidate videos are ranked based on a combination of the number of matching frames retrieved for the query frame captions, the longest common subsequence of frame matches, and the tree edit distance between the candidate and query frame captions, to return the top-k matches to the user. Figures 5.18 and 5.19 illustrates the execution following the above steps for the query video.

Chapter 6

Implementation

In this chapter, we describe in detail, the implementation details of `QIK`.

`QIK` Indexer was primarily written in Java and compiled using Java 1.8[1]. Java was the language of choice as it is perfect for developing large-scale web applications that are distributed over a large number of systems. Java also enables easy connectivity to database systems via Java Database Connectivity (JDBC) Application Programming Interface (API)[2] making it the most ideal candidate. The parse trees and dependency trees for captions were generated using the Stanford Parser package[3] (version 3.9.2). The compiled application was deployed as a web service on Apache Tomcat 9.0.20[4], a popular web server and servlet container for Java applications.

XML data was stored and indexed using BaseX [94](version 9.2[5]). BaseX is an open-source high-performance XML database engine apt for enabling complex data-

[1] `https://www.java.com/en/`
[2] `https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc`
[3] `https://nlp.stanford.edu/software/lex-parser.shtml`
[4] `http://tomcat.apache.org/`
[5] `https://files.basex.org/releases/9.2/`

intensive web applications. It is lightweight and scalable, enabling storage, querying, and processing of large volumes of textual (XML, HTML, JSON, CSV, etc.) and binary data. It comes with a highly optimized XQuery 3.1 Processor with full support for W3C Update and Full-Text extensions. During ranking, `QIK` used the `apted` python package[6] (version 1.0.3) for computing tree edit distance between the query image's parse tree (or dependency tree) and the candidate image's parse tree (or dependency tree). The `apted` package is an implementation of APTED [21, 22], a robust, main-memory approach for computing tree edit distance.

`QIK` Indexer and Query Processor used Show and Tell [31] and the ClipCap [33] models for generating captions of images. Show and Tell uses a CNN image encoder for recognizing the various objects and extracting visual features from an image. An LSTM RNN is then used to transform these features into sentences. We used a pre-trained Inception v3 model [41] used for initializing the parameters of the Show and Tell image encoder. The training was done for 3 million steps on the MS COCO dataset [9], containing 83K training images and 41K validation images. The accuracy of the model was further improved by executing the second round of training for 2 million steps to enable fine-tuning of the Inception v3 model. During fine-tuning, the visual features extracted by the CNNs and the vision and language networks are jointly trained on the human-generated captions made available with the MS COCO dataset. ClipCap combines CLIP [34] with GPT-2 [35] to generate image captions.

The model was trained on the Conceptual Captions [95] dataset consisting of 3.3 million images and their descriptions extracted from the web. During training, the images are passed through CLIP's image encoder network. The output of this

[6] `https://pypi.org/project/apted/`

network is mapped to the embedding space of GPT-2 by a mapping network that again comprises of a trained lightweight transformer network. At the time of inference, the image encodings output by CLIP are passed only through the mapping network to generate a prefix which is then used to predict the next words using beam search. For object detection, Faster-RCNN with NASNet-A image featurization [96] trained on MS COCO was used. Neural Architecture Search Network (NASNet) aims at finding an architectural building block using a small dataset and then transferring this to larger datasets by stacking multiple copies of these blocks. A NASNet model is capable of achieving state-of-the-art results while at the same time having a lower complexity due to its smaller model size.

To construct similar XPath queries, to enable intuitive exploration of the repository, the `QIK` Indexer maintained a Word2Vec [28] word embedding model. This model was trained on all the captions for the images in the repository. Bloom filters [24] were used for storing and testing the existence of the closest words obtained from the model. The `QIK` Indexer used the Bloom filter implementation provided by `guava`[7], a set of core Java libraries from Google. The Indexer maintained separate Bloom filters for each POS tag. Together they enabled the fast generation of similar image queries.

PySceneDetect[8] is a free and open-source software that can detect shot changes in a video and split them into separate clips. It provides a wide range of features such as splitting videos into scenes, detection and removal of commercials from PVR-saved video sources, video analytics, etc. It is packaged as a python library as well as a command-line tool. The `QIK` Video Processor used the `scenedetect` module

[7] `https://github.com/google/guava`
[8] `https://pyscenedetect.readthedocs.io/en/latest/`

of the command-line tool for identifying and extracting the key scenes from a video. For video retrieval, the Indexer and Query Processor used ClipCap [33] for generating keyframe captions. ClipCap comprises of a CLIP image encoder to construct an image embedding. These embeddings are mapped to a language model (GPT-2) embedding space using a mapping network to generate a caption. We use a pre-trained ClipCap model trained on the Conceptual Captions [97] dataset. The Conceptual Captions dataset consists of 3.3 million images and their descriptions extracted from the web. The dataset has a variety of images and is not limited by a set of categories, unlike the MS-COCO dataset making it more appropriate for captioning keyframes extracted from videos. For obtaining the longest common subsequence of frames required for ranking candidate videos, `QIK` Video Processor used Machine Learning PYthon (`mlpy`) library[9]. `mlpy` is a high-performance Python library primarily aimed at accelerating predictive modeling tasks. It is built on top of `numpy` and `scipy` and offers a wide range of machine learning methods for supervised and unsupervised learning.

The user interfaces for querying and navigating through the results were built using Python web development frameworks: Django[10] and Flask[11]. Python web development frameworks were considered due to the computationally intensive nature of the tasks involved. Compared to javascript based frameworks such as NodeJS[12], they offer the ability to leverage multiple threads required to speed up overall processing. A screenshot of `QIK` for image retrieval is shown in Figure 6.1. On the top right, we see the new images suggested, which are similar to the query image. By clicking on the suggested similar image queries, a user can intuitively explore the repository. The

[9] `http://mlpy.sourceforge.net/`
[10] `https://www.djangoproject.com/`
[11] `https://flask.palletsprojects.com/en/2.0.x/`
[12] `https://nodejs.org/en`

`QIK` software is available publicly on GitHub[13].

