VISE: Vehicle Image Search Engine with Traffic Camera

Hyewon Choi
University of Toronto
jennachoi@cs.toronto.edu

Erkang Zhu
University of Toronto
ekzhu@cs.toronto.edu

Arsala Bangash
University of Toronto
bangashm@cs.toronto.edu

Renée J. Miller
Northeastern University
miller@northeastern.edu

ABSTRACT

We present VISE, or Vehicle Image Search Engine, to support the fast search of similar vehicles from low-resolution traffic camera images. VISE can be used to trace and locate vehicles for applications such as police investigations when high-resolution footage is not available. Our system consists of three components: an interactive user-interface for querying and browsing identified vehicles; a scalable search engine for fast similarity search on millions of visual objects; and an image processing pipeline that extracts feature vectors of objects from video frames. We use transfer learning technique to integrate state-of-the-art Convolutional Neural Networks with two different refinement methods to achieve high retrieval accuracy. We also use an efficient high-dimensional nearest neighbor search index to enable fast retrieval speed. In the demo, our system will offer users an interactive experience exploring a large database of traffic camera images that is growing in real time at 200K frames per day.

PVLDB Reference Format:
Hyewon Choi, Erkang Zhu, Arsala Bangash, and Renée J. Miller. VISE: Vehicle Image Search Engine with Traffic Camera Images. *PVLDB*, 12(12): 1842-1845, 2019.
DOI: https://doi.org/10.14778/3352063.3352080

1. INTRODUCTION

After a hit-and-run incident, police officers typically review available camera footage to trace the suspected offender's vehicle. Since camera footage may be of insufficient resolution for identifying license plates, this process can be laboriously time-consuming and may lead to missing the best opportunity to stop the vehicle before it becomes untraceable or abandoned. VISE, Vehicle Image Search Engine, can be used to narrow down the locations of the offender's vehicle in seconds. In another use case, VISE can also be used to investigate a vehicle's past activities by searching through massive historical video frames.

We developed VISE to support the fast search of highly-similar vehicles given an existing image of a vehicle in question. Through a simple and intuitive user interface, our demonstration will let users select an image of a vehicle, and obtain a ranked list of similar vehicle images located by traffic cameras from a large search index containing more than approximately 600,000 identified vehicles per day. The user can refine the search result based on time, location of cameras, and the travel distance from the initial location of the vehicle in question.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352080

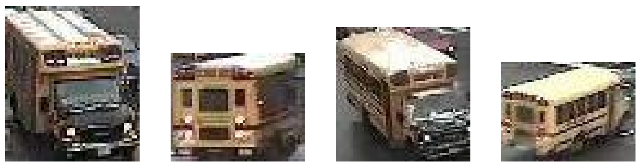


Figure 1: Objects from different viewpoints

In Section 2, we discuss our innovations in combining Convolutional Neural Networks (CNNs) with our custom refining methods to improve retrieval accuracy. We also discuss a two-stage retrieval process that enables both interactive speed and high accuracy. In Section 3, we present the architecture of the search engine system and discuss its capability. Finally, we present the user interface features in a walk-through of a usage scenario in Section 4.

2. TECHNICAL OVERVIEW

In this section, we describe the details of VISE and our innovations in searching visual objects (i.e., vehicles). The visual object search problem can be defined as follow:

Definition 1. *Given an input image of an object Q, find images that also contain Q from a large collection of images of objects.*

In practice, we may not find images that contain exactly Q. However, we can find images that contain *Q with high probabilities.* It is important to note that objects may be observed from different perspectives in different images. For example, the input image may depict the front of a vehicle and we need to be able to find images of the same vehicle from the side or rear, as the example images in Figure 1) demonstrates.

