PicSys: Energy-Efficient Fast Image Search on Distributed Mobile Networks

Noor Felemban, Fidan Mehmeti, Hana Khamfroush, Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta, *Fellow, IEEE*

Abstract—Mobile devices collect a large amount of visual data that are useful for many applications. Searching for an object of interest over a network of mobile devices can aid human analysts in a variety of situations. However, processing the information on these devices is a challenge owing to the high computational complexity of the state-of-the-art computer vision algorithms that primarily rely on Convolutional Neural Networks (CNNs). Thus, this paper builds PicSys, a system that enables answering visual search queries on a mobile network. The objective of the system is to minimize the maximum completion time over all devices while taking into account the energy consumption of mobile devices as well. First, PicSys carefully divides the computation into multiple filtering stages, such that only a small percentage of images need to run the entire CNN pipeline. Splitting such CNN computation into multiple stages requires understanding the intermediate CNN features and systematically trading off accuracy for the computation speed. Second, PicSys determines where to run each of the stages of the multi-stage pipeline to fully utilize the available resources. Finally, through extensive experimentation, system implementation, and simulation, we show that PicSys performance is close to optimal and significantly outperforms other standard algorithms.

Index Terms—Distributed systems, deep learning, crowdsourcing, energy

1 Introduction

The wide spread use of camera-enabled mobile devices has led to the generation of large amounts of images and videos that are rich sources of information for critical applications like surveillance and monitoring [1], [2]. The computer vision community has made headway in developing algorithms to extract information from visual data. Deep Neural Networks (DNN), specifically Multi-layer CNNs, have been shown to achieve high accuracy in visual analytics tasks such as image classification, detection, segmentation and activity detection [3], [4], [5]. However, this comes at a high computational complexity – often these algorithms push the limits of what can be achieved on current generation of mobile devices.

In this paper, we focus on the specific problem of searching for objects of interest on a distributed network of mobile devices and a server in the cloud, in response to a query, which is issued in a centralized fashion. Images that contain the object of interest are gathered in a central server.[1] Responding to such queries requires: (i) identifying all mobile devices that are likely to contain the images of interest, (ii) sending queries, (iii) searching for the objects within the images taken by these devices and finally, (iv) uploading the images of interest to the central server. In this paper, we propose PicSys to address steps (iii) and (iv).[2]

PicSys consists of a number of mobile devices that have stored images, and a more powerful server machine, which we call the *cloud*, to which the mobile devices communicate over a wireless link. When a query is issued, it is expected that the issuer (analyst) will receive all the positively identified images that match the query description. The mobile device makes decisions as to where the image processing should take place, on the device or the cloud, in a distributed fashion. The goal of the system is to process all images as fast as possible, including uploading all images that contain the object of interest to the central server. The structure of PicSys and the decision algorithms used by the mobile devices meet this goal while respecting energy constraints of the mobile devices. The energy constraints are put in place by the users to control how much of their energy they are willing to expend for this crowd-processing application.

To help mitigate the low processing power and highly constrained energy resources of mobile devices, we introduce a processing pipeline on the mobile device which includes what we call an *accelerated* CNN. The accelerated CNN is a simple CNN that is faster, requires less energy, and is less accurate than a more complex CNN. It is used by the mobile device as a filter to remove images that likely do not have the object of interest from the processing pipeline. This reduces the burden on the mobile device of either offloading images for processing, or processing the full complex CNN locally.

1. The public has shown an increasing willingness to share images and videos with law enforcement in circumstances such as terrorist attacks and missing people. We assume voluntary participation, with mobile devices taking appropriate privacy measures [6].

2. In our system, we consider only processing images from mobile devices, i.e., those images that are not already stored in the cloud.

- *N. Felemban, F. Mehmeti, and T. La Porta are with The Pennsylvania State University, University Park, PA 16802 USA. E-mail: {nmf154, fzm82, tlp}@cse.psu.edu.*
- *H. Khamfroush is with University of Kentucky, Lexington, KY 40506 USA. E-mail: khamfroush@cs.uky.edu.*
- *Z. Lu is with the Department of Computer Science, Peking University, Haidian 100871, China. E-mail: zongqing.lu@pku.edu.cn.*
- *S. Rallapalli is with the IBM T. J. Watson Center, Yorktown Heights, NY 10598 USA. E-mail: srallapalli@us.ibm.com.*
- *K. Chan is with the Army Research Laboratory, Adelphi, MD 20783 USA. E-mail: kevin.s.chan.civ@mail.mil.*

Manuscript received 7 May 2019; revised 15 Oct. 2019; accepted 21 Dec. 2019. Date of publication 31 Dec. 2019; date of current version 4 Mar. 2021. (Corresponding author: Noor Felemban.)
Digital Object Identifier no. 10.1109/TMC.2019.2963150

The main contributions of our work are:

- We design the PicSys system which has the structure, via its multi-stage CNN pipeline, to support the design of algorithms that minimize query completion times with energy constraints.
- We perform extensive experimentation to determine processing times, performance and energy consumption for various CNNs and pre-processing tasks to select the best set to meet our objectives.
- We formulate and solve an optimization problem to decide where to run each of the stages of the computation, so that we utilize the available computational resources optimally. We do this for two cases: with and without energy constraints on mobile devices.
- We design heuristics to solve the problem. The heuristic has one component that is not energy sensitive, which is used as a building block for the energy-sensitive heuristic.
- We present a detailed implementation of our system in a real testbed and illustrate experimental and simulation results for both a small case network as well as on large networks.

After implementing our system and using the task-level measurements to run a large scale simulation, the obtained results show that PicSys outperforms existing schemes, and that using the filtering stage in the mobile devices yields a 35 percent in time savings as opposed to when no-filtering stage is available.

The remaining part of the paper is organized as follows. Section 2 provides an overview of image processing. Section 3 gives an overview of the model. In Section 4 we present experimental results that help in designing and evaluating the system performance. The optimization and algorithms are presented in Section 5, followed by the performance evaluation relying on both simulation and experimental results in Section 6. Section 7 reviews some related work. Finally, the paper is concluded in Section 8.