[13] `https://github.com/MU-Data-Science/QIK`

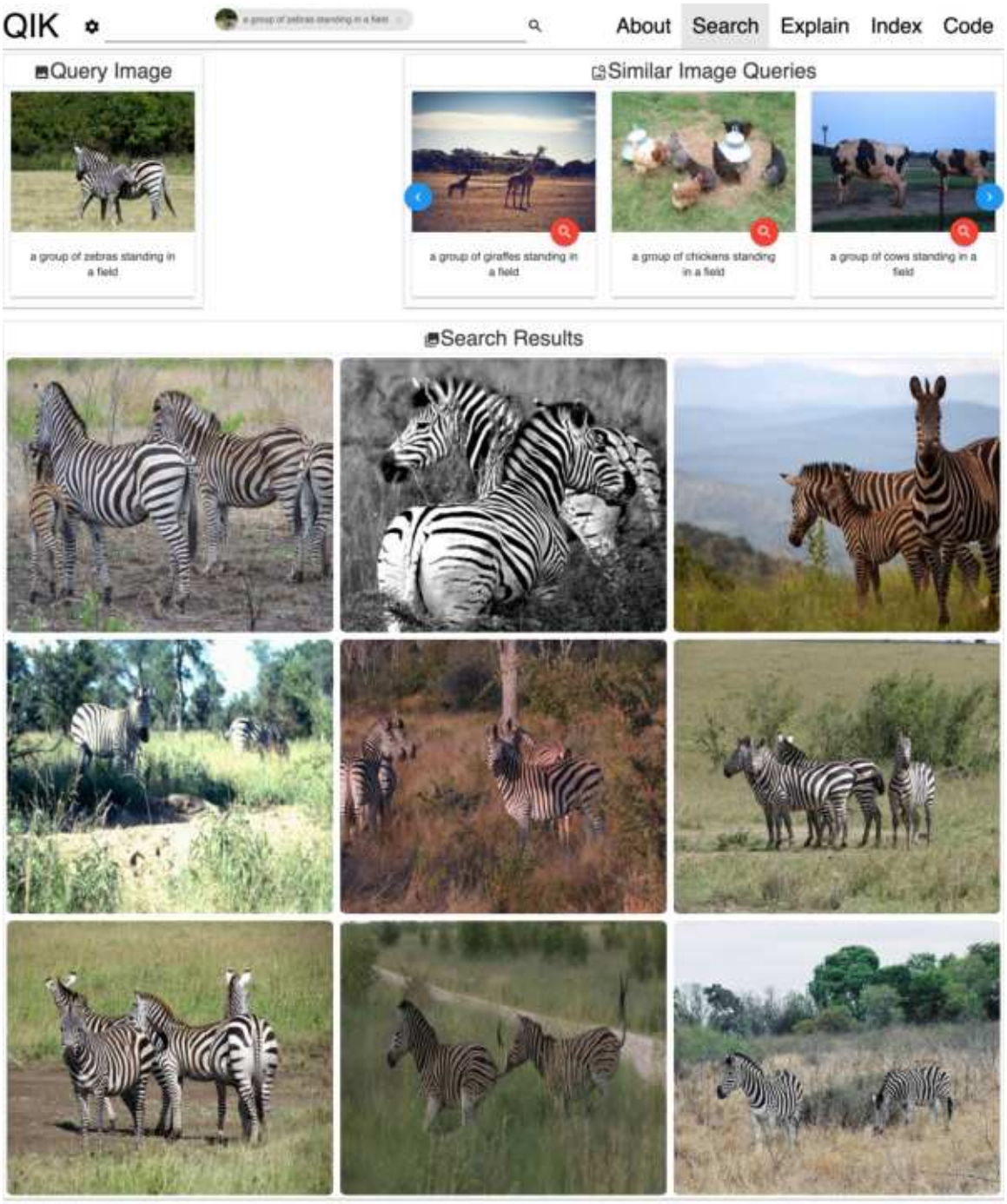


Figure 6.1: `QIK` user interface

Chapter 7

Evaluation

In this chapter, we report the performance evaluation of `QIK` for image and video retrieval tasks. We compare `QIK` against various state-of-the-art image and video retrieval techniques. We start by describing the datasets used for evaluating image and video retrieval performance in section 7.1, followed by the techniques compared in section 7.2. Section 7.3 describes the experimental setup and section 7.4 describes the evaluation metric used. Finally, we present the evaluation results in section 7.6.

7.1 Datasets

We used four benchmark image and video datasets to evaluate the retrieval performance of `QIK`. Table 7.1 summarizes the datasets used, which are, in brief, described in the following subsections.

Dataset	Data Type	Total Instances	Dataset Size
MS COCO	Image	15K	2.4GB
MSR-VTT	Video	10K	6.3GB

Table 7.1: Dataset summary

7.1.1 MS COCO

MS COCO [9][1] is a large-scale dataset comprising images of complex everyday scenes involving 80 common objects in their natural context. The main goal of this dataset was to advance the state-of-the-art in scene understanding which involves image captioning, object detection, segmentation, and key-point detection. The dataset was first released in 2014 and now has 164K images split into training (118K), validation (5K), and test (41K) sets. The annotations for each image in the dataset were crowdsourced through Amazon Mechanical Turk (MTurk)[2]. For object detection, the dataset annotation comprises the bounding box coordinates along with the per-instance segmentation masks for all the 80 object categories. For image captioning, each image has five human-annotated captions. We used the MS COCO data for evaluating the image retrieval performance of `QIK`. We used a random subset of 15K images for evaluation as some of the competitors of `QIK` could not operate on larger number of images.

7.1.2 MSR-VTT

The Microsoft Research Video to Text dataset, abbreviated as MSR-VTT [12], is a large-scale video dataset curated for advancing the field of video understanding.

[1] `https://cocodataset.org`
[2] `https://www.mturk.com`

It comprises 10,000 web video clips totaling 41.2 hours. The clips were obtained, by executing 257 queries, corresponding to 20 categories, on a commercial video search engine. The top 150 videos for the queries were then cleaned up to remove short, duplicate, and poor-quality videos. Each clip is accompanied by 20 captions, annotated by MTurk workers, that describe its content. The clips are split into 7,010 training videos and 2,990 testing videos. The uniqueness of the MSR-VTT dataset lies in is its diversity of visual content that spans a wide variety of categories.

7.2 Competitors

In this section, we briefly list out competitors used for evaluation the image and video retrieval performace of `QIK`

7.2.1 Image Retrieval

- **DIR:** DIR [6] produces a global and compact fixed-length representation of each image by aggregating many region-wise descriptors so that it is robust to scale and transformation. The regions to be pooled are predicted using a region proposal network.

- **DELF:** DELF [5] extracts dense features from an image by passing it through a fully convolutional network, which are then passed through an attention layer that measures the relevance of the local features descriptors.

- **CroW:** CroW [7] is an efficient non-parametric weighting and aggregation scheme to transform convolutional image features to a compact global image

feature. It creates image representations by cross-dimensional weighting and aggregation of deep convolutional neural network layer outputs. To improve upon the efficiency of the system, spatial and channel wise weighting is done by aggregating the weights derived from the spatial activation of the output layer and the sparsity weights based on of the feature maps.

- **FR-CNN:** FR-CNN [8] uses image-wise and region-wise representations pooled from an object detection CNN such as Faster R-CNN. While the image-wise representations serve as global features used during the filtering step, the region-wise representations serve as local features used for re-ranking.

- **CSQ:** CSQ [10], introduces a global similarity metric called central similarity that efficiently maps image features to hash codes and constructs hash centers. Similar pairs would have a common center and dissimilar pairs would converge to different centers hence improving learning efficiency and retrieval accuracy.

- **LIRE:** LIRE [63] is an open-source content based image retrieval system system that provides a wide range of options such as color histograms, color and edge directivity descriptor, etc., to extract local features and global feature vectors.

7.2.2 Video Retrieval

- **DnS:** DnS [11] is a Knowledge Distillation framework consisting of a coarse-grained student network that enables fast retrieval but has low performance, a fine-grained student network that has high performance but is computationally expensive, and a selection network that decides upon the query-target feature pairs need to be re-ranked by the fine-grained student networks.

- **CSQ:** In case of video retrieval, to retain temporal information videos are hashed using 3D CNNs [89]. The training data videos and labels are then associated with hash centers generated to further obtain the semantic hash centers. Videos having the same center as the query video are considered to be a match.

7.3 Experimental Setup

We ran the experiments on CloudLab [98][3], a testbed for cloud computing research and new applications. We used the bare-metal machines in the Utah data center. All nodes had a Ten-core Intel E5-2640v4 CPU at 2.4 GHz, with 64 GB of RAM. The operating system installed across all of the was Ubuntu 18.04.

7.4 Evaluation Metric

We use the Mean Average Precision (mAP) [99] to evaluate image and video retrieval performance. mAP is considered the standard measure for comparing search algorithms and is obtained by averaging the Average Precision (AP). AP combines precision and recall over a set of ranked results.

Precision signifies the ability to retrieve the most relevant items. It is defined as the fraction of relevant items among the retrieved items and is represented as

$$Precision = \frac{Total\ number\ of\ items\ retrieved\ that\ are\ relevant}{Total\ number\ of\ items\ retrieved}$$

[3] `https://www.cloudlab.us/`

Recall signifies the ability to find all the relevant items in the database. It is defined as the fraction of relevant items that were retrieved and is represented as

$$Recall = \frac{Total\ number\ of\ items\ retrieved\ that\ are\ relevant}{Total\ number\ of\ relevant\ items\ in\ the\ database}$$

