Towards edge-caching for image recognition

Utsav Drolia*, Katherine Guo†, Jiaqi Tan*, Rajeev Gandhi*, Priya Narasimhan*
*Carnegie Mellon University, †Bell Labs, Nokia
udrolia@andrew.cmu.edu, kguo@bell-labs.com, jiaqit@andrew.cmu.edu, rgandhi@ece.cmu.edu, priya@cs.cmu.edu

***Abstract*—With the available sensors on mobile devices and their improved CPU and storage capability, users expect their devices to recognize the surrounding environment and to provide relevant information and/or content automatically and immediately. For such classes of real-time applications, user perception of performance is key. To enable a truly seamless experience for the user, responses to requests need to be provided with minimal user-perceived latency.**

Current state-of-the-art systems for these applications require offloading requests and data to the cloud. This paper proposes an approach to allow users' devices and their onboard applications to leverage resources closer to home, i.e., resources at the edge of the network. We propose to use edge-servers as specialized caches for image-recognition applications. We develop a detailed formula for the expected latency for such a cache that incorporates the effects of recognition algorithms' computation time and accuracy. We show that, counter-intuitively, large cache sizes can lead to higher latencies. To the best of our knowledge, this is the first work that models edge-servers as caches for compute-intensive recognition applications.

I. Introduction

With the large number of sensors available on mobile devices and their improved CPU and storage capability, users expect their devices and applications to be smarter, i.e., recognize the surrounding environment and to provide relevant information and/or content automatically and immediately [17], [7]. Users need a more seamless way of fetching information, and are moving away from typing text into devices. Vision-based applications help users augment their understanding of the physical world by querying it through the camera(s) on their mobile devices. Applications such as face recognition, augmented reality, place recognition [6], object recognition, vision-based indoor localization, all fall under this category. There are a number of mobile applications available which do one or more of these [3], [1], [5]. Specialized augmented reality hardware is also available to the public now, such as Google's Glass [2] and Microsoft's Hololens [4], to augment their vision of their surroundings.

This class of applications has a real-time nature - user perception of performance is of primary importance; users will find delays unacceptable and will cease using the application when that happens. We want to enable a truly seamless experience for the users by providing instantaneous responses to user's requests, with minimal user-perceived latency. Moreover, with devices like Glass and Hololens, users will not make explicit requests - objects will have to be recognized in the stream of images captured while the user is looking at something. This further makes the case for real-time performance of vision-based recognition.

Currently, such services and applications rely on cloud computing. These services are compute- and data- intensive since image data needs to be processed to make sense of it, e.g. a captured image has to be recognized as a specific person. Cloud computing enables running these intensive services on powerful servers and allows mobile devices to use them remotely. Devices capture the data and upload it to the remote servers, where the data is processed, and then the results are returned to the devices. This network-based interaction already induces latency and the ever-increasing number of mobile devices and other Internet-capable things will further aggravate this model of computing. Such large numbers of devices, especially in high densities, can cause network congestion in the Internet backbone, and will inundate remote servers with a deluge of data. This will increase user-perceived latency as user-requests fight for network and compute resources. The centralized model of cloud computing cannot satisfy the low-latency requirements of such applications for the growing number of users.

One proposed approach towards meeting this challenge has been to place more compute resources at the edge of the network, e.g. fog computing [9] and cloudlets [23], both suggest placing and utilizing compute resources at the edge, near the user, alongside last-mile network elements. It is proposed that computation can be offloaded from users' devices to these edge-resources instead of offloading to the cloud. This would possibly reduce the expected end-to-end latency for applications.