This problem lies in the domain of content-based image retrieval (CBIR). There are several commercial image search engines for CBIR: Google Image Search[1] provides general image search, and TinyEye[2] provides near-duplicate image

[1] `https://images.google.com`
[2] `https://www.tineye.com`

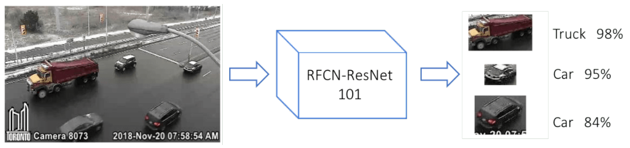


Figure 2: Object detection using RFCN-ResNet 101

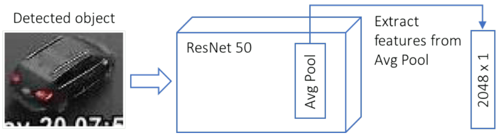


Figure 3: Feature extraction using ResNet 50

search. These search engines are for images found on the web, not for domain-specific images such as traffic camera images. In addition, they are built to find highly-similar images, not objects, which may have different perspectives. For domain-specific CBIR, Pinterest [5] provides a feature for finding images of similar products on its platform. They rely on user-generated text annotations of images to classify images into categories, and retrieve similar images within selected categories. In our case, we do not have annotations to guide us.

2.1 Object Detection and Feature Extraction

A traffic camera image can contain multiple objects. Figure 2 illustrates how we detect and extract objects from each image. We use a Region-based Fully Convolutional Networks (RFCN) [3] for object detection. We choose a 101-layer Residual Network to be the backbone of the network. This model is adopted because RFCN models using Residual Network generally has a good balance between accuracy and speed. With our dataset, 1) the model has high recall rate for object detection and 2) it also has higher inference speed than other high-recall object detection models that we tried, such as Faster R-CNN with Residual Network [8]. Using this model, we detect objects within each image and detect their bounding boxes; then, to extract each object, we crop the image to the bounding box.

Cropped images of extracted objects may not have the same dimension. In addition, we want to represent each object based on its semantic features rather than its raw pixels. Thus we use another CNN model, ResNet-50 [4], to extract the object's features as a fixed-sized vector. ResNet-50 is commonly used for object classification, and it is also used for feature extraction. The last layer of this network (before the classifier layer) collects all the features of images that we have gained from the preceding convolutional layers. This layer is called the Average Pooling Layer in the ResNet-50 architecture. A vector is extracted from the Average Pooling Layer of the model with the classifer layer removed, as shown in Figure 3. This approach is often called "transfer learning", and it has been used by the text-to-image search engine at Etsy [6].

2.2 Nearest Neighbor Search and Refinement

As mentioned earlier, we want to find images containing similar objects. To do so, we use Hierarchical Navigable Small World (HNSW) [7], which is a K-nearest neighbor search index, to efficiently find objects that are highly-likely

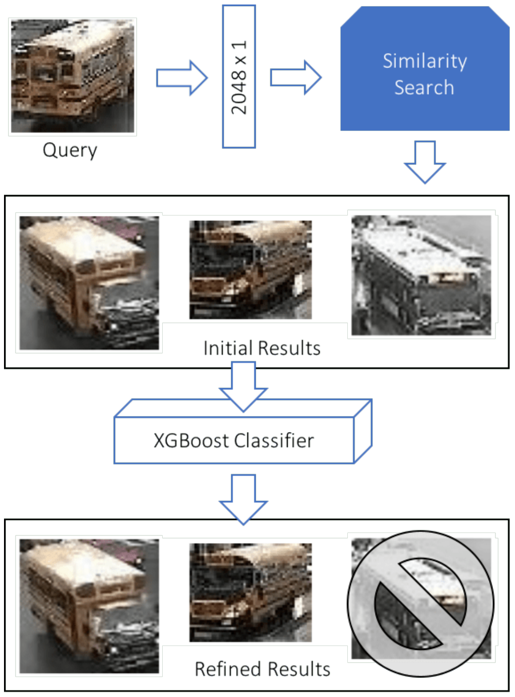


Figure 4: Nearest neighbor search using HNSW and result refinement using the XGBoost Classifier

to be the same object as a query object. The similarity is determined based on the cosine distance of the feature vectors to that of the query object.