Items that are correctly labeled as belonging to the positive class are considered True Positives(TPs). Items that are incorrectly labeled as belonging to the positive class are considered False Positives(FPs). When an item is not labeled as belonging to the positive class when it should have been, it is considered a False Negative(FN). Items that are correctly labeled as not belonging to the positive class are considered True Negatives(TN). Precision and Recall can be defined in terms of TPs, FPs, FNs, and TNs as

$$Precision = \frac{TP}{TP + FP}$$

$$Precision = \frac{TP}{TP + FN}$$

For web-scale information retrieval, we use $P@k$. $P@k$ denotes the number of relevant items among the top-k retrieved documents. $AP@k$ measures the average relevance score of a set of top-k documents obtained as the result of a query computed as

$$AP = \frac{\sum_{k=1}^{R} P@k}{R}$$

Hence $mAP@k$ (for the total number of queries Q) is computed as

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

7.5 Query and Ground Truth Formulation

7.5.1 Image Retrieval

To evaluate the image retrieval performance of `QIK`, we chose a random subset of 15K images from the MS COCO [9] dataset. We generated two-, three-, and four-object combinations using the 80 objects specified in MS COCO such as "person + couch", "person + car + cup", etc. The number of two-object, three-object, and four-object combinations were 50, 50, and 40, respectively. Consider only two-object combinations. For each combination c, we did the following: We selected the images containing those two objects based on human labeling from the 15K dataset. Let I denote the selected images. For each image $i \in I$, we identified the true matches for i as a query image using a pre-trained Universal Sentence Encoder [100] model. Essentially, we computed the similarity between the human-annotated captions of i against the human-annotated captions of other images in I and used a similarity threshold τ to determine a true match. That is, if any caption of i was similar to a caption of an image j in I with similarity greater than τ, then j was considered a true match for i. Thus, we completely relied on human judgment by using their annotations for deciding the true matches for an image query. We computed the mAP value for the combination c for different top-k matches (k=2, k=4, k=8, and k=16). We followed the same procedure for three-object and four-object combinations. The total number of image queries in two-object, three-object, and four-object combinations were 15478, 4400, and 1745, respectively.

7.5.2 Video Retrieval

To evaluate the video retrieval performance of QIK, we use the MSR-VTT [12] dataset. The dataset, comprised of the manually labeled videos that was split into 7,010 training and 2,990 testing videos. All the training videos constituted the database of videos, while the testing videos constituted the query videos. All videos that belonged to a particular class were considered as the ground truth.

7.6 Evaluation Results

7.6.1 QIK: Image Retrieval using Captions vs. Image Retrieval using Detected Objects

We first compared the image retrieval performance of QIK when using captions versus detected objects in PIUs. Hereinafter, we denote QIK when using captions as $\texttt{QIK}_c$ and QIK when using detected objects as $\texttt{QIK}_o$. Next, we compared $\texttt{QIK}_c$ with its competitors, which used CNN-based features for filtering. Finally, we compared the video retrieval performance of QIK, hereinafter, denoted as $\texttt{QIK}_v$ with its competitors. Note that $\texttt{QIK}_c$ used Algorithm 2 and $\texttt{QIK}_o$ used Algorithm 5.

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
$\texttt{QIK}_c$	**0.93**	**0.92**	**0.92**	**0.91**	**0.82**	**0.82**	**0.82**	**0.81**
$\texttt{QIK}_o^{0.9}$	0.70	0.70	0.69	0.69	0.58	0.59	0.58	0.57
$\texttt{QIK}_o^{0.8}$	0.75	0.77	0.77	0.76	0.56	0.60	0.59	0.58

Table 7.2: $\texttt{QIK}_c$ vs $\texttt{QIK}_o$: two-object combinations (avg. mAP)

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
$\texttt{QIK}_c$	**0.92**	**0.91**	**0.91**	**0.91**	**0.82**	**0.82**	**0.82**	**0.81**
$\texttt{QIK}_o^{0.9}$	0.67	0.67	0.67	0.66	0.56	0.56	0.55	0.54
$\texttt{QIK}_o^{0.8}$	0.70	0.72	0.74	0.73	0.48	0.55	0.56	0.55

Table 7.3: $\texttt{QIK}_c$ vs $\texttt{QIK}_o$: three-object combinations (avg. mAP)

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
$\texttt{QIK}_c$	**0.92**	**0.90**	**0.90**	**0.89**	**0.77**	**0.78**	**0.79**	**0.78**
$\texttt{QIK}_o^{0.9}$	0.55	0.56	0.56	0.56	0.46	0.47	0.46	0.45
$\texttt{QIK}_o^{0.8}$	0.73	0.75	0.76	0.75	0.56	0.60	0.59	0.58

Table 7.4: $\texttt{QIK}_c$ vs $\texttt{QIK}_o$: four-object combinations (avg. mAP)

Our goal was to show that captions provide superior performance as they can capture the relationships between important objects in an image compared to just retrieving images containing certain objects. Table 7.2, Table 7.3, and Table 7.4 show the average of the mAP values for the two-object, three-object, and four-object combinations, respectively. Clearly, $\texttt{QIK}_c$ outperformed $\texttt{QIK}_o$ for two different probability thresholds for object detection, *i.e.*, 0.9 and 0.8. Thus, one can conclude that captions in PIUs indeed capture the object relationships leading to superior image retrieval performance for everyday scenes.

7.6.2 $\texttt{QIK}_c$ vs. Its Competitors

Next, we compared $\texttt{QIK}_c$ with its competitors, which used CNN-based features for filtering. For fair evaluation, we used the default parameters in the code of DIR [6], DELF [5], CroW [7], FR-CNN [8] and CSQ [10]. For LIRE [63], we used CEDD [67] to

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
QIK$_c$	**0.93**	**0.92**	**0.92**	**0.91**	**0.82**	**0.82**	**0.82**	**0.81**
DIR	0.75	0.77	0.76	0.75	0.61	0.63	0.61	0.59
DELF	0.54	0.58	0.56	0.54	0.37	0.40	0.40	0.37
CroW	0.82	0.85	0.84	0.83	0.69	0.71	0.69	0.67
FR-CNN	0.78	0.80	0.79	0.77	0.65	0.66	0.63	0.59
CSQ	0.81	0.82	0.82	0.81	0.66	0.67	0.68	0.65
LIRE	0.45	0.51	0.50	0.46	0.24	0.29	0.29	0.27

Table 7.5: Results for two-object combinations (avg. of mAP)

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
QIK$_c$	**0.92**	**0.91**	**0.91**	**0.91**	**0.82**	**0.82**	**0.82**	**0.81**
DIR	0.72	0.73	0.72	0.72	0.57	0.59	0.57	0.55
DELF	0.45	0.50	0.49	0.48	0.31	0.34	0.33	0.32
CroW	0.85	0.85	0.83	0.81	0.66	0.67	0.64	0.61
FR-CNN	0.77	0.81	0.79	0.78	0.65	0.65	0.62	0.58
CSQ	0.83	0.82	0.82	0.80	0.60	0.60	0.62	0.61
LIRE	0.44	0.50	0.49	0.45	0.22	0.27	0.27	0.25

Table 7.6: Results for three-object combinations (avg. of mAP)

extract the features of images and indexed them using Lucene. In addition, the sentence parse trees were used during the ranking step for tree edit distance computation. For the two-object combinations, we computed the mAP value for each combination and report the average of the mAP values. Similarly, we report the average of mAP values for the three-object and four-object combinations. Tables 7.5, 7.6, and 7.7 report these values for two different τ values and different values of k.

In all cases, QIK outperformed its competitors by virtue of using captions in PIUs and applying NLP processing. QIK$_c$ was able to capture the the relationships between objects in everyday scenes leading to superior performance. Other approaches relied

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
QIK$_c$	**0.92**	**0.90**	**0.90**	**0.89**	**0.77**	**0.78**	**0.79**	**0.78**
DIR	0.77	0.78	0.76	0.74	0.62	0.64	0.62	0.59
DELF	0.48	0.53	0.52	0.50	0.32	0.34	0.34	0.32
CroW	0.83	0.85	0.84	0.82	0.68	0.68	0.67	0.64
FR-CNN	0.75	0.78	0.78	0.77	0.65	0.65	0.63	0.61
CSQ	0.77	0.78	0.79	0.79	0.62	0.63	0.63	0.62
LIRE	0.45	0.50	0.48	0.45	0.22	0.26	0.26	0.25

Table 7.7: Results for four-object combinations (avg. of mAP)

on local/global descriptors of images constructed from features with or without CNNs. In most cases, CroW was the best approach among the chosen competitors; LIRE was the worst approach. This confirms that techniques using CNN-based features are superior to traditional feature-based indexing of images.