Our work proposes an alternative in which we use an edge-server as a "cache with compute resources". Using a caching model gives us an established framework and parameters to reason about the performance of the system. This model makes the edge-server transparent to the client, and self-managed - administrators do not need to place content on it manually. The key insight lies in the fact that an edge-server serves a limited physical area, dictated by the network element it is attached to, and thus experiences spatiotemporal locality in requests and data from users. It has already been established that document retrieval systems over the Internet, i.e. the World Wide Web (WWW), can reap extensive benefits with caches at the edge [10] by leveraging locality in requests. Recognition applications are similar to such retrieval systems and our proposed cache for recognition can accelerate applications transparently by leveraging locality as well. In fact, we believe, recognition applications will benefit even more from caching, since many users will be making requests related to the same physical objects in the area. The challenge is that, unlike requests

in the WWW that use identifiers that deterministically map to specific content being requested, e.g. Uniform Resource Identifier (URI), requests in recognition applications are not deterministic. The object in the image needs to be recognized before the related content can be fetched. This is a compute-intensive task. This property has direct implications on how an edge-cache for recognition is designed.

In this work we develop a detailed model for expected latency for a cloud-backed recognition cache. We incorporate the effects of computation time and recognition accuracy into the model. The intention is that a detailed model can help in understanding tradeoffs and be used for prediction purposes in the future, to dynamically adjust the cache. As a first step towards that goal, we show the impact of changing the cache size on image-recognition applications. To the best of our knowledge, this is the first work that models edge-servers as caches for compute-intensive recognition applications and shows how offline analysis can help decide important model parameters. We do not propose any new computer vision or image-recognition algorithms themselves. Instead we show how we can take existing algorithms and applications, and use edge-resources effectively to reduce their user-perceived latency. Also, our approach does not involve the users' mobile devices. Instead we address the interaction between edge-servers and backend servers (a.k.a the cloud).

In the next section we provide background on mobile image-recognition and edge-computing. In Section III we model the edge-server as an image-recognition cache. We then discuss future work in Section IV. Section V presents related work, and we conclude in Section VI.

II. Background

image-recognition.pdf

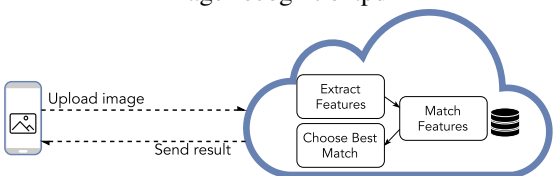


Fig. 1: Example architecture for mobile image-recognition applications

A. Mobile Image Recognition

To understand the latencies involved in recognition applications, one needs to understand how these image-recognition algorithms work.

Figure 1 shows a typical image-recognition architecture. A basic overview is as follows.

Feature extraction. First a set of features are extracted from the request image. One can think of this collection of features as a signature of the image. Some notable features used in image recognition are SIFT [20], SURF [8] and ORB [22].

Feature matching. Next, these extracted features are compared to a database of features of training images of all possible objects that can be queried. This database is collated offline. It can contain features from more than one image of each object. The more objects one wants to recognize, the bigger this database will be. The matching procedure boils down to finding the nearest-neighbor feature in the database for each feature of the input image. To accelerate this process, approximate nearest-neighbor searches have been proposed, such as Locality Sensitive Hashing [13] and kd-trees [24].

Best match selection. Once the closest features are found, they are analyzed to predict the object contained in the request image. The object that has the most feature-matches with the input image, is chosen. This can be followed by geometric verification for confirmation.

This high level procedure, of extraction, matching with an object-features set and verification, is common across vision-based recognition algorithms [25].

These stages for mobile image-recognition applications are typically carried out in the cloud. As shown in Figure 1, the image is captured by the mobile device, uploaded over a wireless network and sent across the Internet to the cloud. The image-recognition procedures are then carried out in the cloud, and the response is returned to the device. These responses are typically of some form of information or content.

In this architecture, overall latency consists of two main factors: (a) Network latency, and (b) Compute time.

(a) *Network Latency:* Applications incur this latency since they need to upload significant amounts of data to the cloud, over the Internet, for each recognition request. This latency can be reduced if the distance between the computing entity, currently the cloud, and the mobile device is reduced.

(b) *Compute Time:* Image recognition algorithms are compute intensive. The main contributor to latency is the matching of request-image features against the object-features set. Moreover, the size of this set has a direct impact on the latency. For a large number of objects, the computation time can lead to high latency and hence a poor user experience.

