Focus: Querying Large Video Datasets with Low Latency and Low Cost

Kevin Hsieh[†] Ganesh Ananthanarayanan[§] Peter Bodik[§] Shivaram Venkataraman[§ ℵ]
Paramvir Bahl[§] Matthai Philipose[§] Phillip B. Gibbons[†] Onur Mutlu[⋆†]

[†]*Carnegie Mellon University* [§]*Microsoft* [ℵ]*University of Wisconsin* [⋆]*ETH Zürich*

Abstract

Large volumes of video are continuously recorded by cameras deployed for traffic control and surveillance with the goal of answering "after the fact" queries such as: *identify video frames with objects of certain classes (cars, bags)* from many days of recorded video. Current systems for processing such queries on large video datasets incur either high cost at video ingest time or high latency at query time. We present Focus, a system providing both low-cost and low-latency querying on large video datasets. Focus' architecture flexibly and effectively divides the query processing work between *ingest time* and *query time*. At ingest time (on live videos), Focus uses cheap convolutional network classifiers (CNNs) to construct an *approximate index* of all possible object classes in each frame (to handle queries for *any* class in the future). At query time, Focus leverages this approximate index to provide low latency, but compensates for the lower accuracy of the cheap CNNs through the judicious use of an expensive CNN. Experiments on commercial video streams show that Focus is 48× (up to 92×) cheaper than using expensive CNNs for ingestion, and provides 125× (up to 607×) lower query latency than a state-of-the-art video querying system (NoScope).

1. Introduction

Cameras are ubiquitous, with millions of them deployed by public and private entities at traffic intersections, enterprise offices, and retail stores. Videos from these cameras are continuously recorded [2,6], with the main purpose of answering "after-the-fact" queries such as: *identify video frames with objects of certain classes (like cars or bags)* from many days of recorded video. Because the results from these video analytics queries may be needed quickly in many use cases, achieving low latency is crucial.

Advances in convolutional neural networks (CNNs) backed by copious training data and hardware accelerators (e.g., GPUs [12]) have led to highly accurate results in tasks like object detection and classification of images. For instance, the ResNet152 classifier CNN [45], winner of the ImageNet challenge 2015 [73], surpasses human-level performance in classifying $1,000$ object classes on a public image dataset that has labeled ground truths [44].

Despite the accuracy of image classifier CNNs (like ResNet152) and object detectors (like YOLOv2 [68]), using them for video analytics queries is both expensive and slow. For example, even *after* using various motion detection techniques to filter out frames with no moving objects, using an object detector such as YOLOv2 [68] to identify frames with a given class (e.g., ambulance) on a month-long traffic video requires ≈ 190 hours on a high-end GPU (NVIDIA P100 [12]) and costs over $380 in the Azure cloud (Standard_NC6s_v2 instances). To achieve a query latency of say one minute on 190 GPU hours of work would require tens of thousands of GPUs detecting objects in the video frames in parallel, which is two to three orders of magnitude more than what is typically provisioned (few tens or hundreds of GPUs) by traffic jurisdictions or retail stores. Recent work like NoScope [51] has significantly improved the filtering of frames by using techniques like lightweight binary classifiers for the queried class (e.g., ambulance) before running heavy CNNs. However, the latencies are still long, e.g., it takes *5 hours* to query a month-long video on a GPU, in our evaluations. Moreover, videos from many cameras often need to be queried, which increases the latency and the GPU requirements even more.

The objective of our work is to enable *low-latency and low-cost querying over large historical video datasets*.

A natural approach to enable low latency queries is doing most of the work at *ingest-time*, i.e., on the *live* video that is being captured. If object detection, using say YOLO, were performed on frames at ingest-time, queries for specific classes (e.g., ambulance) would involve only a simple *index* lookup to find video frames with the queried object class. There are, however, two main shortcomings with this approach. First, most of the ingest-time work may be wasteful because typically only a small fraction of recorded frames ever get queried [16], e.g., only after an incident that needs investigation. Second, filtering techniques that use binary classifiers (as in NoScope [51]) are ineffective at ingest-time because *any* of a number of object classes could be queried later and running even lightweight binary classifiers for many classes can be prohibitively expensive.

Objectives & Techniques. We present Focus, a system to support low-latency, low-cost queries on large video datasets. To address the above challenges and shortcomings, Focus has the following goals: (a) provide low-cost indexing of *multiple* object classes in the video at ingest-time, (b) achieve high accuracy and low latency for queries, and (c) enable trade-offs between the cost at ingest-time and the latency at query-time. Focus takes as inputs from the user a *ground-truth CNN* (or "GT-CNN", e.g., YOLO) and the desired accuracy of results that Focus needs to achieve relative to the GT-CNN. With these inputs, Focus uses three key techniques to achieve the above goals: (1) an *approximate* indexing scheme at ingest-time using cheap CNNs, (2) redundancy elimination by *clustering* similar objects, and (3) a tunable mechanism for judiciously *trading off* ingest cost and query latency.

(1) *Approximate indexing using a cheap ingest CNN.* To make video ingestion cheap, Focus uses *compressed* and *specialized* versions of the GT-CNN that have fewer convolutional layers [78], use smaller image sizes, and are trained to recognize the classes specific to each video stream. The cheap ingest CNNs, however, are less accurate than the expensive GT-CNN, both in terms of *recall* and *precision*. We define recall as the fraction of frames in the video that contain objects of the queried class that were *actually* returned in the query's results. Precision, on the other hand, is the fraction of frames in the query's results that contain objects of the queried class.

Using a cheap CNN to filter frames upfront risks incorrectly eliminating frames. To overcome this potential loss in recall, Focus relies on an empirical observation: while the top (i.e., most confident) classification results of the cheap CNNs and expensive GT-CNN often do not match, the top result of the expensive CNN often falls within the *top*-K most confident results of the cheap CNN. Therefore, at ingest-time, Focus indexes each frame with the "top-K" results of the cheap CNN, instead of just the top result. To increase precision, at query-time, after filtering frames using the top-K index, we apply the GT-CNN and return only frames that actually contains the queried object class.

(2) *Redundancy elimination via clustering.* To reduce the query-time latency of using the expensive GT-CNN, Focus relies on the significant similarity between objects in videos. For example, a car moving across a camera will look very similar in consecutive frames. Focus leverages this similarity by clustering the objects at ingest-time. We classify *only* the cluster centroids with the GT-CNN at query-time, and assign the same class to all objects in the cluster. This considerably reduces query latency. Clustering, in fact, identifies redundant objects even across non-contiguous and temporally-distant frames.

(3) *Trading off ingest cost vs. query latency.* Focus intelligently chooses its parameters (including K and the cheap ingest-time CNN) to meet user-specified targets on precision and recall. Among the parameter choices that meet the accuracy targets, it allows the user to trade off between ingest cost and query latency. For example, using a cheaper ingest CNN reduces the ingest cost but increases the query latency as Focus needs to use a larger K for the top-K index to achieve the accuracy targets. Focus automatically identifies the "sweet spot" in parameters, which sharply improves one of ingest cost or query latency for a small worsening of the other. It also allows for policies to balance the two, depending on the fraction of videos the application expects to get queried.

In summary, Focus' ingest-time and query-time operations are as follows. At ingest-time, Focus classifies the detected objects using a cheap CNN, clusters similar objects, and indexes each cluster centroid using the top-K most confident classification results, where K is auto-selected based on the user-specified precision, recall, and cost/latency trade-off point. At query-time, Focus looks up the ingest index for cluster centroids that match the class X requested by the user and classifies them using the GT-CNN. Finally, Focus returns all objects from the clusters that are classified as class X to the user.

Evaluation Highlights. We build Focus and evaluate it on fourteen 12-hour videos from three domains – traffic cameras, surveillance cameras, and news. We compare against two baselines: "Ingest-heavy", which uses the heavy GT-CNN for ingest, and "NoScope", a recent state-of-the-art video querying system [51]. We use YOLOv2 [68] as the GT-CNN. On average, across all the videos, Focus is 48× (up to 92×) cheaper than Ingest-heavy and 125× (up to 607×) faster than NoScope, all the while achieving $\geq$ 99% precision and recall. In other words, the latency to query a month-long video drops from 5 hours to only 2.4 minutes, at an ingest cost of $8/month/stream. Figure 1 also shows representative results with different trade-off alternatives for a surveillance video.

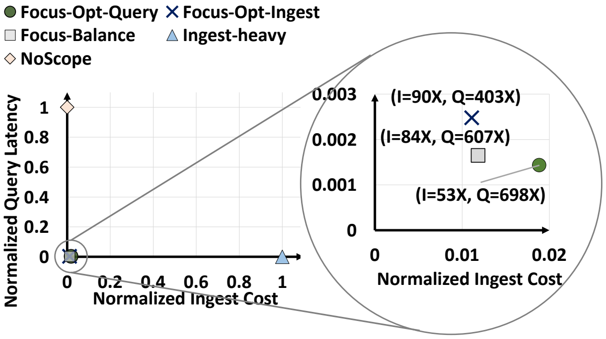

Figure 1: Effectiveness of Focus at reducing both ingest cost and query latency, for an example surveillance video. We compare against two baselines: "Ingest-heavy" that uses the YOLOv2 [68] object detector CNN for ingestion, and "NoScope", the state-of-the-art video querying system [51]. On the left, we see that Focus (the Focus-Balance point) is simultaneously 84× cheaper than Ingest-heavy in its cost (the *I* value) *and* 607× faster than NoScope in query latency (the *Q* value), all the while achieving at least 99% precision and recall (not plotted). Zooming in, also shown are two alternative Focus designs offering different trade-offs, Focus-Opt-Query and Focus-Opt-Ingest, each with at least 99% precision and recall.

Contributions: Our contributions are as follows.