	Two-object combination	Three-object combination	Four-object combination
QIK$_c$	0.74 s	0.82 s	0.91 s
CroW	3.33 s	3.37 s	3.36 s
FR-CNN	21.12 s	21.03 s	21.06 s
DIR	0.79 s	0.79 s	0.79 s
DELF	34.05 s	35.07 s	34.24 s
LIRE	0.26 s	0.26 s	0.26 s
CSQ	2.07 s	2.07 s	2.07 s

Table 7.8: Average time taken (in seconds) for image retrieval

We measured the average time taken by each technique for image retrieval. As reported in Table 7.8, QIK was competitive in terms of average retrieval time. LIRE had the fastest execution time but yielded the lowest mAP value. CroW's mAP value was the second-best, followed by CSQ. Both the techniques had competitive execution

times. In terms of execution times, DELF was the slowest. The time taken for each step involved in the retrieval process has been listed in Table 7.9

	Time Taken
Image Captioning	0.46 s
Candidate Retrieval	0.14 s
Candidate Ranking	0.21 s

Table 7.9: Break up of the average time taken (in seconds) for image retrieval

7.6.3 QIK: Image Retrieval using Show and Tell vs. Image Retrieval using ClipCap

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
$\texttt{QIK}_{Show\ and\ Tell}$	0.95	0.96	0.96	0.96	0.85	0.85	0.86	0.86
$\texttt{QIK}_{ClipCap}$	0.93	0.92	0.92	0.91	0.82	0.82	0.82	0.81

Table 7.10: $\texttt{QIK}_c$ vs $\texttt{QIK}_o$: two-object combinations (avg. mAP)

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
$\texttt{QIK}_{Show\ and\ Tell}$	0.92	0.93	0.95	0.95	0.80	0.79	0.82	0.82
$\texttt{QIK}_{ClipCap}$	0.92	0.91	0.91	0.91	0.82	0.82	0.82	0.81

Table 7.11: $\texttt{QIK}_c$ vs $\texttt{QIK}_o$: two-object combinations (avg. mAP)

To demonstrate the impact of the captioning technique used, we compared the retrieval accuracy of QIK using two different captioning models namely Show and Tell [31] and ClipCap [33]. Show and Tell was trained on the MS COCO dataset [9],

	τ=0.6				τ=0.7			
	k=2	k=4	k=8	k=16	k=2	k=4	k=8	k=16
$\texttt{QIK}_{Show\ and\ Tell}$	0.96	0.96	0.97	0.97	0.86	0.86	0.86	0.87
$\texttt{QIK}_{ClipCap}$	0.92	0.90	0.90	0.89	0.77	0.78	0.79	0.78

Table 7.12: $\texttt{QIK}_c$ vs $\texttt{QIK}_o$: two-object combinations (avg. mAP)

while ClipCap was trained on the Conceptual Captions dataset. Although the Conceptual Captions dataset [95] is a more challenging dataset, covering a vast set of objects, there is a large diversity in its captions since they were harvested from the web. Table 7.10, Table 2 7.11, and Table 7.12 show the average of the mAP values for the two-object, three-object, and four-object combinations, respectively. We can see that the QIK attains comparable accuracy irrespective of the captioning model used. In all cases, QIK outperformed its competitors by using NLP on the captions in PIUs to construct tree structures and capturing the relationships between objects, leading to superior image retrieval performance for everyday scenes.

7.6.4 Scalability of QIK

To test the scalability of QIK, we indexed 124K images in MS COCO. We executed all 124K image queries; on an average, QIK took 2.63 seconds for each query. This shows that QIK can support efficient retrieval on large image repositories. The time taken for each step involved in the retrieval process over the 124K image dataset has been listed in Table 7.13

	Time Taken
Image Captioning	0.45 s
Candidate Retrieval	1.32 s
Candidate Ranking	2.63 s

Table 7.13: Break up of the average time taken (in seconds) for image retrieval

	k=2	k=4	k=8	k=16
QIK$_s$	0.4166	0.4372	0.4338	0.4232
QIK$_l$	0.4283	0.4485	0.4473	0.4341
QIK$_t$	0.4306	0.4500	0.4477	0.4343
QIK$_v$	**0.4349**	**0.4569**	**0.4527**	**0.4376**

Table 7.14: QIK$_v$ Results for ranking schemes (avg. of mAP)

7.6.5 QIK: Video Retrieval Ranking Approaches

We first compared the video retrieval performance of QIK using the various ranking procedures described in Section 5.6. Our goal was to show that ranking candidate videos based on the order of events and query frame caption would provide superior performance as they can capture the relationships between frames and between important objects in a frame. Hereinafter, we denote the procedure using Algorithm 8 as QIK$_s$, Algorithm 9 as QIK$_l$, Algorithm 10 as QIK$_t$, and Algorithm 11 as QIK$_v$. Table 7.14 show the average of the mAP values for the various ranking procedures. QIK$_l$ outperforms QIK$_s$ proving the significance of frame ordering. QIK$_t$ outperformed QIK$_l$ and QIK$_s$ concluding that considering ordering of events and captions in PIUs to capture object relationships in a frame lead to superior video retrieval. Clearly, QIK$_v$ outperformed all the three approaches. The ranking time taken for each technique is listed in Table 7.15. QIK$_s$ and QIK$_l$ were the fastest, while QIK$_{td}$ and QIK$_v$ were competitive enough considering the overhead incurred in computing the tree

edit distance.

	Time Taken
$\texttt{QIK}_s$	0.03 s
$\texttt{QIK}_l$	0.04 s
$\texttt{QIK}_t$	1.59 s
$\texttt{QIK}_v$	1.58 s

Table 7.15: Average ranking time taken (in seconds)

7.6.6 $\texttt{QIK}_v$ vs. Its Competitors

Next, we compared QIK, using Algorithm 11 denoted as $\texttt{QIK}_v$ with its competitors that uses CNN-based features namely CSQ and DnS. For evaluating our competitor, we used the publicly available implementations along with the default parameters defined by the authors. CSQ used a 64-bit MFNet [101] CNN to extract video features. The model and separately trained on the Hollywood2 and MSR-VTT datasets to identify, hash, and cluster the videos.

While DnS proposed a selection network that decides upon which query-target feature pairs need to be re-ranked, their implementation lacked it. So, we use a defined percentage of pairs to be re-ranked. We evaluated DnS by re-ranking 100% 50% and 0% of the candidates by the fine grained student network. 100% of the candidates were re-ranked as theoretically, it would have the best accuracy. Re-ranking 0% of the candidates would mean the fastest retrieval time. We also re-rank 50% of the candidates as it would have the right balance of accuracy and computational efficiency. Table 7.16 compare the video retrieval performance of $\texttt{QIK}_v$ with its competitor. QIK outperformed CSQ and DnS, proving its significance in retrieving realistic and chal-

lenging scenes and videos.

	k=2	k=4	k=8	k=16
$\texttt{QIK}_v$	**0.4349**	**0.4569**	**0.4527**	**0.4376**
DnS (%=1)	0.3692	0.3966	0.3916	0.3652
DnS (%=0.5)	0.0913	0.1216	0.1436	0.1507
DnS (%=0)	0.1288	0.1612	0.1773	0.1700
CSQ	0.4028	0.4160	0.4116	0.4038

Table 7.16: Video retrieval results (avg. of mAP)

The retrieval times for each approach is compared in Table 7.17. QIK was competitive in terms of retrieval times, while CSQ was the fastest. DnS had the worst execution time. While increasing the number of candidates re-ranking helped improve the mAP for DnS, the time taken for execution incurs an overhead in the process.