B. Web Caching

Caching at the edge is a widely used technique for reducing user-perceived latency when retrieving information from across the World Wide Web. This is not compute-intensive - users' requests contain the specific identifier for the content they want to view, e.g. URI, and if the cache contains the indicated content, it is returned to the user, else the request is forwarded to the backend servers. In such systems, the cache has to maximize the amount of relevant content it can serve directly, and minimize the number of requests forwarded to the backend. This is how it can minimize the expected latency for users, reduce network load and backend load.

C. Edge Computing

Computing on servers at the edge of the network has been recently proposed [9], [23].Since it is not feasible for all the increasing number of devices and "things" to communicate with cloud-based backends [9], edge computing proposes

that devices can offload compute-intensive tasks to compute resources placed at the edge of the network. Instead of just placing content and data near the user, edge-computing opens up the edge-resources to also provide compute-resources at the edge.

In this work, we show how we can take the idea of caching at the edge, leverage the compute-resources now available at the edge, and create an image-recognition (IR) cache.

III. Edge-cache for Image Recognition

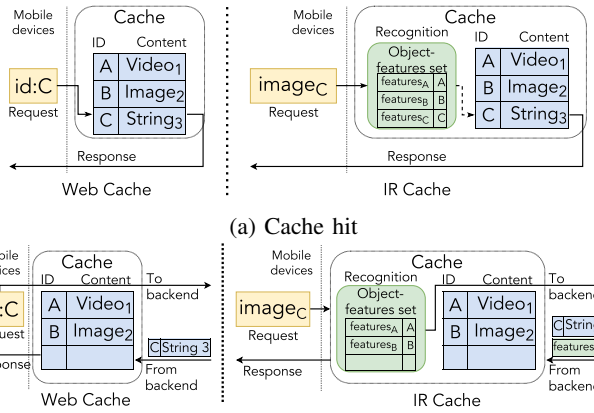
We model edge-resources and their interaction with the cloud using principles from caching systems, instead of the computation-offloading model proposed in earlier systems. A caching model provides a framework for analyzing the system, along with knobs to tune the cache to minimize different kinds of costs. Caching systems are inherently hierarchical and hence fit the edge-cache and cloud interaction. They are also extendable - more caches can be added in the hierarchy.

The primary metric for mobile image recognition is the user-perceived latency. In this paper, we are concerned about the latency in the wired part of the network, i.e. between the wireless networking infrastructure, to which the edge-cache is attached, and the cloud. The latency from the edge-cache's perspective is the duration between the reception of a request at the cache and the transmission of the response from the cache. The formula for reasoning about this latency is given by the Average Memory Access Time (AMAT) formula: $H + m * M$, where H = Hit latency, M = Miss latency and m = Miss ratio. However, for an IR cache at the edge of the network, backed by the cloud, each component of this formula needs to be broken down. In this section we present our formulation for the expected latency of an IR cache. We believe this is the first work to present such a formulation for edge-caching for image-recognition.

A. Overall Operation

We believe that an IR cache should be (1) transparent to the user, and (2) it should populate itself autonomously, similar to a web cache. However, these two caches are not interchangeable. Figure 2 presents the overall operation of an IR cache alongside a web cache.

An image-recognition cache receives images, which themselves do not deterministically map to the content or information that needs to be returned to the user. The object within the image needs to be recognized using the known features set, and then the object's identifier is used to return related content, if locally present, unlike a typical web cache, whose requests are deterministic. This is illustrated in Figure 2(a). On a miss (Figure 2(b)), the web cache forwards the request to the backend, receives the response, stores it locally and sends the response to the user. The image-recognition cache does the same, but along with the response, the backend also sends the features needed to recognize this request in the future. This is inserted into the features set.



(a) Cache hit

(b) Cache miss

Fig. 2: Request look-up in a web cache and in an image-recognition cache. The incoming request for the web cache has a specific I.D. (*C*), while for the IR cache, the request is an image containing the object *C*. The IR cache needs to first use recognition to know that the image contains *C* in it and then look-up *C* in the cache.