2 Overview of Image Processing

In this section we present some background on image processing as it is required to understand this paper.

Object classification refers to recognizing the objects appearing within images. Specifically, object classification algorithms return a probability distribution of certain objects being contained within an image. The higher the probability for an object, the more likely that object is in the image. The objects that may be searched for in an image are predefined by training the object classifier to recognize features of the objects of interest.

There are two main performance metrics considered with object classification: *recall* and *false positives*. Recall measures the number of images containing an object of interest that are returned considering the ground truth of total number of images that contain the object. Formally, it is expressed as $R = \frac{TP}{FN+TP}$, where TP and FN are true positives and false negatives of the object classification, respectively. False positives are the number of images that are positively classified that in fact do not contain the object of interest. The rate at which true positives are returned is also called the hit-rate.

We refer to the CNN as a classifier. It consists of a set of layers, starting with an input layer and ending with an output layer. Layers vary depending on the CNN architecture; some common layers are convolutions, max pooling, dropout, data augmentation, ReLU activations, etc. We mainly use GoogLeNet [7] and Squeezenet [8] in our experimentation. We refer the interested reader to [7] and [8] for more details about each layer in those architectures.

Because the classifier only returns a probability that an object is within an image, a threshold k is often defined which is used to label whether an image contains an object, or not. If the probability that an object is present in the image is kth highest or better out of all the objects detected, the image is labeled as containing the object. By setting a small value of k, only images that contain the object with high certainty will be labeled. This may miss some images that contain the object and will result in a lower recall and a lower false positive. By setting a large value of k, the results will be more inclusive and will lead to a higher recall, but also to a higher false positive.

State-of-the-art classification algorithms are based on deep CNNs, such as Alexnet [3], GoogLeNet [7], VGGNet [9] and ResNet [10]. Many CNNs, designed for classification, end with a Softmax layer. The Softmax layer assigns a probability for each class the CNN was trained on (with total sum being equal to 1). That probability is the likelihood of an image containing an object that belongs to that particular class. We use the term top-k to refer to the top k predictions that have the highest probabilities. In general, the best results are achieved when the full CNN is processed, but tradeoffs can be made to allow lower classification performance with lower processing times by processing fewer layers of the CNN.

3 PicSys Overview

In PicSys, operation starts when mobile devices receive a query from a centralized source requesting images containing an object of interest. This can be done in a similar way to subscription alert services.

Images can be processed on mobile devices or the cloud. The cloud has high computational power and a persistent power supply, and as such enables the use of more accurate models that complete processing in a shorter time period. Mobile devices are constrained by their computational power and battery capacity. PicSys uses a *multi-stage decision technique* to reduce the overall processing time while meeting a specific recall requirement and energy constraints.

Fig. 1 illustrates the mobile device's three-stage decision process. In the first stage, the *Initial Stage*, the mobile device makes a decision to either (i) process the image locally, or (ii) offload it for processing on the cloud. That decision is based on the network conditions, energy state of the mobile device, cloud backlog and the hit-rate estimate. The hit-rate estimate is continuously updated based on recent historical data. If the hit-rate is very high, it may be better to offload the picture to the cloud for processing as, ultimately, images of interest need to be uploaded anyway. This is not only faster, but more energy efficient as there is no processing cost on the mobile device. However, note that *even if the hit-rate is high, not all images will be uploaded as that may overwhelm the cloud which serves multiple mobile devices simultaneously*. If images are

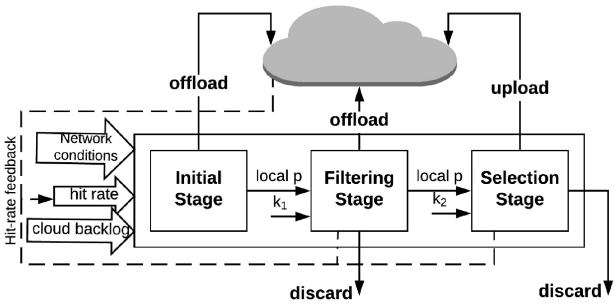


Fig. 1. Overview of the system on the mobile device.

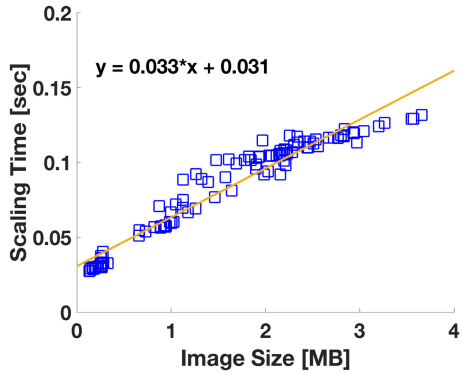


Fig. 2. Scaling time using FFmpeg.

uploaded to the cloud, the remainder of the processing takes place there.

If local processing is chosen in the first stage, then the *Filtering Stage* is executed on the mobile device. The Filtering Stage involves a computationally efficient, but less accurate algorithm, which we refer to as the *accelerated* CNN. It extracts coarse-grained features from the images, based on which a decision is made as to if the image likely contains the object of interest. depending on whether the top-k_1 requirement is met or not. If not, the image is discarded. If so, the mobile device decides to (i) complete processing locally by executing the *Selection Stage*, or (ii) offload the image to the cloud for further accurate processing.

In the Selection Stage on the mobile device, a more complex and accurate CNN is executed which achieves more accurate results albeit at a higher energy cost. After processing, one of two decisions is made: (i) upload the image to the requester if the image is likely to contain the object, or (ii) discard the image.

From an energy perspective, devices with high hit rates should offload their images because it saves the energy of performing the object classification processing on the device when images will eventually be uploaded anyway. For devices with low hit rates, the decision is more complex. If the accelerated CNN is less expensive from an energy standpoint than offloading the image over the wireless interface, local processing will be preferable. Whether this may be the case or not depends on the wireless technology and the quality of the wireless signal.

Further, the system enables the interaction between mobile devices and the cloud. In order to avoid increased communication complexity, there is no interaction between the mobile devices.