	Time Taken
$\texttt{QIK}_v$	11.38 s
DnS (%=1)	123.22 s
DnS (%=0.5)	116.11 s
DnS (%=0)	110.49 s
CSQ	1.00 s

Table 7.17: Average time taken (in seconds) for video retrieval

The time taken for each step involved in the retrieval process has been listed in Table 7.18

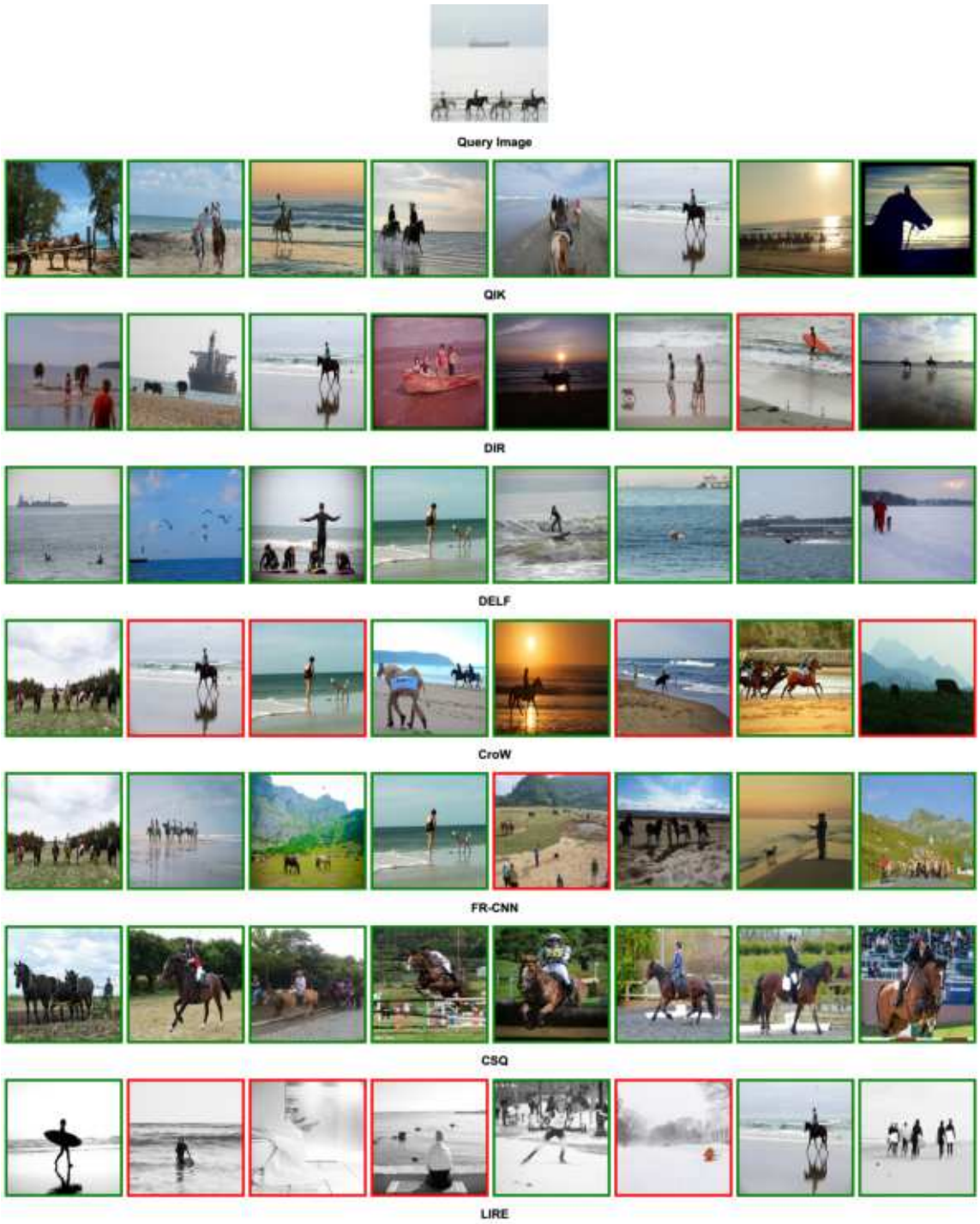


Figure 7.1: `QIK` image retrieval results compared with its competitors - 1 ($\tau = 0.7$)

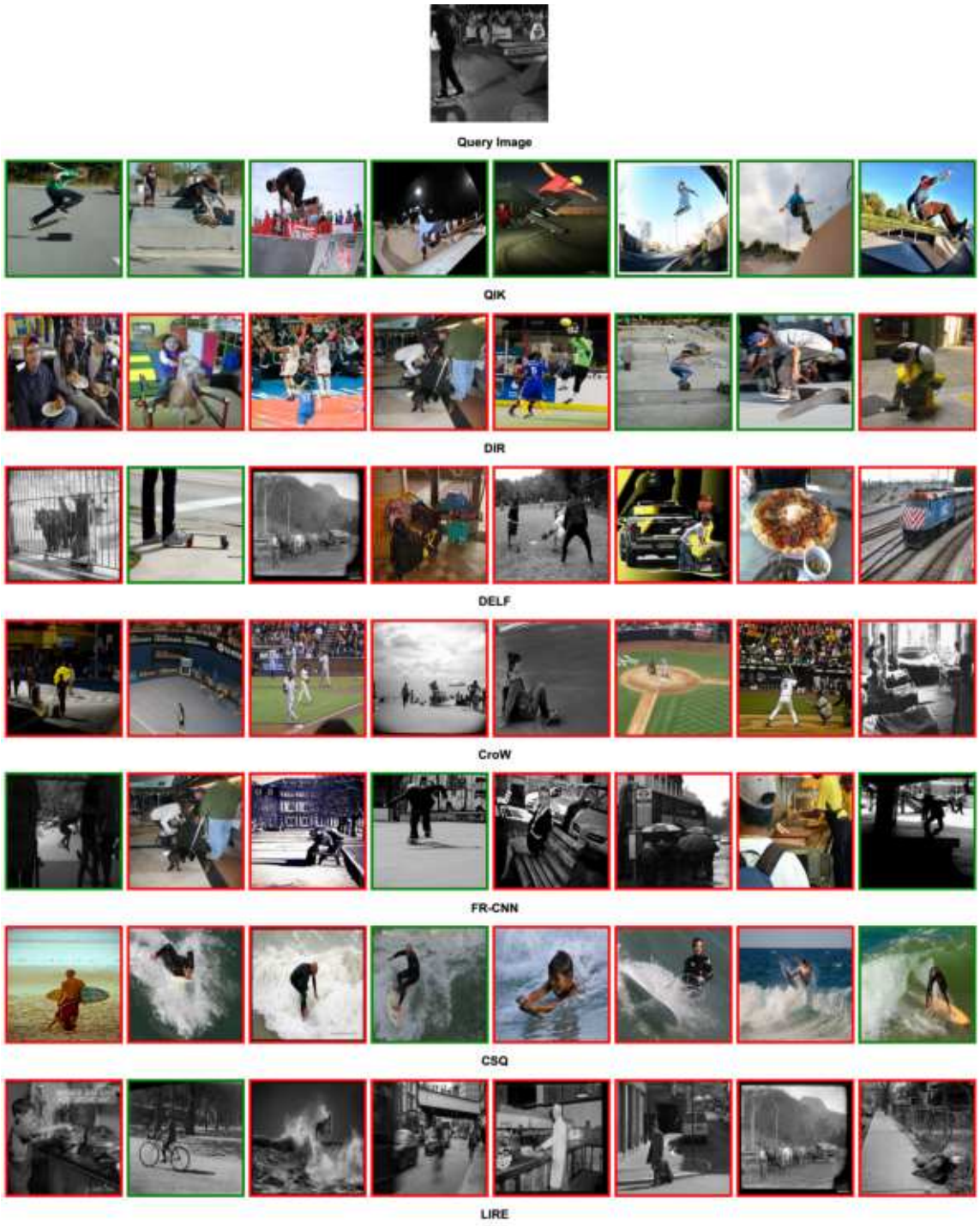


Figure 7.2: `QIK` image retrieval results compared with its competitors - 2 ($\tau = 0.7$)

	Time Taken
Scene Detection	0.62 s
Scene Captioning	5.44 s
Candidate Retrieval	3.63 s
Candidate Ranking	1.58 s

Table 7.18: Break up of the average time taken (in seconds) for video retrieval

7.7 Examples

7.7.1 Image Retrieval

Figures 7.1 and 7.2 show two queries and the output of the different techniques compared for $k = 8$. As seen in Figure 7.1, QIK returned only images of people with one or more horse in them, while other techniques returned at least one false positive for the query. While there were images containing giraffes, there overlook having people in them. Similarly, Figure 7.2 shows that QIK returned only images of skateboarders, while all other techniques returned at least one false positive.