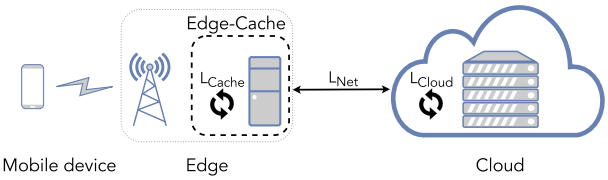


Fig. 3: Location of an edge-cache, backed by the cloud.

B. Expected Cache Latency

In order to understand how this difference impacts latency, we can model the expected latency for an edge-cache (web or recognition), by extending the AMAT formula. For an edge-cache, this is given by:

$$E[L] = L_{Cache} + m * (L_{Net} + L_{Cloud}) \quad (1)$$

Where L_{Cache} = cache lookup latency, i.e. time taken by the cache to match a request to the related response, L_{Net} = edge-server-to-cloud network latency, i.e. time taken for a request to reach the cloud after being issued from the edge-server, L_{Cloud} = cloud lookup latency, i.e. time taken by the cloud to match a request to the related response, and m = Miss ratio. This is depicted in Figure 3.

Lookup is trivial in web caches and hence latency (L_{Cache}) is small. The size of the cache has negligible impact on this latency for web-caches. This is not the case in an IR cache. In the IR cache, lookup involves the image-recognition phase. Firstly, recognition is compute-intensive, and lookup time is significant. Secondly, number of objects being recognized, which is the cache size here, impacts the latency. Thus, for an IR cache, $L_{Cache} = f(k)$, $f(k)$ = function of cache size, k. Replacing in Equation (1),

$$E[L] = f(k) + m * (L_{Net} + L_{Cloud}) \quad (2)$$

To investigate further, we evaluated an IR edge-cache with a Least Frequently Used (LFU) cache-eviction policy, for increasing cache sizes, deployed as shown in Figure 3. Figure 4 shows the results for that experiment. In this experiment, the cloud is a resourceful server (12 cores, 24 Hyper-Threads, 32GB RAM) while the edge-server is a powerful PC (4 cores, 8 Hyper-Threads, 8GB RAM). The request distribution is a Zipf distribution with $\alpha = 0.8$. We chose the Zipf distribution since user requests over the WWW tend to have a Zipf distribution [10]. The network conditions between the edge and cloud is fixed at 5 Mpbs and 40ms RTT. The cloud contains features for all the objects that can be present in images (400 objects). The cache is warmed to be full before the measurements are taken. This setup is relatively conservative, especially with respect to the distribution. We expect that actual request distributions will be even more skewed, and thus lead to even more gains due to the cache.

Initially, as the cache size (k) increases, the hit ratio increases, i.e. the cache serves more items locally and avoids sending requests to the cloud, and the overall latency decreases. Although the cloud is computationally powerful, it needs to classify incoming images as one of 400 known objects. The cache, on the other hand, knows of only those k objects whose features it has cached. Hence it tries to classify incoming images as one of k objects, where $k < 400$. This lowered computational load, combined with the effects of the network conditions, favors having a cache.

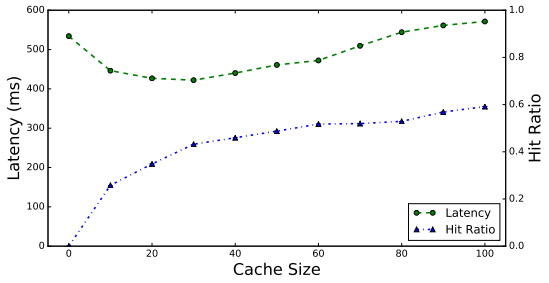


Fig. 4: Overall latency and hit ratio for LFU-based IR edge-cache, over different cache sizes.