- We present a new architecture for low-cost and low-latency querying over large video datasets, based on a principled split of ingest and query functionalities.
- We propose techniques for efficient indexing of multiple object classes: we create a top-K index at ingest time for high recall, while ensuring high precision by judiciously using expensive CNNs at query time.
- We show new policies that trade off between ingest cost and query latency: our system is significantly cheaper than an ingest-heavy design and significantly faster than query-optimized techniques like NoScope.

2. Background and Motivation

We first provide a brief overview of convolutional Neural Networks, the state-of-the-art approach to detecting and classifying objects in images (§2.1). We then discuss new observations we make about real-world videos, which motivate the design of our techniques (§2.2).

2.1. Convolutional Neural Networks

Convolution Neural Networks (CNNs) are the state-of-the-art method for many computer vision tasks such as object detection and classification (e.g., [45, 53, 59, 68, 84]).

Figure 2 illustrates the architecture of a representative image classification CNN. Broadly, CNNs consist of different types of layers including convolutional layers, pooling layers and fully-connected layers. The output from the final layer of a classification CNN is the probabilities of all object classes (e.g., dog, flower, car), and the class with the highest probability is the predicted class for the object in the input image.

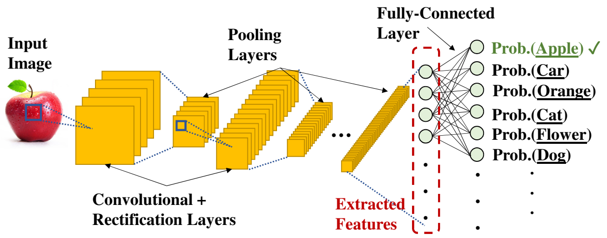

Figure 2: Architecture of an image classification CNN.

The output of the penultimate (i.e., previous-to-last) layer can be considered as "representative features" of the input image [53]. The features are a real-valued vector, with lengths between 512 and 4096 in state-of-the-art classifier CNNs (e.g., [45, 53, 78, 84]). It has been shown that images with similar feature vectors (i.e., small Euclidean distances) are visually similar [24, 53]. Thus, the distance between feature vectors is a standard metric to measure similarity of images in many applications, such as face recognition (e.g., [47]) and image retrieval (e.g., [23, 24, 67]).

Because *inference* using state-of-the-art CNNs is computationally expensive (and slow), two main techniques have been developed to reduce the cost of inference. First, *compression* is a set of techniques that can dramatically reduce the cost of inference at the expense of *accuracy*. Such techniques include removing some expensive convolutional layers [78], matrix pruning [34, 42], reducing input image resolution [68], and others [48, 71]. For example, ResNet18, which is a ResNet152 variant with only 18 layers, is 8× cheaper. Likewise, Tiny YOLO [68], a shallower variant of the YOLO object detector, is 5× cheaper than YOLOv2. However, the tradeoff is that compressed CNNs are usually less accurate than the original CNNs.

The second technique is CNN *specialization* [43], where the CNNs are trained on a subset of a dataset specific to a particular context (such as a video stream). Specialization *simplifies* the task of a CNN because specialized CNNs only need to consider a particular context. For example, differentiating object classes in any possible video is much more difficult than doing so in a traffic video, which is likely to contain far fewer object classes (e.g., cars, bicycles, pedestrians). As a result, specialized CNNs can be *more accurate* and *smaller* at the expense of generality. Leveraging compressed and specialized CNNs is a key facet of our solution (see §4).

2.2. Characterizing Real-world Videos

We aim to support queries of the form: *find all frames in the video that contain objects of class X*. We identify some key characteristics of real-world videos towards supporting these queries: (*i*) large portions of videos can be excluded (§2.2.1), (*ii*) only a limited set of object classes occur in each video (§2.2.2), and (*iii*) objects of the same class have similar feature vectors (§2.2.3). The design of Focus is based on these characteristics.

We analyze six 12-hour videos from three domains: traffic cameras, surveillance cameras, and news channels (§6.1 provides the details.) In this paper, we use results from YOLOv2 [68], trained to classify 80 object classes based on the COCO [60] dataset, as the ground truth.

2.2.1. Excluding large portions of videos. We find considerable potential to avoid processing large portions of videos at query-time. Not all the frames in a video are relevant to a query because each query looks only for a *specific class* of objects. In our video sets, an object class occurs in only 0.16% of the frames on average, and even the most frequent object classes occur in no more than $26\% - 78\%$ of the frames. This is because while there are usually some dominant classes (e.g., cars in a traffic camera, people in a news channel), most other classes are rare. Overall, the above data suggests considerable potential to speed up query latencies by *indexing* frames using the object classes. Also, in our experience, a system for querying videos is more useful for less frequent classes:

querying for "ambulance" in a traffic video is more interesting than querying for something commonplace like "car".

2.2.2. Limited set of object classes in each video. Most video streams have a limited set of objects because each video has its own context (e.g., traffic cameras can have automobiles, pedestrians or bikes, but not airplanes).

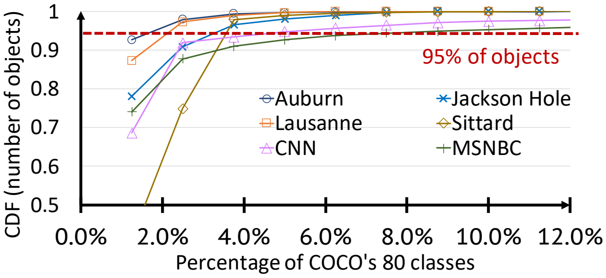

Figure 3: CDF of frequency of object classes. The x-axis is the fraction of classes out of the 80 classes recognized by the COCO [60] dataset (truncated to 12%).

Figure 3 shows the cumulative distribution function (CDF) of the frequency of object classes in our videos (as classified by YOLOv2). We make two observations. First, $2\% - 10\%$ of the most frequent object classes cover $\geq 95\%$ of the objects in all video streams. In fact, for some videos like `Auburn` and `Jackson Hole` we find that only $11\% - 19\%$ object classes occur in the entire video. Thus, for each video stream, if we can *automatically* determine its most frequent object classes, we can train efficient CNNs *specialized* for these classes. Second, a closer analysis reveals that there is little overlap between the object classes among different videos. On average, the Jaccard index [85] (i.e., intersection over union) between the videos based on their object classes is only 0.46. This implies that we need to specialize CNNs for each video stream separately to achieve the most benefits.

2.2.3. Feature vectors for finding duplicate objects. Objects moving in the video often stay in the frame for several seconds; for example, a pedestrian might take 15 seconds to cross a street. Instead of classifying *each instance* of the same object across the frames, we would like to *inexpensively* find duplicate objects and only classify one of them using a CNN (and apply the same label to all duplicates). Thus, given n duplicate objects, we would like only one CNN classification operation instead of n.

Comparing pixel values across frames is an obvious technique to identify duplicate objects, however, this technique turns out to be highly sensitive to even small changes in the camera's view of an object. Instead, feature vectors extracted from the CNNs (§2.1) are more robust because they are specifically trained to extract visual features for classification. We verify the robustness of feature vectors using the following analysis. In each video, for each object i, we find its *nearest* neighbor j using feature vectors from a cheap CNN (ResNet18) and compute the fraction of object pairs that belong to the same class. This fraction is over 99% in each of our videos, which shows the promise of using feature vectors from cheap CNNs to identify duplicate objects *even* across frames that are *not* temporally contiguous.

3. Overview of Focus

The goal of Focus is to *index live video streams* by the object classes occurring in them and enable answering "after-the-fact" queries later on the stored videos of the form: *find all frames that contain objects of class X*. Optionally, the query can be restricted to a subset of cameras and a time range. Such a query formulation is the basis for many widespread applications and could be used either on its own (such as for detecting all cars or bicycles in the video) or used as a basis for further processing (e.g., finding all collisions between cars and bicycles).

System Configuration. Focus is designed to work with a wide variety of current and future CNNs. The user (system administrator) provides a *ground-truth CNN* (GT-CNN), which serves as the accuracy baseline for Focus, but is far too costly to run on every video frame. Through a sequence of techniques, Focus provides results of nearly-comparable accuracy but at greatly reduced cost. In this paper, we use YOLOv2 [68] as the default GT-CNN.

Because different applications require different accuracies, Focus permits the user to specify the accuracy target, while providing reasonable defaults. The accuracy target is specified in terms of *precision*, i.e., fraction of frames output by the query that actually contain an object of class X according to GT-CNN, and *recall*, i.e., fraction of frames that contain objects of class X according to GT-CNN that were actually returned by the query.

Architecture: Figure 4 overviews the Focus design.

- At *ingest-time* (left part of Figure 4), Focus classifies objects in the incoming video frames and extracts their feature vectors. For its ingest, Focus uses highly compressed and specialized alternatives of the GT-CNN model (IT_1 in Figure 4). Focus then clusters objects based on their feature vectors (IT_2) and assigns to each cluster the *top* K most likely classes these objects belong to (based on classification confidence of the ingest CNN) (IT_3). It creates a *top*-K *index*, which maps each class to the set of object clusters (IT_4). The top-K index is the output of Focus' ingest-time processing of videos.
- At *query-time* (right part of Figure 4), when the user queries for a certain class X (QT_1), Focus retrieves the matching clusters from the top-K index (QT_2), runs the *centroids* of the clusters through GT-CNN (QT_3), and returns all frames from the clusters whose centroids were classified by GT-CNN as class X (QT_4).