Most crowdsourcing applications, such as law enforcement or intelligence gathering require a person to inspect a photo, and virtually all military applications require a person to verify an image, so a full high quality image is needed. We choose to upload the image with its original size for the requester to have access to it in case further processing or analysis needs to be performed. When offloading the image to the cloud for processing we also choose to send the original sized image to avoid the need to re-transmit the original image if the object of interest ends up being present. Offloading the scaled version of the image and then re-transmitting it would result in a different problem setup, and is beyond the scope of this work.

It should be mentioned that PicSys tunes the parameters of the different stages of the pipeline based on the recall requirement of the application. In PicSys we have to pick two k values - namely k_1 and k_2 (as explained in Section 4.3) for Filtering and Selection Stages, respectively.

4 Experiments and Design Decisions

This section elaborates the PicSys design choices backed by extensive experimentation. We use the experimental results to drive the simulations. In the following experiments we use *Caffe* [11], a deep learning framework, to train and test our CNNs. The *gpu flag* is used to run the models on NVIDIA GTX TITAN X 12 GB, modeling the cloud. For implementation on mobile devices we use the *caffe-android-lib* [12] on Samsung Galaxy S9. The two labeled data sets we use for classification are ImageNet [13], ILSVRC2012, with 1000 classes, and VOC2012 [14], with 20 classes.

To get the processing times, we used a small set of 100 images, captured from a Galaxy S9. For the ImageNet data set we used the trained models from Model Zoo [15]. For VOC2012, we fine-tune the CNN to classify the 20 classes.

4.1 Image Pre-Processing

To use Caffe, images must be scaled to the particular dimensions required by the CNN. This scaling may be done internally by Caffe, or by pre-processing. We find that pre-processing the image for scaling using FFmpeg [16] instead of Caffe has two benefits: (i) the scaling time is a linear function of image size as shown in Fig. 2, and the processing time after scaling is constant, as shown in Fig. 3; (ii) more importantly, scaling using FFmpeg and then processing the image is faster than pre-scaling on Caffe, as shown in Fig. 3.

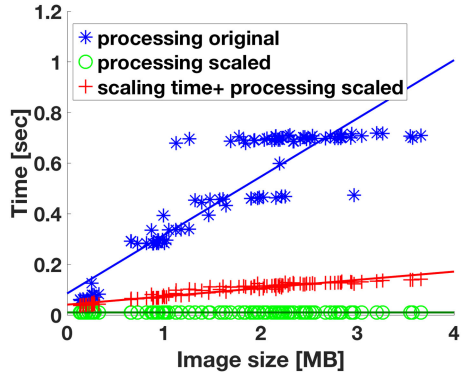


Fig. 3. Processing time for images.

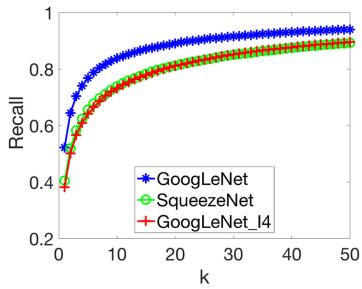


Fig. 4. Recall-ILSVRC2012.

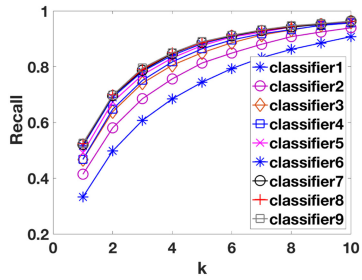


Fig. 5. Recall-per classifier.

These realizations help make the offloading decisions in Section 5, and serve as the argument to choose pre-processing the images using FFmpeg.

4.2 Choosing the Deep Learning Models and Model Partitioning

Three factors need to be considered when choosing the deep learning models: recall, processing time and energy consumption. The requirement for the accelerated CNN is to have a faster processing time with lower energy expenditure than the expensive CNN. In this section we study the three factors in more detail as we explore our two choices for the accelerated CNN: a truncated version of the expensive CNN, i.e., terminate the processing after only a subset of layers are complete, or a separate, simpler full CNN such as SqueezeNet.

In the first set of experiments, we compare the recall of GoogLeNet, SqueezeNet and a compressed version of GoogLeNet. We use the publicly available Caffe models of GoogLeNet and SqeezeNet trained on ILSVRC2012 [15]. We leverage the original design of GoogLeNet and use the existing Softmax layer available in an early stage after a layer called *inception_4a*; we use this subset of the network and call it 'GoogLeNet_i4'.

Fig. 4 shows the recall versus top-k, where top-k is the highest k probabilities (classes) from the classifier, as explained in Section 2. The results are obtained using 25k images from the ILSVRC2012 validation set. We see that running the complete GoogLeNet model yields higher recall than running a subset of it or a more compressed model, such as SqueezeNet. We also observe that choosing either SqueezeNet or GoogLeNet_i4 will give approximately the same recall.

Note that with the recent development in compressed CNN models and the large amount of research on deploying these models on mobile devices [17], [18], [19] , the accelerated model and the expensive model can be exchanged with other models as the user sees fit. If using a deep learning framework that enables dynamic computation on CNNs, choosing the accelerated CNN as a subset of the expensive CNN is beneficial, and enables the download of a single model on the device.

In the second set of experiments we show more details on partitioning complex CNNs. The work in [20] shows processing times for full CNNs as well as for individual layers. For our work, in addition to the processing time for each layer, we require the addition of a classifier after the layer where partitioning takes place. To get more insights, we fine-tuned GoogLeNet on VOC2012 by adding various classifiers after inception layers. We add *Pooling*, *Dropout*, *Classifier*, and *Softmax* layers after each inception *"Concat"* layer, summing up to nine classifiers.

Fig. 5 shows the recall after each classifier. The processing time on the mobile device increases from 0.3 to 0.4 sec as the number of layers increases. The higher the number of layers used in the forward pass, the higher the recall. However, the increase is not linear and the difference gets smaller the further into the network we process. The choice of the classifier is based on a tradeoff between accuracy and computation cost.