However, this is only true up to a point. For cache sizes over 30, the latency starts to increase. In fact, for cache sizes 80 and above, the latency is worse than when there is no cache at all (cache size is 0). This is counter-intuitive - for a typical cache, a bigger size implies lower latency. For an IR-cache, however, there are other factors in play. The size at which the inflection occurs is affected by multiple factors - the nature of $f(k)$, the request distribution, the network and cloud conditions (L_{Net} and L_{Cloud}). These factors are inter-dependent, e.g. even if $f(k)$ is very high, a cache can still lower latency if the request distribution is skewed and L_{Net} and L_{Cloud} are high. Thus a simple, static LFU cache cannot be used for image recognition applications, since it does not incorporate the effects of all these factors.

Next, we discuss the nature of $f(k)$ and we deconstruct the miss ratio (m) further to incorporate the effect of recognition accuracy into the expected latency, a factor that does not affect typical caches.

C. Understanding Expected Latency for IR Cache

The formulation in Equation (2) is dominated by two key components, (1) the effect of cache size on latency, $f(k)$, and (2) miss ratio, m. In this section we discuss the effects of these parameters to construct a detailed formulation.

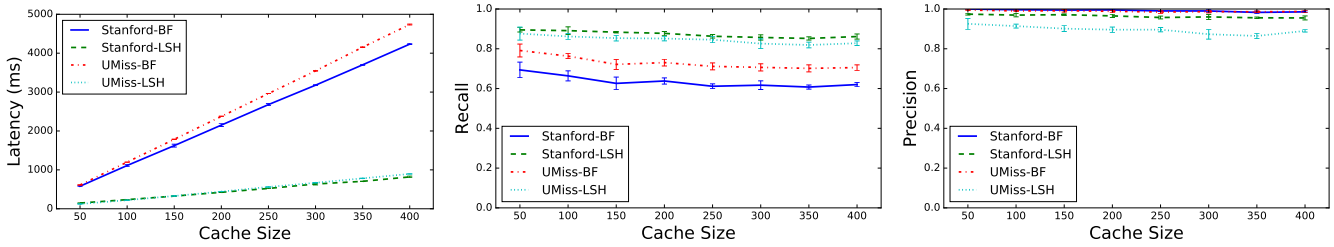
1) Effect of cache size on lookup latency: The number of objects represented in the trained, local feature set in the edge-server, i.e. the cache size, has a direct impact on the lookup latency. This relationship is captured by $f(k)$. This relationship depends on the type of feature, the number of features extracted per object image and the recognition/search algorithm utilized to look for the input features in the object-features set. For recognition algorithms $f(k)$ is a monotonically increasing function - as the number of recognizable objects increases, the time taken to process one input increases. Figure 5(a) depicts $f(k)$ for object recognition using two algorithms over two different datasets. Figure 6 shows the training and query images from the datasets. Each query in the experiment goes through the aforementioned image recognition pipeline, running on the PC. We see that for both datasets and both algorithms, the computation time increases as the number of objects, i.e. the cache size, increases. This is the reason we saw the inflection point in Figure 4. For larger cache sizes, $f(k)$ was too high and led to an overall increase in latency. This implies that simply providing a large cache size to an IR-cache is not enough. It should only be done if needed, i.e. the cache size should be dynamic and controllable.

2) Miss ratio: Typically the miss ratio, m, depends on the cache size, the cache replacement policy and the underlying request distribution. In an IR cache, along with these factors, accuracy of the recognition algorithms also impacts misses - when the cache fails to respond to a request even though the requested object is in the cache. This needs to be captured in the formulation.

Now, $m = 1 - P(hit)$, where $P(hit) = P(recognized \cap cached)$ is the probability of a cache hit, i.e. a query being recognized successfully when the corresponding object is in the cache. Then, $P(recognized \cap cached) = P(recognized|cached) * P(cached)$.

$P(cached)$ is the probability that a randomly chosen object is in the cache and depends on the cache size, the cache replacement policy and the underlying request distribution.