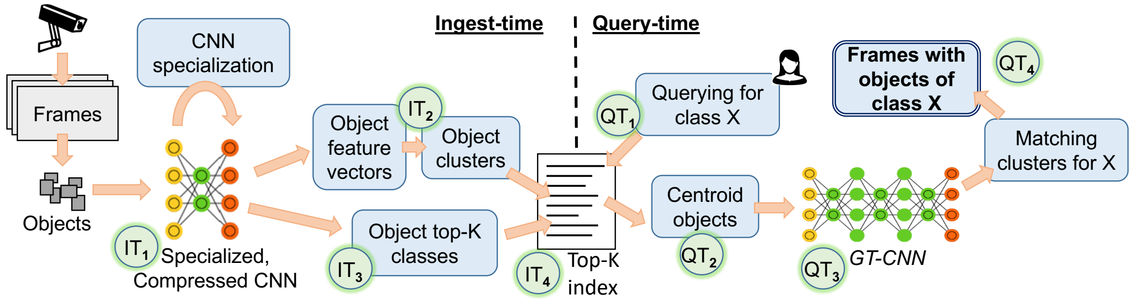

Figure 4: Overview of Focus.

The top-K ingest index is a mapping between the object classes and the clusters. In particular, we create a mapping from each object class to the clusters with top K matching object classes. Separately, we store the mapping between clusters and their corresponding objects and frames. The structure of the index is:

```
object class → ⟨cluster ID⟩
cluster ID → [centroid object, ⟨objects⟩ in
cluster, ⟨frame IDs⟩ of objects]
```

We next explain how Focus' key techniques keep ingest cost and query latency low while also meeting the user-specified recall and precision targets.

1) Top-K index via cheap ingesting: Focus makes indexing at ingest-time cheap by using compressed and specialized alternatives of the GT-CNN for *each* video stream. *Compression* of CNNs [34, 42, 48, 78] uses fewer convolutional layers and other approximations (§2.1), while *specialization* of CNNs [43,75] uses the observation that a specific video stream contains only a small number of object classes and their appearance is more constrained than in a generic video (§2.2.2). Both optimizations are done automatically by Focus and together result in ingest-time CNNs that are up to 96× cheaper than the GT-CNN.

The cheap ingest-time CNNs are less accurate, i.e., their top-most results often do not match the top-most classifications of GT-CNN. Therefore, to improve recall, Focus associates each object with the *top*-K classification results of the cheap CNN, instead of only its top-most result. Increasing K increases recall because the top-most result of GT-CNN often falls within the ingest-time CNN's top-K results. At query-time, Focus uses the GT-CNN to remove objects in this larger set that do not match the class, to regain the precision lost by including the top-K.

2) Clustering similar objects. A high value of K at ingest-time increases the work done at query time, thereby increasing query latency. To reduce this overhead, Focus clusters similar objects at ingest-time using feature vectors from the cheap ingest-time CNN (§2.2.3). In each cluster, at query-time, we run only the cluster centroid through GT-CNN and apply the classified result from the GT-CNN to all objects in the cluster. Thus, a tight clustering of objects is crucial for high precision and recall.

3) Trading off ingest vs. query costs. Focus automatically chooses the ingest CNN, its K, and specialization and clustering parameters to achieve the desired precision and recall targets. These choices also help Focus trade off between the work done at ingest-time and query-time. For instance, to save ingest work, Focus can select a cheaper ingest-time CNN, and then counteract the resultant loss in recall by using a higher K and running the expensive GT-CNN on more objects at query time. Focus chooses its parameters so as to offer a sharp improvement in one of the two costs for a small degradation in the other cost. Because the desired trade-off point is application-dependent, Focus provides users with options: "ingest-optimized", "query-optimized", and "balanced" (the default). Figure 1 (§1) presents an example result.

4. Video Ingest & Querying Techniques

We describe the main techniques used in Focus: constructing approximate indexes with cheap CNNs at ingest-time (§4.1), specializing the CNNs to the specific videos (§4.2), and identifying similar objects and frames to save on redundant CNN processing (§4.3). §4.4 describes how Focus flexibly trades off ingest cost and query latency.

4.1. Approximate Index via Cheap Ingest

Focus indexes the live videos at *ingest-time* to reduce the *query-time* latency. We detect and classify the objects within the frames of the live videos using *ingest-time* CNNs that are far cheaper than the ground-truth GT-CNN. We use these classifications to index objects by class.

Cheap ingest-time CNN. As noted earlier, the user provides Focus with a GT-CNN. Optionally, the user can also provide other CNN architectures to be used in Focus' search for cheap CNNs. Examples include object detector CNNs (which vary in their resource costs and accuracies) like YOLO [68] and Faster RCNN [69] that jointly detect the objects in a frame and classify them. Alternatively, objects can be detected separately using relatively inexpensive techniques like background subtraction [28], which are well-suited for static cameras, as in surveillance or traffic installations, and then the detected objects can be classified using object classification CNN architectures such as ResNet [45], AlexNet [53] and VGG [78].[1]

Starting from these user-provided CNNs, Focus applies various levels of compression, such as removing convolutional layers and reducing the input image resolution (§2.1). This results in a large set of CNN options for ingest, $\{\text{CheapCNN}_1, \ldots, \text{CheapCNN}_n\}$, with a wide range of costs and accuracies, out of which Focus picks its ingest-time CNN, $\text{CheapCNN}_{\text{ingest}}$.

Top-K Ingest Index. To keep recall high, Focus indexes each object using the *top* K object classes from the output of $\text{CheapCNN}_{\text{ingest}}$, instead of using just the top-most class. Recall from §2.1 that the output of the CNN is a list of classes for each object in descending order of confidence. We make the following *empirical* observation: the top-most output of the expensive GT-CNN for an object is often contained within the top-K classes output by the cheap CNN, for a small value of K.

Figure 5 demonstrates the above observation by plotting the effect of K on recall on one of our video streams from a static camera, lausanne (see §6.1). We explore many cheaper ResNet18 [45] models by removing one layer at a time with various input image sizes. The trend is the same among the CNNs we explore so we present three models for clarity: ResNet18, and ResNet18 with 4 and 6 layers removed; correspondingly to each model, the input images were rescaled to 224, 112, and 56 pixels, respectively. These models were also *specialized* to the video stream (more in §4.2). We make two observations.

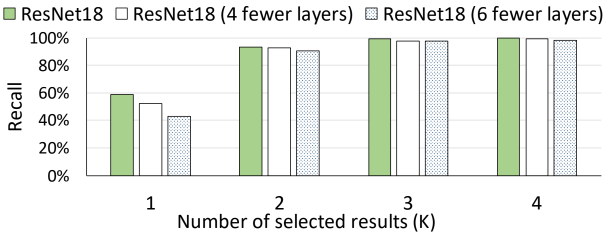

Figure 5: Effect of K on the recall of three cheap classifier CNNs to classify the detected objects. Recall is measured relative to the results of the GT-CNN, YOLOv2 [68].

First, we observe steady increase in recall with increasing K, for all three CheapCNNs. As the figure shows, all the cheap CNNs reach $\geq 99\%$ recall when $\text{K} \geq 4$. Note that all these models recognize 80 classes, so $\text{K} = 4$ represents only 5% of the possible classes. Second, there is a *trade-off* between different models – typically, the cheaper they are, the lower their recall with the same K. However, we can compensate for the loss in recall in cheaper models using a larger K to reach a certain recall value. Overall, we conclude that by selecting the appropriate model and K, Focus can achieve the target recall.

Achieving precision. Focus creates the *top-*K *index* from the top-K classes output by $\text{CheapCNN}_{\text{ingest}}$ for every object at ingest-time. While filtering for objects of the queried class X using the top-K index (with the appropriate K) will have a high recall, this will lead to very low precision. Because we associate each object with K classes (while it has only one true class), the average precision is only $1/\text{K}$. Thus, at query time, to improve precision, Focus determines the *actual* class of objects from the top-K index using the expensive GT-CNN and returns only the objects that match the queried class X.

Skipping GT-CNN for high-confidence indexes. Focus records the prediction confidence along with the top-K results by $\text{CheapCNN}_{\text{ingest}}$. The system can skip invoking GT-CNN for the indexes with prediction confidence higher than a chosen threshold (Skip_{th}). Not invoking GT-CNN for these indexes can cause precision to fall if the threshold is too low. Hence, this parameter needs to be carefully selected to retain high precision.

Parameter selection. The selection of the cheap ingest-time CNN model ($\text{CheapCNN}_{\text{ingest}}$) and the K value (for the top-K results) has a significant influence on the recall of the output produced. Lower values of K reduce recall, i.e., Focus will miss frames that contain the queried objects. At the same time, higher values of K increase the number of objects to classify with GT-CNN at query time, and hence adds to the latency. §4.4 describes how Focus sets these parameters because they have to be jointly set with other parameters described in §4.2 and §4.3.

4.2. Video-specific Specialization of Ingest CNN

To further reduce the ingest cost, Focus *specializes* the ingest-time CNN model to each video stream. As §2.1 describes, model specialization [43] reduces cost by simplifying the task of CNNs. Specifically, model specialization takes advantage of two characteristics in real-world videos. First, most video streams have a limited set of object classes (§2.2.2). Second, objects in a specific stream are often *visually more constrained* than objects in general (say, in the COCO [60] dataset). The cars and buses that occur in a specific traffic camera have much less variability, e.g., they have very similar angle, distortion and size, compared to a generic set of vehicle images. Thus, classifying objects from a specific camera is a much simpler task than doing so from all cameras, resulting in cheaper ingest-time CNNs.

While specializing CNNs to specific videos has been attempted in computer vision research (e.g., [43, 75]), we explain its two key implications within Focus.

1) Lower K values. Because the specialized CNN classifies across fewer classes, they are more accurate, which enables Focus to achieved the desired recall with a much smaller K (for the top-K ingest index). We find that spe-

[1] Focus is agnostic to whether object detection and classification are done together or separately. In practice, the set of detected object bounding boxes (but not their classifications!) remain largely the same with different ingest CNNs, background subtraction, and the GT-CNN.

cialized models can usually use K ≤ 4 (Figure 5), much smaller compared to the typical K needed for *generic* cheap CNNs. A smaller K translates to fewer objects that have to be classified by GT-CNN at query time, thus reducing latency.