In another set of comparisons, we measure the energy cost for processing images using the three CNNs: Squeezenet, GoogLeNet and the accelerated CNN at Classifier-2. The measurements were taken by profiling battery usage using Battery Historian [21]. We represent the energy as the percentage of battery consumed per image. We find that Classifier-2 CNN and Squeezenet have almost the same energy cost, which is 64 percent lower than that of the complete GoogLeNet. In Section 6 we use Squeezenet as the accelerated CNN, and GoogLeNet as the expensive CNN.

4.3 Choosing the k_1 and k_2 Values

The two processing phases may be filtered using different k values. k_1 is used for the accelerated CNN and k_2 is used for the expensive CNN. The expensive CNN only receives and processes images that pass the accelerated CNN's top-k_1 requirement. Thus, the loss in recall from the Filtering Stage cannot be regained regardless of how high k_2 is set. However, setting the value of k_1 too high may result in unnecessary uploads or expensive CNN processing for the mobile device because the accelerated CNN will not filter out some of the images that do not contain the object. Fig. 6 shows the recall on VOC2012 with 20 classes. We see that a higher recall is reached as the values of k_1 and k_2 increase. Using Fig. 6, we pick k_1 and k_2 values based on the recall requirement of the application. In Section 6 we choose $k_1 = 5$ and $k_2 = 3$.

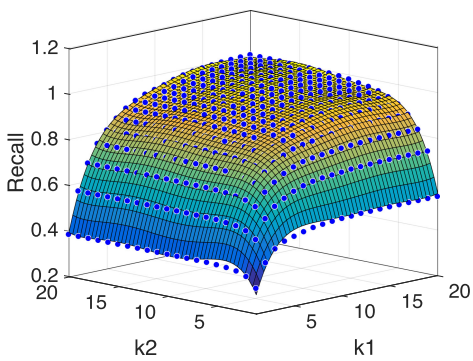


Fig. 6. Aggregate recall on VOC2012.

5 PicSys Optimization

In this section we present formal optimization problems for allocating resources in PicSys, and heuristics for algorithms that come close to achieving optimal performance. Ultimately, the goal is to process all images and have those with objects of interest uploaded to the cloud as fast as possible while respecting the energy constraints imposed by users of devices. The energy constraints are the budget for how much energy each user is willing to spend on the image processing application.

We first formulate the problem when there are no constraints on the energy consumption and show that the problem is a nonlinear optimization. We then propose heuristics to solve the problem, which result into near optimal running times. We then formulate the problem where the energy consumption of mobile devices is taken into account, which is followed by the corresponding heuristics.

5.1 Mathematical Formulation: No-Energy Constraints

We assume that initially every image is stored in one of the devices in the system. Devices process images in series, i.e., they perform scaling and local processing consecutively. However, scaling and local processing can take place while other images are being transmitted or are waiting in the transmission queue. In the following, we use M to represent the set of mobile devices in the system ($m \in M$), and I to denote the set of images in the system, i.e., $I = \{I_1, I_2, \ldots, I_{|M|}\}$.

We assume that the transmission rate r_m, and the top-k values are given, and that k_1 and k_2 are set to meet the recall requirement. We also assume that the mobile devices may be heterogeneous, and after an image is scaled, the processing time for that image on any mobile device is $c^p_{i,m}$, whereas the processing time on the cloud is c^p_u.

Let q be the queried class; we define two binary variables: $z^{k_1}_{i,m}$ and $z^{k_2}_{i,m}$. $z^{k_1}_{i,m}$ is 1 if q is in the top-k_1 classes after accelerated processing, and 0 otherwise. Similarly, $z^{k_2}_{i,m}$ is 1 if q is in the top-k_2 classes after expensive processing, and 0 otherwise. Table 1 summarizes the notation used throughout this paper.

5.1.1 Components of Delay

In our problem formulation, we are accounting for the following delay components:

Scaling delay, $c^s_{i,m}$, is the time required for the input image i on device m to be scaled to the input size of the CNN.

Processing delay, $c^p_{i,m}$, is the sum of processing times for image i that take place on device m. Since we have two stages where local processing can take place, namely, accelerated CNN and expensive CNN, we need to account for both delays when considering processing delay. Therefore, $c^p_{i,m} = a\alpha_m + b\beta_m$, where $a \in \{0,1\}$, $b \in \{0,1\}$, whereas α_m is the processing time for the accelerated CNN on device m, and β_m is the processing time for the expensive CNN on the same device (m).

Note. An image that is uploaded to the cloud can only start processing after it has been completely uploaded, and when a server is free. Both the cloud and mobile devices can start processing while sending or receiving images.

TABLE 1
Nomenclature and Notation

$c^s_{i,m}$	scaling time of image i on device m
c^p_u	processing delay of an image on the cloud
$c^p_{i,m}$	processing delay of image i on device m
$c^t_{i,m}$	transmission delay of image i on device m
α_m	processing time for accelerated CNN on device m
β_m	processing time for expensive CNN on device m
r_m	transmission rate for device m
$B_{i,m}$	size of image i on device m
$z^{k_1}_{i,m}$	truth value of image i, stored on device m, after accelerated processing
$z^{k_2}_{i,m}$	truth value of image i, stored on device m, after expensive processing
$cost1_{i,m}$	cost of direct offload
$cost2_{i,m}$	cost of accelerated processing followed by offload
$cost3_{i,m}$	cost of accelerated processing followed by discard
$cost4_{i,m}$	cost of accelerated processing, then expensive processing followed by upload
$cost5_{i,m}$	cost of accelerated processing, then expensive processing followed by a discard
T_{max}	completion time for all images (makespan)
$t^{completion}_{i,m}$	completion time of image i that is on device m
$t^{start}_{i,m}$	start time of image i on device m
$x1_{i,m}, \ldots, x5_{i,m}$	decision variables
$l^t_{i,m}$	transmission queueing delay of image i on device m
$l^q_{i,m}$	cloud queueing delay for image i from device m
a_m	% of battery device m dedicates to crowdsourcing
$E_{i,m}$	energy consumed for image i on device m (in %)
E^r_m	residual energy in the budget of device m (in %)
$e^s_{i,m}$	scaling energy of image i on device m (in %)
$e^t_{i,m}$	transmission energy of image i on device m (in %)
e_{α_m}	processing energy for accelerated CNN of device m (in %)
e_{β_m}	processing energy for expensive CNN of device m (in %)