$P(recognized|cached)$ is the probability that a randomly selected query will be recognized given that the queried object is in the cache. This is also known as *recall*. It is the ratio of number of correct responses to number of queries whose corresponding object is in the trained model/dataset being queried. This incorporates the accuracy of the recognition algorithm into the miss ratio and is dependent on the number of objects in the cache. Given this dependence, we denote this as $recall(k)$. Figure 5(b) shows how $recall(k)$ varies for different number of objects (k) for object recognition using two algorithms over two different datasets, while keeping the



(a) Effect on latency (b) Effect on recall (c) Effect on precision

Fig. 5: Effect of varying the number of objects in the cache, i.e. the cache size, for object recognition using two datasets (Stanford [11] and UMiss [26]) and two matching algorithms (Brute-force and Locality Sensitive Hashing [13].

Fig. 6: **Top:** Training images from the dataset. **Bottom:** Corresponding query images in dataset, taken by mobile devices.

precision of the recognition close to 1.0 (Figure 5(c)). By keeping precision high, we ensure that the cache will rather categorize a request as "unknown" and forward it to the cloud than return an incorrect result. As the number of objects in the cache increases, the accuracy tends to drop, i.e. it is more difficult to precisely recognize the correct object when there are many options to choose from.

Putting this formulation back in Equation (2),

$$E[L] = f(k) + (1 - recall(k) * P(cached)) * (L_{Net} + L_{Cloud}) \quad (3)$$

This formulation presents a detailed model of the expected latency in a cloud-backed IR cache that incorporates the effects of the compute intensive recognition algorithms in terms of computation time and accuracy. This model presents an alternative perspective to look at edge-servers, along with the computation-offloading model.

IV. Future Work

We pointed out earlier that typical caching policies cannot be used for an IR cache due to the effects of the recognition algorithms. We also showed through Figure 4 that simply using the largest cache size is not useful. The cache size will be dependent on multiple factors and hence will need to be adjusted in a dynamic manner. Now that a detailed model for the latency in an IR cache has been developed, it can be used to find the correct cache size. By finding the k that minimizes the formulation presented in Equation (3), we can find that cache size. To be able to do this, the other parameters of the formulation will need to be estimated. In future work, we will develop methods to estimate each component and find their relationship with the cache size, k, such that k is the only unknown variable in the formulation and can be set through the minimization. We expect that estimating $P(cached)$ will be challenging but will also capture the underlying spatiotemporal locality. We expect that the minimization will have to be carried out at regular intervals to incorporate changes in $P(cached)$, L_{Net}, and L_{Cloud}. By doing so, we will be able to develop a dynamic system that can react to changes, predict what the optimal cache size should be and adjust accordingly such that overall latency is minimized.

V. Related Work

Web caching. Our work has drawn inspiration from web-caching and CDN systems. These systems cache content at the edge of the network and serve requests from it to avoid going to a centralized server. [10] showed that requests have spatiotemporal locality, and this is why caching works well. People tend to query for similar web pages and this popularity has a Zipf-like distribution. We claim that recognition-based applications present even higher spatial and temporal localities, and caching can drastically reduce their latency. However, our approach to minimize latency is different compared to other caching systems. CDN systems try to optimize for different objectives, such as byte-hit-rate [21], i.e. maximize the hit rate and the number of bytes served per hit. We are proposing to minimize latency too, but by taking into account the added compute time and inaccuracy due to the recognition algorithms.

Edge-computing. The dilemma of relying on cloud computing for perception and recognition applications has been highlighted in literature [7], [23], [9]. [14] showcases why offloading these applications to the cloud can increase perceived latency. They present "cloudlets", an edge-server that is connected the wireless infrastructure. It makes the case for moving compute resources close to the user to avoid network latency. However, in such an offloading approach, how does the cloudlet know the set of recognizable objects? In [14], the feature set is placed explicitly on the cloudlet. We believe that the cloudlet should be able to do this autonomously, and our caching approach will enable that. [9] presents a generalized computing paradigm called *fog computing*. It proposes to diffuse the concept of cloud computing across the network infrastructure between the cloud and the user, thus bringing

computational resources closer to the user and making the network smarter.