2) Most frequent classes. On each video stream, Focus periodically obtains a small sample of video frames and classifies their objects using GT-CNN to estimate the ground truth of the distribution of object classes for the video (similar to Figure 3). From this distribution, Focus selects the most frequently occurring L_s object classes to retrain new specialized models. Because just a handful of classes often account for a dominant majority of the objects (§2.2.2), low values of L_s usually suffice.

While Focus specializes the CNN towards the most frequently occurring L_s classes, we also want to support querying of the *less* frequent classes. For this purpose, Focus includes an additional class called "OTHER" in the specialized model. Being classified as OTHER simply means not being one of the L_s classes. At query time, if the queried class is among the OTHER classes of the ingest CNN's index, Focus extracts all the clusters that match the OTHER class and classifies their centroids through the GT-CNN model.[2]

The parameter L_s (for each video stream) exposes the following trade-off. Using a small L_s enables us to train a simpler model with cheaper ingest cost and lower query-time latency *for the popular classes*, but, it also leads to a larger fraction of objects falling in the OTHER class. As a result, querying for the OTHER class will be expensive because all those objects will have to be classified by the GT-CNN. Using a larger value of L_s, on the other hand, leads to more expensive ingest and query-time models, but cheaper querying for the OTHER classes. We select L_s in §4.4.

4.3. Redundant Object Elimination

At query time, Focus retrieves the objects likely matching the user-specified class from the top-K index and infers their actual class using the GT-CNN. This ensures precision of 100%, but could cause significant latency at query time. Even if this inference were parallelized across many GPUs, it would incur a large cost. Focus uses the following observation to reduce this cost: if two objects are visually similar, their feature vectors are also similar and they would likely be classified as the same class (e.g., cars) by the GT-CNN model (§2.2.3).

Focus *clusters* objects that are similar, invokes the expensive GT-CNN only on the cluster centroids, and assigns the centroid's label to all objects in each cluster. Doing so dramatically reduces the work done by the GT-CNN classifier at query time. Focus uses the feature vector output by the previous-to-last layer of the cheap ingest CNN (see §2.1) for clustering. Note that Focus clusters the *objects* in the frames and not the frames as a whole.[3]

The key questions regarding clustering are *how* we cluster and *when* we cluster. We discuss both below.

Clustering Heuristic. We require two properties in our clustering technique. First, given the high volume of video data, it should be a single-pass algorithm to keep the overhead low, unlike most clustering algorithms, which are *quadratic* complexity. Second, it should make no assumption on the number of clusters and adapt to outliers in data points on the fly. Given these requirements, we use the following simple approach for *incremental* clustering, which has been well-studied in the literature [30, 65].

We put the first object into the first cluster c_1. To cluster a new object i with a feature vector f_i, we assign it to the closest cluster c_j if c_j is at most distance T away from f_i, where T is a distance threshold. However, if none of the clusters are within a distance T, we create a new cluster with centroid at f_i. We measure distance as the L_2 norm [9] between the cluster centroid feature vector and the object feature vector f_i. To bound the time complexity for clustering, we keep the number of clusters actively being updated at a constant C. We do this by *sealing* the smallest cluster when the number of clusters hits $C+1$, but we keep growing the popular clusters (such as similar cars). This maintains the complexity as $O(Cn)$, which is linear in n, the total number of objects. The value of C has a very minor impact on our evaluation results, and we set C as 100 in our evaluations.

Clustering can reduce precision and recall depending on the parameter T. If the centroid is classified by GT-CNN as the queried class X but the cluster contains another object class, it reduces precision. If the centroid is classified as a class different than X but the cluster has an object of class X, it reduces recall. §4.4 discuss setting T.

Clustering at Ingest vs. Query Time. Focus clusters the objects at ingest-time rather than at query-time. Clustering at query-time would involve *storing* all feature vectors, *loading* them for objects filtered from the ingest index and then clustering them. Instead, clustering *at ingest time* creates clusters right when the feature vectors are created and stores only the cluster centroids in the top-K index. This makes the query-time latency much lower and also reduces the size of the top-K index. We observe that the ordering of indexing and clustering operations is mostly *commutative* in practice and has little impact

[2] Specialized CNNs can be retrained quickly on a small dataset. Retraining is relatively infrequent and done once every few days. Also, because there will be considerably fewer objects in the video belonging to the OTHER class, we proportionally re-weight the training data to contain equal number of objects of all the classes.

[3] Recall from §4.1 that Focus' ingest process either (i) employs an object detector CNN (e.g., YOLO) that jointly detects and classifies objects in a frame; or (ii) detects objects with background subtraction and then classifies objects with a classifier CNN (e.g. ResNet). Regardless, we obtain the feature vector from the CNNs for *each object* in the frame.

on recall and precision (we do not present these results due to space constraints). We therefore use ingest-time clustering due to its latency and storage benefits.

4.4. Trading off Ingest Cost and Query Latency

Focus' goals of high recall/precision, low ingest cost and low query latency are affected by its parameters: (i) K, the number of top results from the ingest-time CNN to index an object; (ii) L_s, the number of popular object classes we use to create a specialized model; (iii) CheapCNN$_i$, the specialized ingest-time cheap CNN; (iv) Skip$_{\text{th}}$, the confidence threshold to skip invoking GT-CNN; and (v) T, the distance threshold for clustering objects.

Viable Parameter Choices. Focus first prunes the parameter choices to only those that meet the desired precision and recall targets. Among the five parameters, four parameters (K, L_s, CheapCNN$_i$, and T) impact recall; only T and Skip$_{\text{th}}$ impact precision. Focus samples a representative fraction of the video stream and classifies them using GT-CNN for the ground truth. Next, for each combination of parameter values, Focus computes the precision and recall (relative to GT-CNN's outputs) achievable for each of the object classes, and selects only those combinations that meet the precision and recall targets.

Among the viable parameter choices that meet the precision and recall targets, Focus *balances* ingest- and query-time costs. For example, picking a more accurate CheapCNN$_{\text{ingest}}$ will have higher ingest cost, but lower query cost because we can use a smaller K. Using a less accurate CheapCNN$_{\text{ingest}}$ will have the opposite effect.

Pareto Boundary. Focus identifies "intelligent defaults" that sharply improve one of the two costs for a small worsening of the other cost. Figure 6 illustrates the tradeoff between ingest cost and query latency for one of our video streams. The figure plots all the viable "configurations" (i.e., parameter choices that meet the precision and recall targets) based on their ingest cost (i.e., cost of CheapCNN$_{\text{ingest}}$) and query latency (i.e., the number of clusters that need to be checked at query time according to K, L_s, T and Skip$_{\text{th}}$).

We first extract the *Pareto boundary* [17], which is defined as the set of configurations among which we cannot improve one of the metrics without worsening the other. For example, in Figure 6, the yellow triangles are not Pareto optimal when compared to the points on the dashed line. Focus can discard all non-Pareto configurations because at least one point on the Pareto boundary is better than all non-Pareto points in *both* metrics.

Tradeoff Policies. Focus balances ingest cost and query latency (Balance in Figure 6) by selecting the configuration that minimizes the *sum of ingest cost and query latency*. We measure ingest cost as the compute cycles taken to ingest the video and query latency as the average time (or cycles) required to query the video on the object classes that are recognizable by the ingest CNN. By default, Focus chooses a Balance policy that equally weighs ingest cost and query latency. Users can also provide any other weighted function to optimize their goal.

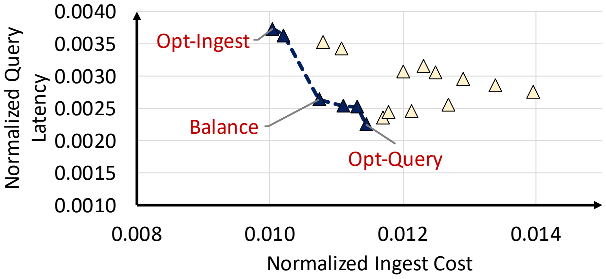

Figure 6: Parameter selection based on the ingest cost and query latency trade-off. The ingest cost is normalized to the cost of ingesting all video frames with GT-CNN (YOLOv2), while the query latency is normalized to the query latency using NoScope. The dashed line is the Pareto boundary.

Focus also allows for other configurations based on the application's preferences and query rates. Opt-Ingest minimizes the ingest cost and is applicable when the application expects most of the video streams to not get queried (such as surveillance cameras), as this policy minimizes the amount of wasted ingest work. On the other hand, Opt-Query minimizes query latency but it incurs a larger ingest cost. More complex policies can be easily implemented by changing how the query latency cost and ingest cost are weighted in our cost function. Such flexibility enables Focus to fit a number of applications.

5. Implementation

Because Focus targets large video datasets, a key requirement of Focus' implementation is the ability to scale and distribute computation across many machines. To this end, we implement Focus as three loosely-coupled modules which handle each of its three key tasks. Figure 7 presents the architecture and the three key modules of Focus: the *ingest processor* (M1), the *stream tuner* (M2), and the *query processor* (M3). These modules can be flexibly deployed on different machines based on the video dataset size and the available hardware resources (such as GPUs). We describe each module in turn.