The nearest neighbor search gives similar objects even when viewed from a different perspective. However, the precision and recall of this similarity search may be low since we used a CNN pre-trained on general images found on the Web (i.e., ImageNet), rather than on a domain-specific corpus. To improve the accuracy of the search index, we use two approaches: 1) a custom classifier trained on labelled pairs for removing false positives and 2) a technique we call multi-object query for getting more results and avoiding false negatives.

To increase retrieval precision, we train a custom classifier (XGBoost [2]) on labeled pairs of images of objects captured by traffic cameras, as illustrated in Figure 4. During labeling, for every pair of cropped images of objects, we concatenate the two feature vectors of size 2048 x 1, and label it either a 1 if they contain the same object or 0 if different. Manually labeling pairs one-by-one can be very time consuming. To speed it up, we first create 11 vehicle categories and assign images of objects (vehicles) to one of the categories. Then, we label all pairs within each category as 1 and all pairs from different categories as 0. Using this approach, we generated 121,024 labeled pairs.

To improve recall, we use a different strategy by supporting multi-object querying: instead of using a single vehicle's image as a query, the user can select multiple images of the same vehicle (identified by the user), and group them as one query. The final search result is the union of the search results of the individual images in the query. Assuming the probability of a false negative be p_f for each search result, in the recall of k results, the probability of a false negative

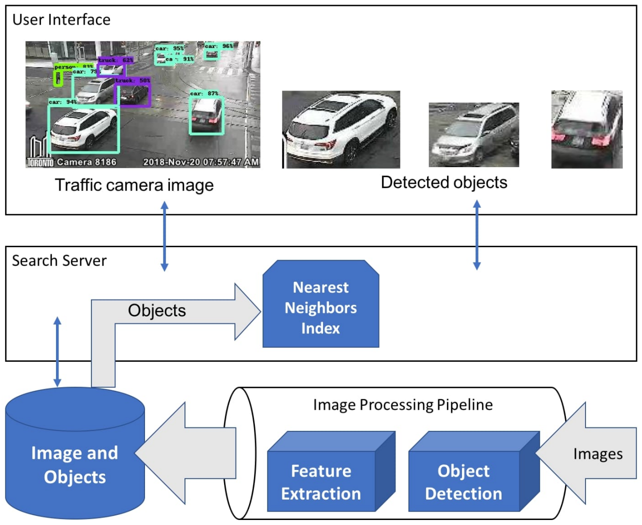


Figure 5: VISE's architecture

would be p_f^k. Because $p_f^k < p_f$, the recall is higher for the union.

3. THE SEARCH ENGINE SYSTEM

As illustrated in Figure 5, the VISE search engine for vehicles consists of three major components: the image processing pipeline, the search server, and the web-based user interface. In this section, we describe the architecture of the image processing pipeline and the search server.

3.1 The Image Processing Pipeline

The image processing pipeline crawls traffic camera images released through Toronto Open Data Portal[3]. The images are from 281 cameras and sampled every two minutes. The images and metadata are first stored in a SQLite[4] database, and then sent to our object detection and feature extraction system running on Tensorflow[5] and two GPUs (GeForce GTX 1080 Ti). The extracted feature vectors are stored to an SQLite database, and then used for calculation of the nearest neighbor index in the backend search server, making them available for users' queries.

The combined object detection and feature extraction has a throughput of approximately 13 images per second, enough to handle the speed at which the images are coming in – 2.5 images per second on average. If the image sampling rate were to increase in the future beyond our current capacity, we could parallelize object detection and feature extraction procedures on multiple GPUs.

Initially, when object detection was performed on each image using a CPU, it took up to 5 seconds per image to run an inference. This was due to the the depth of the Convolution Neural Network, RFCN-ResNet 101, which we were using. We optimized the graph inference by using a GPU, which took up to about 0.10 second per image. Furthermore, we were able to optimize it by adopting TensorRT[6], a high performance neural network inference optimizer and runtime engine, to about 0.07 second per image.

For extracting feature vectors, we use batches of 32 objects, which takes up to 0.25 seconds per batch on our GPU. Thus, the objects' feature vectors are generated at the rate of 128 per second, well within the write throughput of SQLite [1].