Transmission delay, $c^t_{i,m}$, is the time for the image to complete transmission. For simplicity, we assume that each mobile device has a dedicated channel, hence devices are able to transmit simultaneously. This assumption is realistic in many situations where participants are geographically dispersed or access the network via Wifi, which is likely in cases in which a large number of images would be uploaded. The study of the impact of congestion on the wireless links is left for future work. The cloud receives images from many sources, as multiple devices may be uploading images concurrently. However, a single mobile device transmits images sequentially. Thus, the transmission delay of image i from device m to the cloud is calculated as $c^t_{i,m} = \frac{B_{i,m}}{r_m}$, where r_m and $B_{i,m}$ represent the transmission rate and image size, respectively.

Transmission queueing delay, $l^t_{i,m}$, is the time image i spends on device's m transmission queue after it is scheduled to be offloaded/uploaded until it starts transmission.

Cloud queueing delay, $l^q_{i,m}$, is the time the offloaded image i (originating from device m) spends in the cloud queue from the moment it arrives until a server is available to process it. Note that both $l^t_{i,m}$ and $l^q_{i,m}$ depend on the decisions that

have been made at previous time slots, and on the number of images in the queue.

5.1.2 Formulation

The goal is to minimize the system completion time given a minimum recall requirement. A system is said to have completed its tasks when all the images in it have been processed, and all images that are positively classified by the CNNs reach the cloud. Therefore, *the completion time of the system is the maximum of completion times of all the images in the system*. Thus, we have the following problem:

$$\begin{aligned} &\text{minimize } T_{max} \\ &\text{subject to:} \\ &T_{max} \geq t_{i,m}^{completion}, \qquad \forall i \in I, \forall m \in M, \end{aligned} \tag{1}$$

$$x2_{i,m} + x3_{i,m} + x4_{i,m} + x5_{i,m} \leq 1, \qquad \forall i \in I, \forall m \in M, \tag{2}$$

$$\sum_{j=2}^{5} xj_{i,m} - x1_{i,m} = 0, \qquad \forall i \in I, \forall m \in M, \tag{3}$$

$$x2_{i,m} - z_{i,m}^{k_1} x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{4}$$

$$x3_{i,m} - (1 - z_{i,m}^{k_1}) x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{5}$$

$$x4_{i,m} - z_{i,m}^{k_2} x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{6}$$

$$x5_{i,m} - z_{i,m}^{k_1}(1 - z_{i,m}^{k_2}) x1_{i,m} \leq 0, \qquad \forall i \in I, \forall m \in M, \tag{7}$$

$$xj_{i,m} \in \{0, 1\}, \qquad \forall j \in \{1, 2, 3, 4, 5\}, \forall i \in I, \forall m \in M, \tag{8}$$

where $t_{i,m}^{completion}$ is the completion time of image i that is on device m, and is a function of the decision variables. It can be calculated as $t_{i,m}^{completion} = t_{i,m}^{start} + cost_{i,m}$. In this equation, $t_{i,m}^{start}$ and $cost_{i,m}$ are start time for image i on device m and the total cost for processing image i, respectively. We have

$$t_{i,m}^{start} = t_{i-1,m}^{start} + x1_{i-1,m}(c_{i-1,m}^{s} + \alpha_m) + x4_{i-1,m}\beta_m + x5_{i-1,m}\beta_m, \tag{9}$$

and

$$\begin{aligned} cost_{i,m} = (1 - x1_{i,m}) cost1_{i,m} + x2_{i,m} cost2_{i,m} \\ + x3_{i,m} cost3_{i,m} + x4_{i,m} cost4_{i,m} + x5_{i,m} cost5_{i,m}. \end{aligned} \tag{10}$$

The binary decision variables $x1_{i,m}$, $x2_{i,m}$, $x3_{i,m}$, $x4_{i,m}$, $x5_{i,m}$ are defined as follows. $x1_{i,m}$ is equal to 0 if image i on device m is offloaded in the first stage, and 1 otherwise. $x2_{i,m}$ is equal to 1 if image i on device m is passing through accelerated CNN and then offloaded to the cloud, and 0 otherwise. $x3_{i,m}$ is 1 if image i on device m passes accelerated CNN and is discarded, and is 0 otherwise. $x4_{i,m}$ is 1 if image i on device m passes expensive (after it has passed through accelerated) CNN, and is uploaded to the cloud afterwards, and 0 otherwise. $x5_{i,m}$ is 1 if image i on device m is discarded after passing expensive CNN, and 0 otherwise.

Note. To reduce the number of equations required to convey the constraints of the system, the decision variable $x1_{i,m}$ is defined differently when compared to $x2_{i,m}, \ldots, x5_{i,m}$. The cost of each decision is defined as:

$$\begin{aligned} cost1_{i,m} &= l_{i,m}^{t} + c_{i,m}^{t} + l_{i,m}^{q} + c_{u}^{p}, \\ cost2_{i,m} &= c_{i,m}^{s} + \alpha_m + l_{i,m}^{t} + c_{i,m}^{t} + l_{i,m}^{q} + c_{u}^{p}, \\ cost3_{i,m} &= c_{i,m}^{s} + \alpha_m, \\ cost4_{i,m} &= c_{i,m}^{s} + \alpha_m + \beta_m + l_{i,m}^{t} + c_{i,m}^{t}, \\ cost5_{i,m} &= c_{i,m}^{s} + \alpha_m + \beta_m. \end{aligned}$$

Constraint (1) shows that the completion time of the system is greater or equal to the completion time of every image in every device (be it processed on the device or the cloud). Constraint (2) ensures that only one of the decision variables $x2_{i,m}, \ldots, x5_{i,m}$ could have value 1 at a time. Constraint (3) ensures that if an image is not offloaded in the first stage, it will be processed locally, and if it is offloaded in the first stage, there will be no local processing. Constraints (4), (5), (6), and (7) represent the requirements for accelerated CNN and offload, accelerated CNN and discard, expensive CNN and upload, and expensive CNN and discard, respectively. Constraint (8) shows that the decision variables are binary.