5.1. Ingest Processor

Focus' ingest processor (M1) generates the approximate index (§4.1) for the input video stream. The work is distributed across many machines, with each machine running one worker process for *each* video stream's ingestion. An ingest processor handles its input video stream with a four-stage pipeline: (i) extracting the moving objects from the video frames (IP$_1$ in Figure 7), (ii) inferring the top-K indexes and the feature vectors of all detected objects with the ingest-time CNN (IP$_2$ in Figure 7, §4.1),

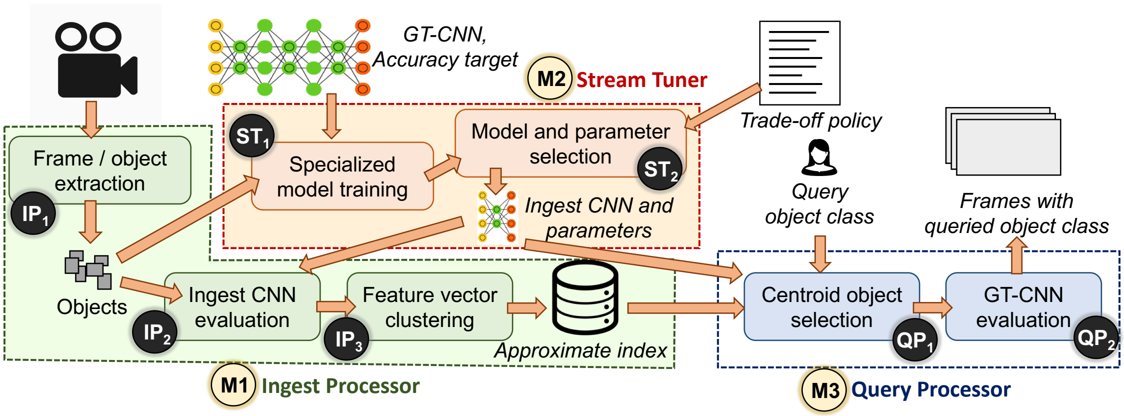

Figure 7: Key components of Focus.

(iii) using the feature vector to cluster objects (IP_3 in Figure 7, §4.3), and (iv) storing the top-K indexes of centroid objects in a database for efficient retrieval at query time.

An ingest processor is configured differently for static (fixed-angle) and moving cameras. For static cameras, we extract object boxes by subtracting each video frame from the *background frame*, which is obtained by averaging the frames in each hour of the video. We then index each object box with an ingest-time *object classifier* CNN. We accelerate the background subtraction with GPUs [14]. We use background subtraction for static cameras because running background subtraction with a cheap object classifier is much faster than running an ingest-time *object detector* CNN, and we find that both approaches have almost the same accuracy in detecting objects in static cameras. Hence, we choose the cheaper ingest option.

For moving cameras, we use a cheap, ingest-time *object detector* CNN (e.g., Tiny YOLO [68]) to generate the approximate indexes. We choose the object detection threshold (the threshold to determine if a box has an object) for the object detector CNN such that we do not miss objects in GT-CNN while minimizing spurious objects.

5.2. Stream Tuner

The stream tuner (M2) determines the ingest-time CNN and Focus' parameters for each video stream (§4.4). It takes four inputs: the sampled frames/objects, the GT-CNN, the desired accuracy relative to the GT-CNN, and the tradeoff policy between ingest cost and query latency (§4.4). Whenever executed, the stream tuner: (i) generates the ground truth of the sampled frames/objects with the GT-CNN; (ii) trains specialized ingest-time CNNs based on the ground truth (ST_1 in Figure 7); and (iii) selects the ingest-time CNN and Focus' parameters (ST_2 in Figure 7).

Focus executes the stream tuner for each video stream *before* launching the corresponding ingest processor. As the characteristics of video streams may change over time, Focus periodically launches the stream tuner to validate the accuracy of the selected parameters on sampled frames. The ingest-time CNN and the system parameters are re-tuned if necessary to meet the accuracy targets.

5.3. Query Processor

The task of the query processor is to return the video frames that contain the user's queried object class. In response to a user query for class X, the query processor first retrieves the centroid objects with matching approximate indexes (QP_1 in Figure 7), and then uses the GT-CNN to determine the frames that do contain object class X (QP_2 in Figure 7, §4.1). The GT-CNN evaluation can be easily distributed across many machines, if needed.

We employ two optimizations to reduce the overhead of GT-CNN evaluation. First, we skip the GT-CNN evaluation for high-confidence indexes (§4.1). Second, we apply a query-specialized binary classifier [51] on the frames that need to be checked before invoking the GT-CNN. These two optimizations make the query processor more efficient by *not* running GT-CNN on all candidate centroid objects.

6. Evaluation

We evaluate our Focus prototype with more than 160 hours of videos from 14 real video streams that span traffic cameras, surveillance cameras, and news channels. Our main results are:

- Focus is simultaneously 48× cheaper on average (up to 92×) than the Ingest-heavy baseline in processing videos and 125× faster on average (up to 607×) than NoScope [51] in query latency — all the while achieving at least 99% precision and recall (§6.2, §6.3).
- Focus provides a rich trade-off space between ingest cost and query latency. If a user wants to optimize for *ingest cost*, Focus is 65× cheaper on average (up to 96×) than the Ingest-heavy baseline, while reducing query latency by 100× on average. If the goal is to optimize for *query latency*, Focus can achieve 202× (up to 698×) faster queries than NoScope with 53× cheaper ingest. (§6.4).

Table 1: Video dataset characteristics

Type	Camera	Name	Description
Traffic	Static	auburn_c	A commercial area intersection in the City of Auburn [5]
		auburn_r	A residential area intersection in the City of Auburn [4]
		bellevue_d	A downtown intersection in the City of Bellevue. The video streams are obtained from city traffic cameras.
		bellevue_r	A residential area intersection in the City of Bellevue
		bend	A road-side camera in the City of Bend [7]
		jackson_h	A busy intersection in Jackson Hole [8]
		jackson_ts	A night street in Jackson Hole. The video is downloaded from the NoScope project website [50].
Surveillance	Static	coral	An aquarium video downloaded from the NoScope project website [50]
		lausanne	A pedestrian plaza (Place de la Palud) in Lausanne [10]
		oxford	A bookshop street in the University of Oxford [15]
		sittard	A market square in Sittard [3]
News	Moving	cnn	News channel
		foxnews	News channel
		msnbc	News channel

6.1. Methodology

Software Tools. We use OpenCV 3.4.0 [13] to decode the videos into frames, and we feed the frames to our evaluated systems, Focus and NoScope. Focus runs and trains CNNs with Microsoft Cognitive Toolkit 2.4 [64], an open-source deep learning system. Our ingest processor (§5.1) stores the approximate index in MongoDB [11] for efficient retrieval at query time.

Video Datasets. We evaluate 14 video streams that span across traffic cameras, surveillance cameras, and news channels. We record each video stream for 12 hours to cover both day time and night time. Table 1 summarizes the video characteristics. We strengthen our evaluation by including down sampling (or frame skipping), one of the most straightforward approaches to reduce ingest cost and query latency, into our evaluation *baseline*. Specifically, as the vast majority of objects show up for at least one second in our evaluated videos, we evaluate each video at 1 fps instead of 30 fps. We find that the object detection results at these two frame rates are almost the same. Each video is split evenly into a *training set* and a *test set*. The training set is used to train video-specialized CNNs and select system parameters. We then evaluate the systems with the test set. In some figures, we show results for only eight representative videos to improve legibility.

Accuracy Target. We use YOLOv2 [68], a state-of-the-art object detector CNN, as our ground-truth CNN (GT-CNN): all objects detected by GT-CNN are considered to be the correct answers.[4] For each query, our default accuracy target is 99% recall and precision. To avoid overfitting, we use the *training set* of each video to explore system parameters with various recall/precision targets (i.e., 100%–95% with a 0.5% step), and we report the best system parameters that can *actually* achieve the recall/precision target on the *test set*. We also evaluate other recall/precision targets such as 97% and 95% (§6.5).

Baselines. We use baselines at two ends of the design spectrum: (1) Ingest-heavy, the baseline system that uses GT-CNN to analyze all frames at ingest time, and stores the results as an index for query; and (2) NoScope, a recent state-of-the-art querying system [51] that analyzes frames for the queried object class at query time. We also use a third baseline, Ingest-NoScope that uses NoScope's techniques at ingest time. Specifically, Ingest-NoScope runs the binary classifiers of NoScope for all possible classes at *ingest time*, invokes GT-CNN if any of the binary classifiers cannot produce a high-confidence result, and stores the results as an index for query. To further strengthen the baselines, we augment all baseline systems with background subtraction, thus eliminating frames with no motion. As Focus is in the middle of the design spectrum, we compare Focus' ingest cost with Ingest-heavy and Ingest-NoScope, and we compare Focus' query latency with NoScope.

Metrics. We use two performance metrics. The first metric is *ingest cost*, the end-to-end machine time to ingest each video. The second metric is *query latency*, the end-to-end latency for an object class query. Specifically, for each video stream, we evaluate the object classes that collectively make up 95% of the detected objects in GT-CNN. We report the average query latency on these object classes. We do not evaluate the bottom 5% classes because they are often random erroneous results in GT-CNN (e.g., "broccoli" or "orange" in a traffic camera).

Both metrics include the time spent on all processing stages, such as detecting objects with background subtraction, running CNNs, clustering, reading and writing to the approximate index, etc. Similar to prior work [51, 68], we report the end-to-end execution time of each system while excluding the video decoding time, as the decoding time can be easily accelerated with GPUs or accelerators.

Experimental Platform. We run the experiments on Standard_NC6s_v2 instances on the Azure cloud. Each instance is equipped with a high-end GPU (NVIDIA Tesla P100), 6-core Intel Xeon CPU (E5-2690), 112 GB RAM, a 10 GbE NIC, and runs 64-bit Ubuntu 16.04 LTS.

6.2. End-to-End Performance

Static Cameras. We first show the end-to-end performance of Focus on static cameras when Focus aims to balance these two metrics (§4.4). Figure 8 compares the ingest cost of Focus and Ingest-NoScope with Ingest-heavy and the query latency of Focus with NoScope. We make three main observations.

[4] We do not use the latest YOLOv3 or other object detector CNN such as FPN [59] as our GT-CNN because one of our baseline systems, NoScope, comes with the YOLOv2 code. Fundamentally, there is no restriction on the selection of GT-CNN for Focus.