7.7.2 Video Retrieval

Figures 7.3 and 7.4 show two video queries posed to QIK and its output compared with the competitors namely CSQ and DnS for $k = 8$. It is seen in Figure 7.3 the query video is a clip from a wrestling game. QIK fetched all videos that had wrestling in them. Even though CSQ fetched videos that belonged to various sports being played, none of them were related to wrestling. While DnS fetched wrestling videos, it had mulitple false positives. Similary, in Figure 7.4 `QIK` returned only video related to a fashion shows. However, other techniques returned at least one false positive.

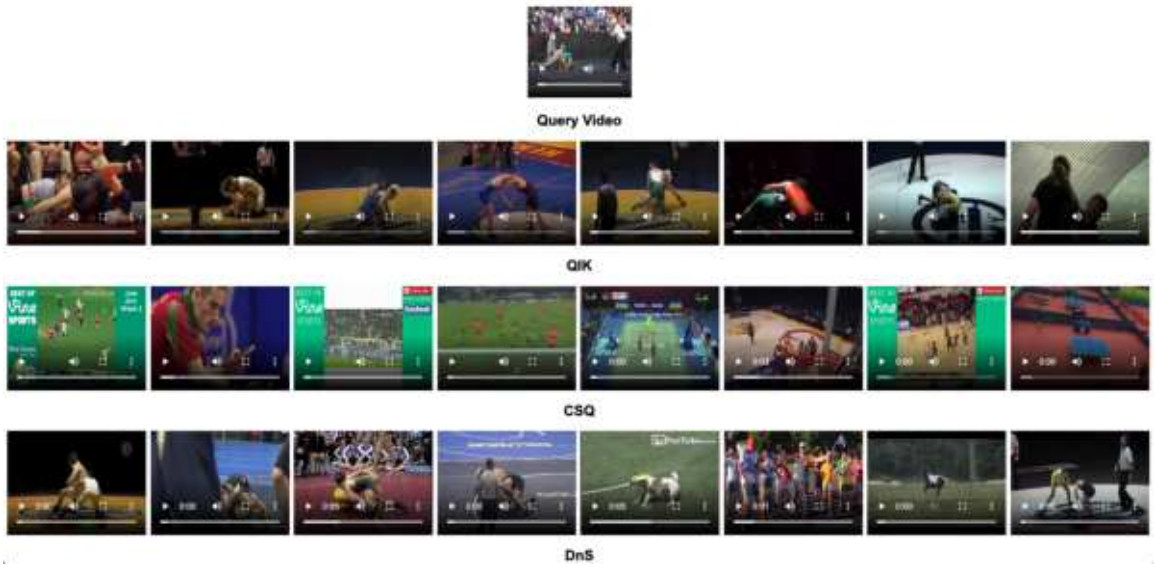


Figure 7.3: QIK video retrieval results compared with CSQ - 1

Figure 7.4: QIK video retrieval results compared with CSQ - 2

Chapter 8

Conclusion and Future Work

Through this dissertation, we introduce `QIK`. `QIK` is an efficient system for large-scale image and video retrieval. It understands an image probabilistically by leveraging the predictions of deep neural networks designed for image understanding tasks to generate a PIU. Thorugh PIUs `QIK` accurately captures relationships between multiple objects present in complex scenes. The captions predicted for the images are analyzed linguistically by constructing sentence parse trees and dependency trees. During the filtering step, an optimized XPath query is constructed by transforming the parse tree of a query image caption into an XML document. The XPath query is then executed to fetch a set of candidate images from the database. During the ranking step, the candidate images are ranked based on the tree-edit distance of the tree structures constructed from their captions.

We described the implementation of `QIK` and conducted a thorough performance evaluation on the MS COCO dataset. We observed that captions provided superior performance compared to just retrieving images containing certain objects as captions

can capture the relationships between important objects in an image. Through our experiments, we observed that QIK achieved superior performance compared to state-of-the-art techniques for large-scale image retrieval such as LIRE, DIR, DELF, CSQ, FR-CNN, and CSQ. Through accurate metadata describing the context of an image QIK improved image retrieval capability than solely using image features. It is by virtue of its ability to capture relationships between objects in complex scenes enables QIK to outperform its competitors.

We also describe how QIK could be extended to enable efficient video retrieval. QIK performs linguistic analysis on the keyframes of a video to generate a PIU, thereby capturing the relationship amongst the frames in a video. The PIUs generated are then stored and indexed in a database. At the time of querying, optimized XPath queries are constructed on the parse tree representations of the captions generated for frames. During the filtering step, the XPath queries for each frame are aggregated using a Boolean OR condition and queried to obtain a set of frames corresponding to a set of candidate videos. During the ranking step, a score is assigned to each candidate video based on the number of frames matching the keyframes of the query video. The candidate videos are further ranked in decreasing order of the scores.

Through comprehensive performance evaluation on the MSR-VTT datasets, we show the potential of QIK for efficient video retrieval compared to state-of-the-art techniques for video retrieval such as CSQ and DnS. It is by virtue of its ability to capture relationships between objects in complex scenes and the relationship among scenes in a given video, that QIK was able to outperform its competitors.

In the future, we plan to explore further on ways that could be used to optimize retrieval performance for video retrieval. This could include investigating the po-

tential of using XQuery for similar scene retrieval. Image captioning was the most time-consuming process among all the sub-processes (due to the model complexity involved). This is often aggravated by the number of key frames extracted that need to be captioned. Optimizing the key frames extracted to filter out the ones that do not contribute to retrieval could help lower retrieval time. Additionally, we also plan to span `QIK` across multiple domains, especially in the area of digital pathology, where the gigabyte size of whole slide images poses a serious challenge.

Bibliography

[1] BV Patel and BB Meshram. Content Based Video Retrieval Systems. *arXiv preprint arXiv:1205.1641*, 2012.

[2] Aasif Ansari and Muzammil H Mohammed. Content Based Video Retrieval Systems-Methods, Techniques, Trends and Challenges. *International Journal of Computer Applications*, 112(7):13–22, 2015.

[3] Michael S Lew, Nicu Sebe, and John P Eakins. Challenges of Image and Video Retrieval. In *International Conference on Image and Video Retrieval*, pages 1–6, 2002.

[4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, USA, 2nd edition, 2009.

[5] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-Scale Image Retrieval with Attentive Deep Local Features. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1–10, 2017.

[6] Albert Gordo, Jon Almazán, Jerome Revaud, and Diane Larlus. Deep Image Retrieval: Learning Global Representations for Image Search. In *Proceedings of the European Conference on Computer Vision*, pages 241–257, 2016.

[7] Yannis Kalantidis, Clayton Mellina, and Simon Osindero. Cross-Dimensional Weighting for Aggregated Deep Convolutional Features. In *Computer Vision - ECCV 2016 Workshops*, pages 685–701, 2016.

[8] Amaia Salvador, Xavier Giro-i Nieto, Ferran Marques, and Shin'ichi Satoh. Faster R-CNN Features for Instance Search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016.

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proceedings of the European Conference on Computer Vision*, pages 740–755, 2014.

[10] Li Yuan, Tao Wang, Xiaopeng Zhang, Francis EH Tay, Zequn Jie, Wei Liu, and Jiashi Feng. Central Similarity Quantization for Efficient Image and Video Retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3083–3092, 2020.

[11] Giorgos Kordopatis-Zilos, Christos Tzelepis, Symeon Papadopoulos, Ioannis Kompatsiaris, and Ioannis Patras. DnS: Distill-and-Select for Efficient and Accurate Video Indexing and Retrieval. *arXiv preprint arXiv:2106.13266*, 2021.

[12] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. MSR-VTT: A Large Video Description Dataset for Bridging Video and Language. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[13] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 Second Edition W3C Recommendation. Technical Report REC-xml-20001006, World Wide Web Consortium, Oct 2000.

[14] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jerome Simeon. XML Path Language (XPath) 2.0 W3C Working Draft 16. Technical Report WD-xpath20-20020816, World Wide Web Consortium, Aug 2002.

[15] Kuo-Chung Tai. The Tree-to-Tree Correction Problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.