As can be seen, the problem formulation is representing a nonlinear optimization problem, since the objective is a function of costs multiplied by decision variables $x1_{i,m}, \ldots, x5_{i,m}$, where the costs are also functions of $l_{i,m}^{t}$ and $l_{i,m}^{q}$. If we simplify the problem by assuming a powerful cloud with infinite servers (no cloud queueing delay), and the transmission times for each device are constant, still the transmission queueing delay remains, introducing thus non-linearity. Relaxing this delay component is not possible because it yields to the unrealistic assumption of infinite bandwidth. Therefore, the transmission queueing delay is non-avoidable and is the cause to the non-linearity of the problem. As a result, the problem cannot be formulated as an integer linear program, which could be solved by the CPLEX optimizer. In the following section, we propose a heuristic to solve our scheduling problem with no energy constraints.

*NP-Hardness of the Problem.*We prove the NP-hardness of our problem by reducing the well-known NP-hard job-shop scheduling problem [22] to our problem. In the original version of job-shop scheduling problem, we have n jobs $j_1, j_2, \ldots, j_n$ of varying processing times, which need to be scheduled on M machines with varying processing power while trying to minimize the makespan (the total length of the schedule or total time taken to complete all jobs). We will show next that the job-shop scheduling problem is a special case of our problem.

Considering the special case where the communication delay of images is zero and the truth values of processing times on every machine are known *a priori*, the data collection scheduling problem can be seen as a generalization of job-shop scheduling with the following assumptions. The set of all possible images to be processed represents the set of jobs, and each decision variable $x1_{i,m}, x2_{i,m}, \ldots, x5_{i,m}$ in our problem represents one machine in the job-shop scheduling problem. Therefore, the total of $5 * I * M$ machines exist in the job-shop scheduling problem given these assumptions, as for each image $i \in I$ there exist 5 different ways of processing it on a machine, and each way could be seen as a separate machine in the context of the job-shop scheduling problem. Also, assuming that the constraints on decision variables correlation in our problem are relaxed,

then the problem defined in Eqs. (1), (2), (3), (4), (5), (6), (7), and (8) can be seen as a simple job-shop scheduling problem, and thus, our problem is more general than the standard job-shop scheduling problem. Thus, data collection scheduling problem is NP-hard.

5.2 Proposed Heuristic: No-Energy Constraints

Given that the optimization problem presented earlier is NP-hard, we resort to heuristic algorithms in order to be able to find a solution to this problem.

As mentioned in Section 3, PicSys consists of multiple stages. Initially, a decision is made to either offload the image immediately to the cloud, or go through the Filtering Stage, where accelerated processing takes place. At the end of this stage, the device decides either to offload the image to the cloud or to complete the processing locally. If the latter is the case, the image undergoes the Selection Stage with expensive processing. There are two possible outcomes from the last stage: (i) upload the image to the cloud, with no further processing needed there, or (ii) discard the image.

5.2.1 Estimating the Cloud Completion Time

First, the device sends a *request packet* to the cloud and in response gets the cloud queue size, q_c. The device has knowledge of the number of images in its local transmission queue, q_t, and can accordingly calculate image i's transmission queueing delay $l^t_{i,m}$. Based on these values, the device estimates the completion time of the head-of-the-queue image if it is offloaded to the cloud. This estimate is given by:

$$e_c = l^t_{i,m} + c^t_{i,m} + (q_t + q_c + 1) * c^p_u. \tag{11}$$

Note that the term q_c is the source of uncertainty in e_c. Its value can change from the moment the device sends the request packet to the moment when the image arrives at the head of the transmission queue, e.g., other images from other mobile devices have arrived in the meantime and/or the cloud has already processed the images that have been there.

5.2.2 Estimating the Device Completion Time

The device estimates its completion time based on both its hit-rate and the number of images remaining in its decision queue. A device is said to have completed its processing when the last image in the decision queue is processed or when the last positive image is offloaded/uploaded.

Let the number of images in the device's queue be q_m, and let the hit-rate of the queried object in the given device be h_m. Assuming k_1 and k_2 are chosen such that the difference in recall between the two stages is not high, the estimated completion time of the device m when $c^t_m > c^s_m + \alpha_m + \beta_m$ can be calculated as follows:

$$e_m = (1 - h_m) * q_m * (c^s_m + \alpha_m) + c^s_m + \alpha_m + \beta_m + h_m * q_m * c^t_m. \tag{12}$$

If for the average transmission time of images it holds that $c^t_m \leq c^s_m + \alpha_m + \beta_m$, then

$$e_m = (1 - h_m) * q_m * (c^s_m + \alpha_m) + h_m * q_m * (c^s_m + \alpha_m + \beta_m) + c^t_m. \tag{13}$$

In both aforementioned estimates, we account for the worst case where the hits appear at the end of the local queue. Since in Eq. (12) the transmission delay is the largest delay component, the term $c^s_m + \alpha_m + \beta_m$ is not multiplied by the number of images with hits, but appears only once (corresponding to the first image with a hit). Likewise, in Eq. (13), c^t_m is not multiplied by the number of images with hits because the largest delay component is $c^s_m + \alpha_m + \beta_m$.

When processing everything on the mobile device, the worst case scenario is when the decision, for every image, occurs only after the second stage. There are two extreme cases corresponding to this scenario. In the first one, the transmission time c^t_m is much higher than $c^s_m + \alpha_m + \beta_m$, in which case the completion time is approximately $h_m * q_m * c^t_m$, similarly to Eq. (13). In the second extreme case, the values of $c^s_m + \alpha_m + \beta_m$ are much higher than c^t_m, and the completion time would have as the dominant component the term $q_m * (c^s_m + \alpha_m + \beta_m)$, similarly to Eq. (12).