We have used this idea of "computing at the edge" proposed by these authors, and proposed a new way to utilize these resources at the edge for image recognition applications.

Computer vision. The rise in the ubiquity of mobile devices has led to numerous optimizations for computer vision and image recognition algorithms to enable efficient processing locally on mobile devices. Feature extractors and descriptors such as SURF [8] have performed decently in mobile settings, but the new binary features such as ORB [22] have made it possible to do extraction in near real-time on the devices. However, local recognition does not scale beyond tens of images [16], [12], and we want to achieve recognition of thousands of objects. Recent thrusts in deep learning and convolutional neural networks have achieved high recognition accuracy [19], and it has been optimized for mobile devices as well [18], but again the device alone cannot achieve the diverse recognition that is desired [15].

Given that mobile devices alone cannot support recognition across more than a few categories, external resources and data from the cloud will be necessary. Our IR cache will make that interaction seamless and ensure low-latency.

VI. Conclusion

Users expect to use mobile recognition applications by querying their environment using their mobile devices and expect immediate responses. To live up to these expectations, applications choose to offload intensive tasks to the cloud. However, this adds network latency and reduces overall responsiveness. Edge-computing proposes to meet this challenge by placing compute-resources at the edge of the network to avoid going to the cloud for all tasks, thus reducing latency. The state-of-the-art model to use edge-resources is similar to how applications use the cloud, i.e. offload intensive tasks to the edge-servers. In this paper, we propose a new model to use edge-resources. Instead of offloading, we propose to use them as recognition caches, backed by the cloud. A caching model provides a framework to reason about the performance of the system and also makes the edge transparent to the application itself. We develop a model for expected latency in an image-recognition cache and show how to incorporate the effects of compute-intensive recognition algorithms. We show how the cache size effects the expected latency through experiments with different algorithms and datasets. We believe that this is the first work that models edge-servers as image-recognition caches, and provides a formulation for expected latency that incorporates the effects of recognition algorithms in a caching system.

References

[1] Bing Vision. https://en.wikipedia.org/wiki/Bing_Vision.

[2] Google Glass. https://en.wikipedia.org/wiki/Google_Glass.

[3] Google Goggls. https://en.wikipedia.org/wiki/Google_Goggles.

[4] Microsoft Hololens. https://www.microsoft.com/microsoft-hololens/en-us.

[5] Nokia Point & Find. https://en.wikipedia.org/wiki/Nokia_Point_%26_Find.

[6] Wikitude. http://www.wikitude.com/.

[7] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan. Advancing the state of mobile cloud computing. In *ACM Workshop on Mobile Cloud Computing and Services*, 2012.

[8] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 126–134. IEEE, 1999.

[11] V. R. Chandrasekhar, D. M. Chen, S. S. Tsai, N.-M. Cheung, H. Chen, G. Takacs, Y. Reznik, R. Vedantham, R. Grzeszczuk, J. Bach, et al. The stanford mobile visual search data set. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 117–122. ACM, 2011.

[12] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168. ACM, 2015.

[13] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[14] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan. The impact of mobile multimedia applications on data center consolidation. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 166–176. IEEE, 2013.

[15] 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 *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[16] P. Jain, J. Manweiler, and R. Roy Choudhury. Overlay: Practical mobile augmented reality. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 331–344, New York, NY, USA, 2015. ACM.

[17] W. Kelly. Computer vision and the future of mobile devices. http://www.techrepublic.com/article/computer-vision-and-the-future-of-mobile-devices/, August 2014.

[18] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–12. IEEE, 2016.

[19] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[20] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[21] B. M. Maggs and R. K. Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[22] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.

[23] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, Oct 2009.

[24] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[25] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science &; Business Media, 2010.

[26] X. Wang, M. Yang, T. Cour, S. Zhu, K. Yu, and T. X. Han. Contextual weighting for vocabulary tree based image retrieval. In *2011 International Conference on Computer Vision*, pages 209–216. IEEE, 2011.