[16] Benjamin Paaßen, Claudio Gallicchio, Alessio Micheli, and Barbara Hammer. Tree Edit Distance Learning via Adaptive Symbol Embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3976–3985, 2018.

[17] Philip Bille. A Survey on Tree Edit Distance and Related Problems. *Theoretical Computer Science*, 337(1):217–239, 2005.

[18] Kaizhong Zhang and Dennis Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[19] Philip N Klein. Computing the Edit-Distance Between Unrooted Ordered Trees. In *European Symposium on Algorithms*, pages 91–102, 1998.

[20] Erik D Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An Optimal Decomposition Algorithm for Tree Edit Distance. *ACM Transactions on Algorithms (TALG)*, 6:1–19, 2009.

[21] Mateusz Pawlik and Nikolaus Augsten. Efficient Computation of the Tree Edit Distance. *ACM Transactions on Database Systems*, 40(1):3:1–3:40, Mar 2015.

[22] Mateusz Pawlik and Nikolaus Augsten. Tree Edit Distance: Robust and Memory-Efficient. *Information Systems*, 56:157–173, 2016.

[23] Stefan Schwarz, Mateusz Pawlik, and Nikolaus Augsten. A New Perspective on the Tree Edit Distance. In *Similarity Search and Applications*, pages 156–170, 2017.

[24] Burton H. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, page 1097–1105, 2012.

[26] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[30] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and Tell: A Neural Image Caption Generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2015.

[31] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and Tell: Lessons Learned from the 2015 MS COCO Image Captioning Challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):652–663, 2017.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[33] Ron Mokady, Amir Hertz, and Amit H Bermano. Clipcap: Clip Prefix for Image Captioning. *arXiv preprint arXiv:2111.09734*, 2021.

[34] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models from Natural Language Supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9, 2019.

[36] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[37] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded-Up Robust Features. *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[38] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual Categorization with Bags of Keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[39] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[40] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in Quantization: Improving Particular Object Retrieval in Large

Scale Image Databases. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[42] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural Codes for Image Retrieval. In *Proceedings of the European Conference on Computer Vision*, pages 584–599, 2014.

[43] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[44] Artem Babenko Yandex and Victor Lempitsky. Aggregating Local Deep Features for Image Retrieval. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[45] Filip Radenovic, Giorgos Tolias, and Ondrej Chum. Fine-Tuning CNN Image Retrieval with No Human Annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1655–1668, 2019.

[46] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating Local Descriptors into a Compact Image Representation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010.

[47] Relja Arandjelovic and Andrew Zisserman. All About VLAD. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2013.

[48] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. NetVLAD: CNN Architecture for Weakly Supervised Place Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[49] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 91–99, 2015.

[50] Marvin Teichmann, Andre Araujo, Menglong Zhu, and Jack Sim. Detect-To-Retrieve: Efficient Regional Aggregation for Image Search. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[51] Giorgos Tolias, Yannis Avrithis, and Herve Jegou. Image Search with Selective Match Kernels: Aggregation Across Single and Multiple Images. *International Journal of Computer Vision*, 116(3):247–261, 2016.

[52] Anastasiya Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiři Matas. Working Hard to Know Your Neighbor's Margins: Local Descriptor Learning Loss. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4829–4840, 2017.

[53] Dmytro Mishkin, Filip Radenović, and Jiři" Matas. Repeatability Is Not Enough: Learning Affine Regions via Discriminability. In *Proceedings of the European Conference on Computer Vision*, pages 287–304, 2018.

[54] Ondrej Chum, James Philbin, Josef Sivic, Michael Isard, and Andrew Zisserman. Total Recall: Automatic Query Expansion with a Generative Feature Model for Object Retrieval. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

[55] Ondřej Chum, Andrej Mikulík, Michal Perdoch, and Jiří Matas. Total Recall II: Query Expansion Revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 889–896, 2011.

[56] Giorgos Tolias, Ronan Sicre, and Herve Jegou. Particular Object Retrieval with Integral Max-Pooling of CNN Activations. In *International Conference on Learning Representations*, pages 1–12, 2016.

[57] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2048–2057, 2015.

[58] Albert Gordo and Diane Larlus. Beyond Instance-Level Image Retrieval: Leveraging Captions to Learn a Global Visual Representation for Semantic Retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[59] Quoc-Tuan Truong and Lauw Hady W. VistaNet: Visual Aspect Attention Network for Multimodal Sentiment Analysis. In *The Thirty-third AAAI Conference*, pages 305–312, 2019.

[60] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

[61] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded Compositional Semantics for Finding and Describing Images with Sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.

[62] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep Hashing Network for Efficient Similarity Retrieval. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[63] Mathias Lux and Savvas A. Chatzichristofis. LIRe: Lucene Image Retrieval: An Extensible Java CBIR Library. In *Proceedings of the 16th ACM International Conference on Multimedia*, pages 1085–1088, 2008.

[64] Jing Huang, S Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zabih. Image Indexing using Color Correlograms. In *Proceedings of IEEE computer society conference on Computer Vision and Pattern Recognition*, pages 762–768. IEEE, 1997.

[65] Shih-Fu Chang, Thomas Sikora, and Atul Purl. Overview of the MPEG-7 Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):688–695, 2001.

[66] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Textural Features Corresponding to Visual Perception. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(6):460–473, 1978.

[67] Savvas A Chatzichristofis and Yiannis S Boutalis. CEDD: Color and Edge Directivity Descriptor: A Compact Descriptor for Image Indexing and Retrieval. In *International Conference on Computer Vision Systems*, pages 312–322. Springer, 2008.

[68] Xiao Wu, Alexander G Hauptmann, and Chong-Wah Ngo. Practical Elimination of Near-Duplicates from Web Video Search. In *Proceedings of the 15th ACM International Conference on Multimedia*, pages 218–227, 2007.

[69] Zi Huang, Heng Tao Shen, Jie Shao, Bin Cui, and Xiaofang Zhou. Practical Online Near-Duplicate Subsequence Detection for Continuous Video Streams. *IEEE Transactions on Multimedia*, 12(5):386–398, 2010.

[70] Jérôme Revaud, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Event Retrieval in Large Video Collections with Circulant Temporal Encoding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2459–2466, 2013.

[71] Zhanning Gao, Gang Hua, Dongqing Zhang, Nebojsa Jojic, Le Wang, Jianru Xue, and Nanning Zheng. ER3: A Unified Framework for Event Retrieval, Recognition and Recounting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2253–2262, 2017.

[72] Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Ioannis Patras, and Yiannis Kompatsiaris. Near-Duplicate Video Retrieval by Aggregating Intermediate CNN Layers. In *International Conference on Multimedia Modeling*, pages 251–263, 2017.

[73] Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Ioannis Patras, and Yiannis Kompatsiaris. Near-Duplicate Video Retrieval with Deep Metric Learning. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 347–356, 2017.

[74] Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Ioannis Patras, and Ioannis Kompatsiaris. ViSiL: Fine-Grained Spatio-Temporal Video Similarity Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6351–6360, 2019.

[75] Tengda Han, Weidi Xie, and Andrew Zisserman. Video Representation Learning by Dense Predictive Coding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, Oct 2019.

[76] Haofei Kuang, Yi Zhu, Zhi Zhang, Xinyu Li, Joseph Tighe, Soren Schwertfeger, Cyrill Stachniss, and Mu Li. Video Contrastive Learning with Global Context. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3195–3204, 2021.

[77] Jingkuan Song, Yi Yang, Zi Huang, Heng Tao Shen, and Richang Hong. Multiple Feature Hashing for Real-Time Large Scale Near-Duplicate Video Retrieval. In *Proceedings of the 19th ACM International Conference on Multimedia*, pages 423–432, 2011.

[78] Liangliang Cao, Zhenguo Li, Yadong Mu, and Shih-Fu Chang. Submodular Video Hashing: A Unified Framework Towards Video Pooling and Indexing. In *Proceedings of the 20th ACM International Conference on Multimedia*, pages 299–308, 2012.

[79] Guangnan Ye, Dong Liu, Jun Wang, and Shih-Fu Chang. Large-Scale Video Hashing via Structure Learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2272–2279, 2013.