For all the more common non-extreme cases corresponding to this scenario, we propose the following approximation for the completion time:

$$e_m = q_m * (c^s_m + \alpha_m + \beta_m) + w_m * h_m * q_m * c^t_m, \tag{14}$$

where w_m is a "correction factor", whose value is a function of c^t_m, c^s_m, α_m, and β_m. In almost all cases the value of w_m is 0 or very small. Only when $c^t_m >> c^s_m + \alpha_m + \beta_m$, the value of w_m is 1.

Note. In Eqs. (13) and (14), we drop the i index for $c^s_{i,m}$, as it represents the *average* scaling delay of an image on that device. Similarly, c^t_m represents the *average* transmission time of an image from device m to the cloud.

5.2.3 Heuristic

We define the heuristic as follows:

- For each image in a device's local queue, the Initial stage and the Filtering stage perform the following:
  1) The device calculates the estimated completion time of the cloud e_c, based on the value of q_c it receives, using Eq. (11).
  2) The device calculates its local estimated completion time e_m using Eq. (12) or Eq. (13).
  3) The device calculates the speedup ratio $\eta = \frac{e_c}{e_m}$. If $\eta < 1$ the image is offloaded to the cloud, else it is processed locally.
  4) Every image that reaches the Selection stage and is labeled as a hit is uploaded to the cloud.

The logic for each stage is described in Algorithms 1 and 2. The complete heuristic is outlined in Algorithm 3. In the remainder of the paper we refer to this as the *time-heuristic*.

Algorithm 1. INITIAL_STAGE_DECISION ()

```
1 calculate η ;
2 if η < 1 then
3    offload image;
4 else
5    accelerated-local processing;
6:   FILTERING_ STAGE_DECISION ();
7 end
```

Algorithm 2. FILTERING_ STAGE_DECISION()

```
1  if class of interest in top-k_1 then
2    calculate η ;
3    if η < 1 then
4      offload image;
5    else
6      expensive-local processing;
7      SELECTION_ STAGE_DECISION ();
       // if the class of interest in top-k_2 then
       upload image, else discard it
8    end
9  else
10   discard image;
11 end
```

Algorithm 3. Decision Process on Mobile Device

```
1  while there are images in the decision queue do
2    if l^t_{i,m} ≥ c^s_{i,m} + α_m then
3      accelerated-local processing;
4      if class of interest in top-k_1 then
5        if l^t_{i,m} ≥ β_m then
6          expensive-local processing;
7          if class of interest in top-k_2 then
8            upload to cloud;
9          else
10           discard image;
11         end
12       else
13         FILTERING_STAGE_DECISION ();
14       end
15     else
16       discard image;
17     end
18   else
19     INITIAL_STAGE_DECISION ();
20   end
21 end
```

Note. Hit-rate is used to estimate the completion times, and the completion times are used to make the decision on the steps to follow in the system by each mobile user. When it comes to using the hit-rate feedback, how often the system sends a feedback to a user depends on how stable that prediction is. In this paper, we use an average for the hit-rate.

5.3 Mathematical Formulation: Energy Constraints

In Section 5.1 we have considered the problem of minimizing the completion time where the mobile devices are willing to help without any conditioning on energy consumption. However, there may be users who would be willing to help but would like to limit their battery consumption for that purpose. In this section we consider the problem where every user has a finite amount of energy they are willing to allocate for crowd-sourcing. To this end, we adapt the previous optimization problem, but adding the energy consumption constraint.

Let a_m denote the percentage of battery the mobile device m would be willing to spend on image detection. The percentage of energy consumed for image i on device m is denoted as $E_{i,m}$. The new optimization problem becomes

$$\begin{aligned} &\text{minimize } T_{max} \\ &\text{subject to: } (1),(2),(3),(4),(5),(6),(7),(8), \\ &\sum_{i \in I_m} E_{i,m} \leq a_m, \qquad \forall m \in M. \end{aligned} \tag{15}$$

In Eq. (15), for $E_{i,m}$ we have

$$\begin{aligned} E_{i,m} = (1 - x1_{i,m}) cost1^{(e)}_{i,m} + x2_{i,m} cost2^{(e)}_{i,m} + x3_{i,m} cost3^{(e)}_{i,m} \\ + x4_{i,m} cost4^{(e)}_{i,m} + x5_{i,m} cost5^{(e)}_{i,m}. \end{aligned} \tag{16}$$

Note that Eq. (16) is the energy counterpart of Eq. (10), and because of that $x1_{i,m}, \ldots, x5_{i,m}$ have the same meaning as in the first optimization problem, whereas $cost1^{(e)}_{i,m}, \ldots, cost5^{(e)}_{i,m}$ represent the cost in terms of energy, corresponding to the same stages as (time) costs $cost1_{i,m}, \ldots, cost5_{i,m}$ in Eq. (10), and are given as:

$$\begin{aligned} cost1^{(e)}_{i,m} &= e^t_{i,m}, \\ cost2^{(e)}_{i,m} &= e^s_{i,m} + e_{\alpha_m} + e^t_{i,m}, \\ cost3^{(e)}_{i,m} &= e^s_{i,m} + e_{\alpha_m}, \\ cost4^{(e)}_{i,m} &= e^s_{i,m} + e_{\alpha_m} + e_{\beta_m} + e^t_{i,m}, \\ cost5^{(e)}_{i,m} &= e^s_{i,m} + e_{\alpha_m} + e_{\beta_m}, \end{aligned}$$

where $e^s_{i,m}$ is the energy cost (in %) of scaling image i on device m, e_{α_m} is the energy cost (in %) of the accelerated CNN for device m, e_{β_m} is the energy cost (in %) of the expensive CNN for device m, whereas $e^t_{i,m}$ (again in %) is the cost of uploading image i from device m.

Since the optimization problem with the energy constraints is essentially the extended version of the first optimization problem, and we have proved that the latter is NP-hard, adding a new constraint to it does not change the NP-hardness. Hence, the new optimization problem is also NP-hard. As a result, similarly to the first case, we will resort to using heuristics for this case as well.