To further discuss the sample rate: the City of Toronto currently publicly releases images at two-minute intervals for every camera, but the sample rate used internally is likely much higher. Higher sample rate surely leads to higher recall in finding vehicle. However, it will also lead to higher hardware cost[7] and put stress on memory and storage capacity. The question we would like to answer is what sample rate is "good enough" for a given route, and how to upgrade our system to handle that sample rate.

To calculate the desired sample rate, let us assume the time it takes for a car to travel through any camera's capturing range is t seconds. Then the probability of the car being sampled if it travels through a single camera is $t \cdot s$ where s is the number of samples per second (e.g., 0.0083, or 1 per 2 minutes for the City of Toronto's public release). If the car is traveling in a route with x number of cameras, the probability of capturing the car at least once is $1-(1-\min(1, t \cdot s))^x$. Assuming $t = 5$ and $x = 20$, the probability with respect to sample rate is shown in Figure 6: to achieve 94% probability, it only takes a sample rate of 0.025 (i.e., 3 per 2 minutes, or 3× the current rate). Most importantly, we do not need to update our current pipeline or add new hardware to meet the new throughput requirement (i.e., $2.5 \times 3 = 7.5$ images per second).

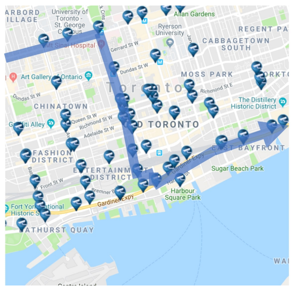
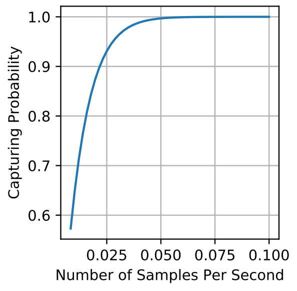
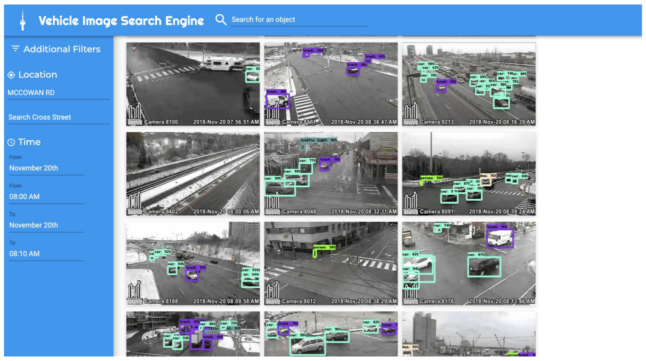



Figure 6: Left: a partial route of a vehicle with cameras; Right: the probability of capturing the vehicle at least once in the route with respect to sample rate.

3.2 The Search Server

The search server hosts the nearest neighbor search index in-memory, as well as handles requests from the web interface. Due to the large number of objects generated every day – approximately 600.000–700.000 objects per day, we only make objects from the last 7 days searchable.

[3]https://www.toronto.ca/city-government/data-research-maps/open-data/
[4]https://www.sqlite.org
[5]https://www.tensorflow.org
[6]https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html
[7]The market price of a new GeForce GTX 1080 Ti GPU is $1,799.98 as of March 2019.