[80] Venice Erin Liong, Jiwen Lu, Yap-Peng Tan, and Jie Zhou. Deep Video Hashing. *IEEE Transactions on Multimedia*, 19(6):1209–1219, 2017.

[81] Yun Gu, Chao Ma, and Jie Yang. Supervised Recurrent Hashing for Large Scale Video Retrieval. In *Proceedings of the 24th ACM International Conference on Multimedia*, pages 272–276, 2016.

[82] Naifan Zhuang, Jun Ye, and Kien A Hua. DLSTM Approach to Video Modeling with Hashing for Large-Scale Video Retrieval. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3222–3227, 2016.

[83] Vivek Veeriah, Naifan Zhuang, and Guo-Jun Qi. Differential Recurrent Neural Networks for Action Recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4041–4049, 2015.

[84] Jie Qin, Li Liu, Mengyang Yu, Yunhong Wang, and Ling Shao. Fast Action Retrieval from Videos via Feature Disaggregation. *Computer Vision and Image Understanding*, 156:104–116, 2017.

[85] Hanwang Zhang, Meng Wang, Richang Hong, and Tat-Seng Chua. Play and Rewind: Optimizing Binary Representations of Videos by Self-Supervised Temporal Hashing. In *Proceedings of the 24th ACM International Conference on Multimedia*, pages 781–790, 2016.

[86] Shuyan Li, Zhixiang Chen, Jiwen Lu, Xiu Li, and Jie Zhou. Neighborhood Preserving Hashing for Scalable Video Retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8212–8221, 2019.

[87] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised Learning of Video Representations using LSTMs. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 843–852, 2015.

[88] Yang Feng, Lin Ma, Wei Liu, and Jiebo Luo. Spatio-temporal Video Re-localization by Warp LSTM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1288–1297, 2019.

[89] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. Multi-Fiber Networks for Video Recognition. In *Proceedings of the European Conference on Computer Vision*, 2018.

[90] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming Embedding and Weak Geometric Consistency for Large Scale Image Search. In *Proceedings of the 10th European Conference on Computer Vision: Part I*, pages 304–317, 2008.

[91] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google Landmarks Dataset v2 - A Large-Scale Benchmark for Instance-Level Recognition and Re-

trieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[92] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465, 2013.

[93] Danqi Chen and Christopher Manning. A Fast and Accurate Dependency Parser using Neural Networks. In *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

[94] Christian Grün, Sebastian Gath, Alexander Holupirek, and Marc H. Scholl. XQuery Full Text Implementation in BaseX. In *Proceedings of the 6th International XML Database Symposium on Database and XML Technologies*, pages 114–128, 2009.

[95] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2556–2565, 2018.

[96] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2018.

[97] Edwin G. Ng, Bo Pang, Piyush Sharma, and Radu Soricut. Understanding Guided Image Captioning Performance across Domains. *arXiv preprint arXiv:2012.02339*, 2020.

[98] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1–14, 2019.

[99] Steven M. Beitzel, Eric C. Jensen, and Ophir Frieder. *MAP*, pages 1691–1692. Springer US, Boston, MA, 2009.

[100] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. Universal Sentence Encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, 2018.

[101] Qishen Ha, Kohei Watanabe, Takumi Karasawa, Yoshitaka Ushiku, and Tatsuya Harada. MFNet: Towards Real-Time Semantic Segmentation for Autonomous Vehicles with Multi-Spectral Scenes. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5108–5115, 2017.

[102] Arun Zachariah, Praveen Rao, Anas Katib, Monica Senapati, and Kobus Barnard. A Gossip-Based System for Fast Approximate Score Computation

in Multinomial Bayesian Networks. In *Proceedings of the 35th IEEE International Conference on Data Engineering (ICDE)*, 2019.

[103] Mohamed Gharibi, Arun Zachariah, and Praveen Rao. FoodKG: A Tool to Enrich Knowledge Graphs Using Machine Learning Techniques. *Frontiers in Big Data*, 3, 2020.

[104] Suveen Angraal, Arun George Zachariah, Raaisa Raaisa, Rohan Khera, Praveen Rao, Harlan M. Krumholz, and John A. Spertus. Evaluation of Internet-Based Crowdsourced Fundraising to Cover Health Care Costs in the United States. *JAMA Network Open*, 4(1):e2033157–e2033157, 2021.

[105] Arun Zachariah and Maha Alrasheed. Private-Share: A Secure and Privacy-Preserving De-Centralized Framework for Large Scale Data Sharing. In *Proceedings of the 3rd ACM International Conference on Multimedia in Asia*, 2021.

[106] Arun Zachariah and Praveen Rao. Large-Scale Image and Video Retrieval on Everyday Scenes With Common Objects. *ACM Transactions on Intelligent Systems and Technology*, 2022. Under Review.

[107] Arun Zachariah, Mohamed Gharibi, and Praveen Rao. A Large-Scale Image Retrieval System for Everyday Scenes. In *Proceedings of the 2nd ACM International Conference on Multimedia in Asia*, 2021.

[108] Arun Zachariah, Mohamed Gharibi, and Praveen Rao. QIK: A System for Large-Scale Image Retrieval on Everyday Scenes With Common Objects. In *Proceedings of the 2020 International Conference on Multimedia Retrieval*, page 126–135, 2020.

[109] Arun Zachariah, Praveen Rao, Brian Corn, and Dominique Davison. Zero Shot Learning for Predicting Energy Usage of Buildings in Sustainable Design. *arXiv preprint arXiv:2202.05206*, 2022.

[110] Praveen Rao, Arun Zachariah, Deepthi Rao, Peter Tonellato, Wesley Warren, and Eduardo Simoes. Accelerating Variant Calling on Human Genomes Using a Commodity Cluster. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, page 3388–3392, 2021.

[111] Praveen Rao and Arun Zachariah. Enabling Large-Scale Human Genome Sequence Analysis on CloudLab. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–2, 2022.

[112] Daniel E. Lopez Barron, Praveen Rao, Deepthi Rao, Ossama Tawfik, and Arun Zachariah. Large-Scale Storage of Whole Slide Images and Fast Retrieval of Tiles using DRAM. In *Big Data II: Learning, Analytics, and Applications*, pages 45 – 50, 2020.

[113] Nouf Alrasheed, Arun Zachariah, Shivika Prasanna, Deepthi Rao, and Praveen Rao. Deepfakes for Histopathology Images: Myth or Reality? In *Proceedings of the IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 1 – 7, 2020.

VITA

Arun George Zachariah is a doctoral candidate in the Department of Electrical Engineering and Computer Science at the University of Missouri-Columbia, under the guidance of Dr. Praveen Rao. His research interest focuses on Knowledge Management, intersecting scalable systems design, data science, and machine learning. He is a part of the Scalable Data Science (SDS) Lab, where he works on designing scalable algorithms and systems for data and knowledge management [102, 103, 104, 105] and information retrieval [106, 107, 108]. As a part of his Ph.D., he is involved in many interdisciplinary research projects requiring the use of AI and cluster computing technologies to enable gathering insights and solving challenging problems in the energy and healthcare domains, such as sustainable housing design [109], genomic analysis [110, 111], cancer cell detection in histopathology images [112, 113], etc.

In 2021, Arun received the Upsilon Pi Epsilon Scholarship Award and the MU Graduate Professional Council Research Development Award. He was also awarded multiple travel grants such as the ACM SIGMM Travel Grant (2021), UMKC School of Graduate Studies Travel Grant (2019), Student Activity Fee Committee Travel Grant (2019), and CSEE Balaji Krithikaivasan Memorial Travel Grant (2019). During his Ph.D., he was involved in organizing numerous workshops and after-school programs.

In the summer of 2021, Arun was an Intern at Arm Inc, automating and optimizing complex 5G system workloads. He started his Ph.D. in the Fall of 2018 at the University of Missouri-Kansas City and transferred to the University of Missouri-Columbia in the Spring of 2020. Prior to starting his Ph.D., he worked as a Technology Analyst

at Infosys Technologies, working with Distributed Databases and Systems.

After completing his Ph.D., Arun shall join Nvidia as a Senior Software Systems Engineer to scale AI-enabled video analytics applications.