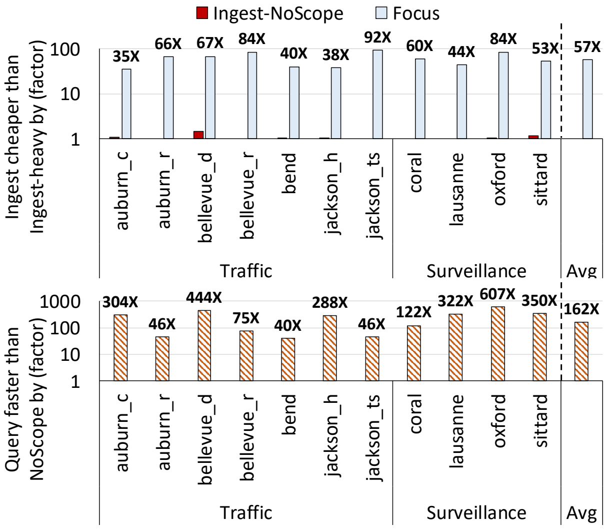

Figure 8: (Top) Focus ingest cost compared to Ingest-heavy. (Bottom) Focus query latency compared to NoScope.

First, Focus significantly improves query latency with a very small cost at ingest time. Focus achieves 162× speedup (on average) in query latency over NoScope with a very small ingest cost (57× cheaper than Ingest-heavy, on average), all the while retaining 99% recall and precision (not shown). Focus achieves two orders of magnitude speedup over NoScope because: (i) the ingest-time approximate indexing drastically narrows down the frames that need to be checked at query time; and (ii) the feature-based clustering further reduces the redundant work. In contrast, NoScope needs to go through *all* the frames at query time, which is especially inefficient for the object classes that appear infrequently. We conclude that Focus' architecture provides a valuable trade-off between ingest cost and query latency.

Second, directly applying NoScope's techniques at ingest time (Ingest-NoScope) does not save much cost over Ingest-heavy. There are two reasons for this: (1) While each binary classifier is relatively cheap, running multiple instances of binary classifiers (for all possible object classes) imposes non-trivial cost. (2) The system needs to invoke GT-CNN when any one of the binary classifiers cannot derive the correct answer. As a result, GT-CNN is invoked for most frames. Hence, the ingest cost of Focus is much cheaper than both, Ingest-heavy and Ingest-NoScope. This is because Focus' architecture only needs to construct the approximate index at ingest time which can be done cheaply with an ingest-time CNN.

Third, Focus is effective across videos with varying characteristics. It makes queries 46× to 622× faster than NoScope with a very small ingest cost (35× to 92× cheaper than Ingest-heavy) among busy intersections (`auburn_c`, `bellevue_d` and `jackson_h`), normal intersections (`auburn_r`, `bellevue_r`, `bend`), a night street (`jackson_ts`), busy plazas (`lausanne` and `sittard`), a university street (`oxford`), and an aquarium (`coral`). The gains in query latency are smaller for some videos (`auburn_r`, `bellevue_r`, `bend`, and `jackson_ts`). This is because Focus' ingest CNN is less accurate on these videos, and Focus selects more conservative parameters (e.g., a larger K such as 4–5 instead of 1–2) to attain the recall/precision targets. As a result, there is more work at query time for these videos. Nonetheless, Focus still achieves at least 40× speedup over NoScope in query latency. We conclude that the core techniques of Focus are general and effective on a variety of real-world videos.

Moving Cameras. We evaluate the applicability of Focus on moving cameras using three news channel video streams. These news videos were recorded with moving cameras and they change scenes between different news segments. For moving cameras, we use a cheap object detector (Tiny YOLO, which is 5× faster than YOLOv2 for the same input image size) as our ingest-time CNN. Figure 9 shows the end-to-end performance of Focus on moving cameras.

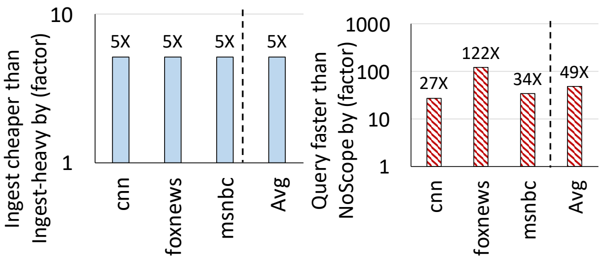

Figure 9: Focus performance on moving cameras. (Left) Focus ingest cost compared to Ingest-heavy. (Right) Focus query latency compared to NoScope.

As the figure shows, Focus is effective in reducing query latency with only a modest ingest cost. Focus achieves a 49× speedup in query latency on average over NoScope, with ingest cost that is 5× cheaper than Ingest-heavy. We make two main observations. First, the ingest cost improvements on moving cameras (5×) is lower than the ones on static cameras (57×). This is because moving cameras require a detector CNN to detect objects, and it is more costly to run a cheap object detector (like Tiny YOLO) as opposed to using background subtraction to detect the objects and then classifying them using a cheap classifier CNN (like compressed ResNet18). Our design, however, does not preclude using much cheaper detectors than Tiny YOLO, and we can further reduce the ingest cost of moving cameras by exploring even cheaper object detector CNNs. Second, Focus' techniques are very effective in reducing query latency on moving cameras. The approximate index generated by a cheap detector CNN significantly narrows down the frames that need to be checked at query time. We conclude that the techniques of Focus are general and can be applied to a wide range of object detection CNNs and camera types.

Averaging over both static and moving cameras, Focus' ingest cost is 48× cheaper than Ingest-heavy and its queries are 125× faster than NoScope.

We now take a deeper look at Focus' performance using representative static cameras.

6.3. Effect of Different Focus Components

Figure 10 shows the breakdown of query latency gains for two core techniques of Focus: (1) Approximate indexing, which indexes each object with the top-K results of the ingest-time CNN, and (2) Approximate indexing + Clustering, which adds feature-based clustering at ingest time to reduce redundant work at query time. We show the results that achieve at least 99% recall and precision. We make two observations.

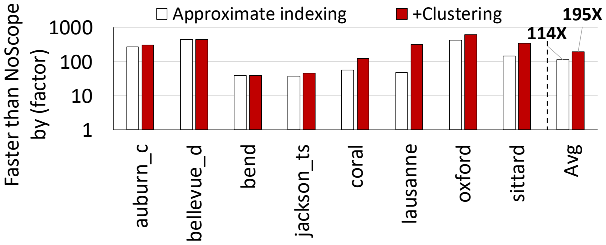

Figure 10: Effect of different Focus components on query latency reduction

First, approximate indexing is the major source of query latency improvement. This is because approximate indexing effectively eliminates irrelevant objects for each query and bypasses the query-time verification for high-confidence ingest predictions. As a result, only a small fraction of frames need to be resolved at query time. On average, approximate indexing alone is 114× faster than NoScope in query latency.

Second, clustering is a very effective technique to further reduce query latency. Using clustering (on top of approximate indexing) reduces the query latency by 195×, significantly better than approximate indexing alone. We see that clustering is especially effective on surveillance videos (e.g., `coral`, `lausanne`, and `oxford`) because objects in these videos tend to stay longer in the camera (e.g., "person" on a plaza compared to "car" in traffic videos), and hence there is more redundancy in these videos. This gain comes with a negligible cost because we run our clustering algorithm (§4.3) on the otherwise idle CPUs of the ingest machine while the GPUs run the ingest-time CNN model.

6.4. Ingest Cost vs. Query Latency Trade-off

One of the important features of Focus is the flexibility to tune its system parameters to achieve different application goals (§4.4). Figure 11 (the zoom-in region of Figure 1) depicts three alternative settings for Focus that illustrate the trade-off space between ingest cost and query latency, using the `oxford` video stream: (1) Focus-Opt-Query, which optimizes for query latency by increasing ingest cost, (2) Focus-Balance, which is the default option that balances these two metrics (§4.4), and (3): Focus-Opt-Ingest, which is the opposite of Focus-Opt-Query. The results are shown relative to the Ingest-heavy and NoScope baselines. Each data label (I,Q) indicates its ingest cost is $I\times$ cheaper than Ingest-heavy, while its query latency is $Q\times$ faster than NoScope.

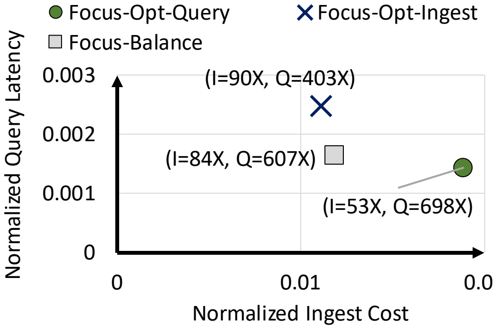

Figure 11: Focus' trade-off policies on an example video

As Figure 11 shows, Focus offers very good options in the trade-off space between ingest cost and query latency. Focus-Opt-Ingest is 90× cheaper than Ingest-heavy, and makes the query 403× faster than a query-optimized system (NoScope). On the other hand, Focus-Opt-Query reduces query latency even more (by 698×) but it is still 53× cheaper than Ingest-heavy. As these points in the design space are all good options compared to the baselines, such flexibility enables a user to tailor Focus for different contexts. For example, a camera that requires fast turnaround time for queries can use Focus-Opt-Query, while a video stream that will be queried rarely would choose Focus-Opt-Ingest to reduce the amount of wasted ingest cost in exchange for longer query latencies.

Figure 12 shows the (I,Q) values for both Focus-Opt-Ingest (Opt-I) and Focus-Opt-Query (Opt-Q) for the representative videos. As the figure shows, the flexibility to make different trade-offs exists in most other videos. On average, Focus-Opt-Ingest is 65× (up to 96×) cheaper than Ingest-heavy in ingest cost while providing 100× (up to 443×) faster queries. Focus-Opt-Query makes queries 202× (up to 698×) faster with a higher ingest cost (53× cheaper than Ingest-heavy). Note that there is no fundamental limitation on the spread between Focus-Opt-Query and Focus-Opt-Ingest as we can expand the search space for ingest-time CNNs to further optimize ingest cost at the expense of query latency (or vice versa). We conclude that Focus enables flexibly optimizing for ingest cost or query latency for application's needs.