MCCOWAN RD

Search Cross Street

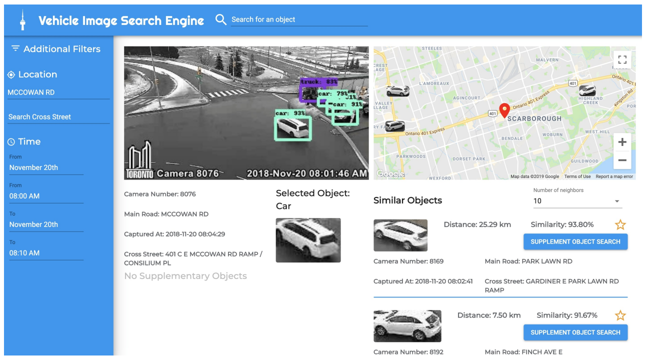
From November 20th

From 08:00 AM

To November 20th

To 08:10 AM

Figure 7: Browsing through traffic camera images



MCCOWAN RD

Search Cross Street

From November 20th

From 08:00 AM

To November 20th

To 08:10 AM

Camera 8076 2018-Nov-20 08:01:46 AM

Camera Number: 8076

Main Road: MCCOWAN RD

Captured At: 2018-11-20 08:04:29

Cross Street: 401 C E MCCOWAN RD RAMP / CONSILIUM PL

No Supplementary Objects

Selected Object: Car

Similar Objects

Number of neighbors: 10

Distance: 25.29 km Similarity: 93.80%

Camera Number: 8169 Main Road: PARK LAWN RD

Captured At: 2018-11-20 08:02:41 Cross Street: GARDINER E PARK LAWN RD RAMP

Distance: 7.50 km Similarity: 91.67%

Camera Number: 8192 Main Road: FINCH AVE E

Figure 8: Inspection of a selected image

The memory usage of the search server including the index is 34.07 GB. The indexing time for 7 days of objects ($\sim$ 4.3 millions) is 22.51 minutes. The average time for a single-object query is 1.5 milliseconds (using a random sample of 1,000).

4. DEMONSTRATION AND CASE STUDY

We now walk through the user interface of VISE through an interesting usage scenario.

After a hit-and-run incident, police officers review available camera footage to trace the vehicle of the suspected offenders. The officers have learned that a white Van at the intersection of Yonge Street and Dundas Street East in Downtown Toronto at 10:30 AM has caused the incident. In order to locate the vehicle in question, the officers initially navigate to the application to see selections of traffic camera images, as shown in Figure 7. The officers query "Van" using the search bar at the top section of the page, in order to retrieve images that contain a van. They then successfully narrow down the search to only images that contain a similar van. The left-side bar contains multiple inputs that enables them to perform advanced search by refining the selection of images by criteria such as the camera number, the time interval in which the images were captured, or the location.

The officers select an image from the initial search results that may contain the vehicle in question, and then they are taken to the inspection page for this vehicle as shown in Figure 8. When they hover their cursor over a box surrounding the vehicle in question, images of the nearest neighbors (highly similar vehicles) are retrieved from the search server, loaded along with metadata such as camera numbers, locations of the streets, and time captured. Highly similar vehicles are retrieved from the nearest neighbor search index using the extracted feature vectors obtained from the image processing pipeline. The search results are ranked by descending order of similarity scores.

To further refine the results based on location, the officers use the interactive map to navigate and view the locations of highly similar vehicles. The vehicle in question (query vehicle) is annotated with a pinpoint marker, while the highly similar vehicles are represented with their image thumbnails. By selecting one of the vehicles, the map computes the distance between the selected and the query vehicles.

Using the map, the officers have identified a candidate match within some reasonable distance from the query vehicle. To gather more candidates, they extend the current result set by using multi-object query: each similar vehicle has a button that adds itself to the query together with the current query vehicle. The new search result set is the union of the nearest neighbors of all query vehicles. Once the officers have identified the most recent location of the vehicle in question, they can quickly deploy ground forces.

5. ACKNOWLEDGEMENTS

This work is partially funded by NSERC.

6. REFERENCES

[1] Database Speed Comparison. `https://www.sqlite.org/speed.html`. Accessed: 2019-03-14.

[2] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, pages 785–794, 2016.

[3] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. In *NIPS*, pages 379–387, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[5] Y. Jing, D. C. Liu, D. Kislyuk, A. Zhai, J. Xu, J. Donahue, and S. Tavel. Visual search at pinterest. In *SIGKDD*, pages 1889–1898, 2015.

[6] C. Lynch, K. Aryafar, and J. Attenberg. Images don't lie: Transferring deep visual semantic features to large-scale multimodal learning to rank. In *SIGKDD*, pages 541–548, 2016.

[7] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016.

[8] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017.