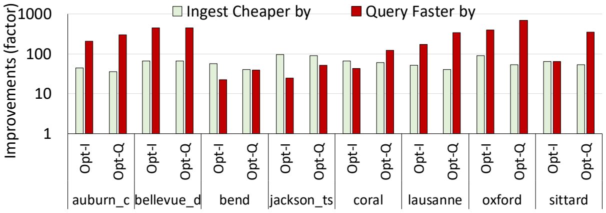

Figure 12: Ingest cost vs. query latency trade-off

It is worth noting that the fraction of videos that get queried can affect the applicability of Focus, especially in the case where only a tiny fraction of videos gets queried. While Focus-Opt-Ingest can save the ingest cost by up to 96×, it can be more costly than any purely query-time-only solution if the fraction of videos that gets queried is less than $\frac{1}{96} \approx 1\%$. In such a case, a user can still use Focus to significantly reduce query latency, but the cost of Focus can be higher than query-time-only solutions.

6.5. Sensitivity to Recall/Precision Target

Figure 13 illustrates Focus' reduction in query latency compared to the baselines under different recall/precision targets. Other than the default 99% recall and precision target, we evaluate both Focus and NoScope with two lower targets, 97% and 95%.

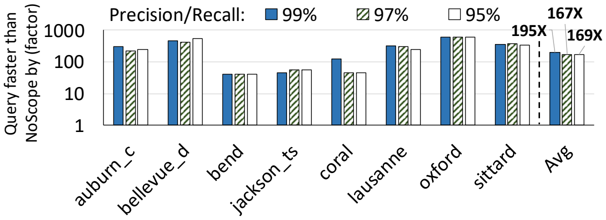

Figure 13: Sensitivity of query latency reduction to recall/precision target

We observe that with lower accuracy targets, the query latency improvement decreases slightly for most videos, while the ingest cost improvement does not change much (not graphed). The ingest cost is not sensitive to the accuracy target because Focus still runs similar ingest CNNs. NoScope can however apply more aggressive query-time optimization to reduce query latency given lower accuracy targets. This decreases Focus' improvement over NoScope for several videos. On average, Focus is faster than NoScope in query latency by 195×, 167×, and 169× with recall/precision of 99%, 97%, and 95%, respectively. We conclude that Focus' techniques can achieve significant improvements on query latency, irrespective of recall/precision targets.

6.6. Sensitivity to Object Class Numbers

We use the 1000 object classes in the ImageNet dataset [73] to study the sensitivity of Focus' performance to the number of object classes (compared to the 80 default object classes in the COCO [60] dataset). Our result shows that Focus is 15× faster (on average) in query latency and 57× cheaper (on average) in ingest cost than the baseline systems, while achieving 99% recall and precision. We observe that the query latency improvements with 1000 object classes is lower than the ones with 80 object classes. The reason is that ingest-time CNNs are less accurate on more object classes, and we need to select a larger K to achieve the target recall. Nonetheless, the improvements of Focus are robust with more object classes as Focus is over one order of magnitude faster than the baseline systems when differentiating 1000 object classes.

7. Other Applications

Applications that leverage CNNs to process large and continuously growing data share similar challenges as Focus. Examples of such applications are:

1) Video and audio. Other than querying for objects, many emerging video applications are also based on CNNs, such as event detection (e.g., [90]), emotion recognition (e.g., [49]), video classification (e.g., [52]), and face recognition (e.g., [74]). Audio applications such as speech recognition (e.g., [19]) are also based on CNNs.

2) Bioinformatics and geoinformatics. Many bioinformatics and geoinformatics systems leverage CNNs to process a large dataset, such as anomaly classification in biomedical imaging (e.g., [57, 72]), information decoding in biomedical signal recordings (e.g., [82]), and pattern recognition in satellite imagery (e.g., [20, 35]).

Naturally, these applications need to answer user-specified queries, such as "find all brain signal recordings with a particular perception" or "find all audio recordings with a particular keyword". Supporting these queries faces similar challenges to Focus, as a system either: (i) generates a precise index at ingest time, which incurs *high cost*; or (ii) does most of the heavy-lifting at query time, which results in *high query latency*. Hence, Focus' architecture offers a low-cost and low-latency option: building an approximate index with cheap CNNs at ingest time and generating precise results based on the approximate index at query time. While the indexing structure may need to be adapted to different applications, we believe Focus' architecture and techniques can benefit many of these emerging applications.

8. Related Work

To our knowledge, Focus is the first system that offers low-cost and low-latency queries for CNN-based object detection in videos by effectively splitting query-processing work between ingest time and query time. We discuss key works related to our system.

1) Cascaded classification. Various works in vision research propose speeding up classification by cascading a series of classifiers. Viola et al. [88] is the earliest work that cascades a series of classifiers (from the simplest to the most complicated) to quickly disregard regions in an image. Many improvements follow (e.g., [58, 91, 92]). CNNs are also cascaded (e.g., [29, 43, 56, 83]) to reduce object detection latency. Our work is different in two major ways. First, we *decouple* the compressed CNN from the GT-CNN, which enables us to choose from a wider range of ingest-time CNNs and thus enables better trade-offs between ingest cost and query latency, a key aspect of our work. Second, we cluster similar objects using CNN features to eliminate redundant work, which is an effective technique for video streams.

2) Context-specific model specialization. Context-specific specialization of models can improve accuracy [63] or speed up inference [43, 51, 75]. Among these, the closest to our work is NoScope [51]. NoScope optimizes for the specified class at query-time using lightweight binary classifiers. In contrast, Focus' architecture splits work between ingest and query times, leading to two orders of magnitude lower latency (§6). To achieve these gains, Focus uses techniques to index *all possible classes* at ingest-time, and thus can handle any class that will get queried in the future. Focus' indexing is especially effective for less frequent object classes, which is arguably of more interest for video querying systems.

3) Stream processing systems. Systems for general stream data processing (e.g., [1,18,22,25,31,32,61,66,86, 87,95]) and specific to video stream analytics (e.g., [96]) mainly focus on general stream processing challenges such as load shedding, fault tolerance, distributed execution, or limited network bandwidth. In contrast, our work is specific to querying on recorded video data with ingest and query trade-offs, and, thus, mostly orthogonal. Focus coud be integrated with one of these general stream processing systems.

4) Video indexing and retrieval. A large body of work in multimedia and information retrieval research proposes various content-based video indexing and retrieval techniques to facilitate queries on videos (e.g., [46,55,80,81]). Among them, most works focus on indexing videos for different types of queries, such as shot boundary detection (e.g., [94]), semantic video search (e.g., [33,37,41]), video classification (e.g., [27]), spatio-temporal information-based video retrieval (e.g., [38, 70]) or subsequence similarity search (e.g., [76, 97]). Some works (e.g., [36, 79]) focus on the query interface to enable querying by keywords, concepts, or examples. These works are largely orthogonal to our work because we focus on reducing *cost and latency* of CNN-based video queries, not on creating an indexing structure for new query types or query interfaces. We believe our approach of splitting ingest-time and query-time work can be extended to many different types of video queries (§7).

5) Database indexing. Using index structures to reduce query latency [77] is a commonly-used technique in conventional databases (e.g., [26, 54]), key-value databases (e.g., [62]), similarity search (e.g., [39,40]), graph queries (e.g., [93]), genome analysis (e.g., [21, 89]), and many others. Our Ingest-heavy and Ingest-NoScope baselines are also examples that index all video frames at ingest time. While queries are naturally faster with these baselines, they are too costly and are potentially wasteful for large-scale videos. In contrast, our work offers new trade-off options between ingest cost and query latency by creating low-cost approximate indexes at ingest time and retaining high accuracy with little work at query time.

9. Conclusion

Answering queries of the form, *find me frames that contain objects of class X*, is an important workload on recorded video datasets. Such queries are used by analysts and investigators for various immediate purposes, and it is crucial to answer them with low latency and low cost. We present Focus, a system that flexibly divides the query processing work between ingest time and query time. Focus performs low-cost ingest-time analytics on live video that later facilitates low-latency queries on the recorded videos. At ingest time, Focus uses cheap CNNs to construct an *approximate index* of all possible object classes in each frame to retain high recall. At query time, Focus leverages this approximate index to provide low latency, but compensates for the lower precision by judiciously using expensive CNNs. This architecture enables orders-of-magnitude faster queries with only a small investment at ingest time, and allows flexibly trading off ingest cost and query latency. Our evaluations using real-world videos from traffic, surveillance, and news domains show that Focus reduces ingest cost on average by 48× (up to 92×) and makes queries on average 125× (up to 607×) faster compared to state-of-the-art baselines. We conclude that Focus' architecture and techniques make it a highly practical and effective approach to querying large video datasets. We hope that the ideas and insights behind Focus can be applied to designing efficient systems for many other forms of querying on large and continuously-growing datasets in many domains, such as audio, bioinformatics, and geoinformatics.

Acknowledgments

We thank our shepherd, Andrew Warfield, and the anonymous OSDI reviewers for their valuable and constructive suggestions. We acknowledge the support of our industrial partners: Google, Huawei, Intel, Microsoft, and VMware. This work is supported in part by NSF and Intel STC on Visual Cloud Systems (ISTC-VCS).

References

[1] Apache Storm. http://storm.apache.org/index.html.

[2] Avigilon. http://avigilon.com/products/.

[3] City Cam, WebcamSittard: Town Square Sittard (NL). https://www.youtube.com/watch?v=iKxhsl3rurA.

[4] City of Auburn North Ross St and East Magnolia Ave. https://www.youtube.com/watch?v=cjuskMMYlLA.

[5] City of Auburn Toomer's Corner Webcam. https://www.youtube.com/watch?v=yJAk_FozAmI.

[6] Genetec. https://www.genetec.com/.

[7] Greenwood Avenue Bend, Oregon. https://www.youtube.com/watch?v=YqyERQwXA3U.

[8] Jackson Hole Wyoming USA Town Square. https://www.youtube.com/watch?v=K-F4CeVsWHA.

[9] L2̂ Norm. http://mathworld.wolfram.com/L2-Norm.html.

[10] Lausanne, Place de la Palud. https://www.youtube.com/watch?v=7uF7DsUQ9vc.

[11] MongoDB. https://www.mongodb.com/.

[12] Nvidia Tesla P100. http://www.nvidia.com/object/tesla-p100.html.

[13] Opencv 3.4. http://opencv.org/opencv-3-4.html.

[14] OpenCV GPU-accelerated computer vision. https://docs.opencv.org/2.4/modules/gpu/doc/gpu.html.

[15] Oxford Martin School Webcam - Broad Street, Oxford. https://www.youtube.com/watch?v=Qhq4vQdfrFw.

[16] Top Video Surveillance Trends for 2016. https://technology.ihs.com/api/binary/572252.

[17] Wikipedia: Pareto efficiency. https://en.wikipedia.org/wiki/Pareto_efficiency.

[18] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the Borealis stream processing engine. In *CIDR*, 2005.

[19] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *ICASSP*, 2012.

[20] A. Albert, J. Kaur, and M. C. Gonzalez. Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale. In *SIGKDD*, 2017.

[21] C. Alkan, J. M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J. O. Kitzman, C. Baker, M. Malig, O. Mutlu, S. C. Sahinalp, R. A. Gibbs, and E. E. Eichler. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature genetics*, 2009.

[22] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, P. Selo, Y. Park, and C. Venkatramani. SPC: A distributed, scalable platform for data mining. In *DM-SSP*, 2006.

[23] A. Babenko and V. S. Lempitsky. Aggregating deep convolutional features for image retrieval. In *ICCV*, 2015.

[24] A. Babenko, A. Slesarev, A. Chigorin, and V. S. Lempitsky. Neural codes for image retrieval. In *ECCV*, 2014.

[25] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. MacroBase: Prioritizing attention in fast data. In *SIGMOD*, 2017.

[26] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. In *SIGFIDET*, 1970.

[27] D. Brezeale and D. J. Cook. Automatic video classification: A survey of the literature. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 2008.

[28] S. Brutzer, B. Höferlin, and G. Heidemann. Evaluation of background subtraction techniques for video surveillance. In *CVPR*, 2011.

[29] Z. Cai, M. J. Saberian, and N. Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection. In *ICCV*, 2015.

[30] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM International Conference on Data Mining*, 2006.

[31] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams - A new class of data management applications. In *VLDB*, 2002.

[32] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing. In *SIGMOD*, 2003.

[33] S. Chang, W. Ma, and A. W. M. Smeulders. Recent advances and challenges of semantic image/video search. In *ICASSP*, 2007.

[34] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015.

[35] G. Cheng, Y. Wang, S. Xu, H. Wang, S. Xiang, and C. Pan. Automatic road detection and centerline extraction via cascaded end-to-end convolutional neural network. *IEEE Trans. Geoscience and Remote Sensing*, 2017.

[36] M. G. Christel and R. M. Conescu. Mining novice user activity with TRECVID interactive retrieval tasks. In *CIVR*, 2006.

[37] S. Dagtas, W. Al-Khatib, A. Ghafoor, and R. L. Kashyap. Models for motion-based video indexing and retrieval. *IEEE Trans. Image Processing*, 2000.

[38] M. E. Dönderler, Ö. Ulusoy, and U. Güdükbay. Rule-based spatiotemporal query processing for video databases. *VLDB J.*, 2004.

[39] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[40] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[41] A. Hampapur. Semantic video indexing: Approach and issue. *SIGMOD Record*, 1999.

[42] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[43] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *MobiSys*, 2016.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[46] W. Hu, N. Xie, L. Li, X. Zeng, and S. J. Maybank. A survey on visual content-based video indexing and retrieval. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 2011.

[47] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[48] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866, 2014.

[49] S. E. Kahou, C. J. Pal, X. Bouthillier, P. Froumenty, Ç. Gülçehre, R. Memisevic, P. Vincent, A. C. Courville, Y. Bengio, R. C. Ferrari, M. Mirza, S. Jean, P. L. Carrier, Y. Dauphin, N. Boulanger-Lewandowski, A. Aggarwal, J. Zumer, P. Lamblin, J. Raymond, G. Desjardins, R. Pascanu, D. Warde-Farley, A. Torabi, A. Sharma, E. Bengio, K. R. Konda, and Z. Wu. Combining modality specific deep neural networks for emotion recognition in video. In *ICMI*, 2013.

[50] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. NoScope project website. `https://github.com/stanford-futuredata/noscope`.

[51] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. NoScope: Optimizing deep CNN-based queries over video streams at scale. *PVLDB*, 2017.

[52] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. F. Li. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[54] P. L. Lehman and S. B. Yao. Efficient locking for concurrent operations on B-trees. *ACM Trans. Database Syst.*, 1981.

[55] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *TOMCCAP*, 2006.

[56] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *CVPR*, 2015.

[57] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen. Medical image classification with convolutional neural network. In *ICARCV*, 2014.

[58] R. Lienhart and J. Maydt. An extended set of Haar-like features for rapid object detection. In *ICIP*, 2002.

[59] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[60] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014.

[61] W. Lin, H. Fan, Z. Qian, J. Xu, S. Yang, J. Zhou, and L. Zhou. StreamScope: Continuous reliable distributed processing of big data streams. In *NSDI*, 2016.

[62] Y. Mao, E. Kohler, and R. T. Morris. Cache craftiness for fast multicore key-value storage. In *EuroSys*, 2012.

[63] A. Mhalla, H. Maâmatou, T. Chateau, S. Gazzah, and N. E. B. Amara. Faster R-CNN scene specialization with a sequential monte-carlo framework. In *DICTA*.

[64] Microsoft. The microsoft Cognitive Toolkit. `https://www.microsoft.com/en-us/cognitive-toolkit/`.

[65] L. O'Callaghan, N. Mishra, A. Meyerson, and S. Guha. Streaming-data algorithms for high-quality clustering. In *ICDE*, 2002.

[66] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman. Aggregation and degradation in JetStream: Streaming analytics in the wide area. In *NSDI*, 2014.

[67] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *CVPR Workshops*, 2014.

[68] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[69] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[70] W. Ren, S. Singh, M. Singh, and Y. S. Zhu. State-of-the-art on spatio-temporal information-based video retrieval. *Pattern Recognition*, 2009.

[71] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. FitNets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014.

[72] H. R. Roth, L. Lu, J. Liu, J. Yao, A. Seff, K. M. Cherry, L. Kim, and R. M. Summers. Improving computer-aided detection using convolutional neural networks and random view aggregation. *IEEE Trans. Med. Imaging*, 2016.

[73] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 2015.

[74] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[75] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *CVPR*, 2017.

[76] H. T. Shen, B. C. Ooi, and X. Zhou. Towards effective indexing for very large video sequence database. In *SIGMOD*, 2005.

[77] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[78] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[79] C. Snoek, K. E. A. van de Sande, O. de Rooij, B. Huurnink, E. Gavves, D. Odijk, M. de Rijke, T. Gevers, M. Worring, D. Koelma, and A. W. M. Smeulders. The mediamill TRECVID 2010 semantic video search engine. In *TRECVID 2010 workshop participants notebook papers*, 2010.

[80] C. Snoek and M. Worring. Multimodal video indexing: A review of the state-of-the-art. *Multimedia Tools Appl.*, 2005.

[81] C. G. M. Snoek and M. Worring. Concept-based video retrieval. *Foundations and Trends in Information Retrieval*, 2009.

[82] S. Stober, D. J. Cameron, and J. A. Grahn. Using convolutional neural networks to recognize rhythm stimuli from electroencephalography recordings. In *NIPS*, 2014.

[83] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, 2013.

[84] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[85] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[86] N. Tatbul, U. Çetintemel, and S. B. Zdonik. Staying FIT: efficient load shedding techniques for distributed stream processing. In *VLDB*, 2007.

[87] Y. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: A control-based approach. In *VLDB*, 2006.

[88] P. A. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[89] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan. Accelerating read mapping with FastHASH. *BMC Genomics*, 2013.

[90] Z. Xu, Y. Yang, and A. G. Hauptmann. A discriminative CNN video representation for event detection. In *CVPR*, 2015.

[91] Z. E. Xu, M. J. Kusner, K. Q. Weinberger, and M. Chen. Cost-sensitive tree of classifiers. In *ICML*, 2013.

[92] Q. Yang, C. X. Ling, X. Chai, and R. Pan. Test-cost sensitive classification on data with missing values. *IEEE Trans. Knowl. Data Eng.*, 2006.

[93] D. Yuan and P. Mitra. Lindex: a lattice-based index for graph databases. *VLDB J.*, 2013.

[94] J. Yuan, H. Wang, L. Xiao, W. Zheng, J. Li, F. Lin, and B. Zhang. A formal study of shot boundary detection. *IEEE Trans. Circuits Syst. Video Techn.*, 2007.

[95] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *SOSP*, 2013.

[96] H. Zhang, G. Ananthanarayanan, P. Bodík, M. Philipose, P. Bahl, and M. J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, 2017.

[97] X. Zhou, X. Zhou, L. Chen, and A. Bouguettaya. Efficient subsequence matching over large video databases. *VLDB J.*, 2012.