5.4 Proposed Heuristic: Energy Constraints

The overall goal of the energy-heuristic is the same as the time-heuristic, but the energy-heuristic tries to ensure that the largest amount of images on a device will be processed before the battery budget is exhausted. At specific intervals the device probes for remaining energy budget. Based on the outcome, the device determines if it needs to choose low-energy processing options, or can be more aggressive and choose processing options that are of higher rate.

At each battery probe the heuristic works as follows:

- The mobile device checks to see if it has enough remaining energy in its budget to process all unprocessed images locally; this is the worst case energy processing scenario. If so, the device can run the time-heuristic without concern for energy because it has sufficient energy to complete the task regardless of the processing decisions.
- If there is not enough energy to process all images locally, in lines $2-8$ of Algorithm 4, the device checks

to determine if there is enough energy to offload all unprocessed images. If not, the device may not be able to process all images before exhausting its budget. In this case it checks to see the energy required to offload all remaining images, or execute the accelerated CNN only on all images, and chooses the cheapest option which will likely result in the most images being processed. This will depend on the type of wireless interface being used.

- If the device does have enough energy to offload all the images, the algorithm is more aggressive. The device first determines how many images can be fully processed locally using its remaining energy, which is the worst case energy scenario, and if it can process at least one, it sets the probe window to this size and runs the time-heuristic. If it does not have enough energy to process a single image locally, it determines the number of images it can process using only the accelerated CNN. If it can process at least one, it sets this value to the next probe window and runs the time-heuristic. If there is not enough energy to perform the accelerated CNN on a single image, then all images are offloaded.

Algorithm 4 shows the logic at each energy probe. Similar to Section 5.2, we drop the index i for the costs, as it represents the *average* values. In the remainder of the paper we call this the *energy-heuristic*.[3]

Algorithm 4. ENERGY_ PROBE()

```
1  if E_m^r < num_imgs *(e_m^s + e_{α_m} + e_{β_m}) then
2    if E_m^r < num_imgs *e_m^t then
3      if num_imgs*e_m^t < num_imgs*(e_m^s + e_{α_m})+num_hits*e_m^t
       then
4        ALL OFFLOAD;
5      else
6        ALL ACCELERATED ;
7      end
8      next_probe=window_size;
9    else
10     if E_m^r / (e_m^s+e_{α_m}+e_{β_m}) > 1 then
11       next_probe=max(E_m^r / (e_m^s+e_{α_m}+e_{β_m}),window_size);
12       TIME_HEURISTIC();
13     else
14:      if E_m^r / (e_m^s+e_{α_m}) > 1 then
15:        next_probe=max(E_m^r / (e_m^s+e_{α_m}+e_{β_m}),window_size);
16:        restrict local processing to accelerated only;
17:        TIME_HEURISTIC()
18       else
19:        ALL OFFLOAD;
20       end
21     end
22   end
23 else
24:   TIME_HEURISTIC();
25 end
```

5.5 Design Considerations

We consider next the design related issues of energy probing and the number of cloud tokens.

5.5.1 Energy Probing

In our heuristic the algorithm probes the device for the remaining energy. Since battery percentages are represented in integer values, we choose our probing window accordingly. We estimate the amount of images a device can locally process using 1 percent energy and set that to be the window size:

$$window\_size = \frac{1}{e_m^s + e_{\alpha_m} + e_{\beta_m} + e_m^t}. \tag{17}$$

5.5.2 Cloud Tokens

When evaluating the performance of PicSys it becomes apparent that at the start of a query every device views the cloud as being unloaded, i.e., has a snapshot of a small q_c, and starts by offloading too many images. This leads to overloading the server, especially in cases where the number of devices is large, and the number of images on each device is small.[4] To prevent this, at the beginning of the process we calculate a number of tokens and assign them to each device to control their upload rate. The device will only make the decision to offload if there is an available token, otherwise it will do local processing. If $|I_m|$ is the total number of images in device m, the number of tokens is calculated as

$$num\_tokens = \frac{max_{m \in M}|I_m|\alpha_m}{c_u^p}. \tag{18}$$

The usefulness of tokens in practice is elucidated later in Section 6.2.3.

5.6 Complexity

The running time of the algorithm on each device increases linearly with the number of images, i.e., it is $O(n)$, where n is the number of images on the device. The reason for this is that in every step of Algorithms 1, 2, and 3 only constant number of operations are performed. The energy-aware algorithm has the same complexity, i.e., $O(n)$, as it follows the time-heuristic with additional energy comparisons and restrictions, all of which induce constant number of operations in each step.

6 Experimental Evaluation

We evaluate our algorithm using both experiments and simulations. Due to the complexity, we are only able to find the optimal solution for cases with a small number of users.

In order to evaluate our heuristics, we introduce an idealized *look-ahead* algorithm, which has future knowledge of the truth values for every image as well as exact processing

3. The energy-heuristic tends to follow the decisions of the time-heuristic whenever there is enough residual enrgy for crowdsourcing in the smartphone.

4. Since distributing the tokens makes sense only for a small number of images on each device (for a large number of devices), the energy consumption on every device is negligible in these scenarios. As a result, this approach is oblivious to the residual energy left on devices for crowdsourcing.

TABLE 2
Completion Time [sec]

opt	look-ahead	heuristic	greedy	all-device	all-cloud
1.74	1.74	1.83	1.98	4.5	2.86

and scaling times. We assume the look-ahead algorithm acts as an *oracle*, which is able to correctly predict the overall outcome of all the images. Note that the performance of the *look-ahead* algorithm is not achievable in practice. Unless stated otherwise, we will be using the same CNNs as those described in Section 4. In this section, all processing times and energy costs are based on our experimental results.

6.1 Baselines

We compare our heuristic with the optimal, the look-ahead algorithm, and the following three standard algorithms:

- *All-device:* All images are processed locally on the mobile device where they are originally stored. Then, every image that is positively classified is uploaded to the cloud.
- *All-cloud:* All images are immediately offloaded to the cloud for processing.
- *Greedy:* Every mobile device makes a greedy choice of whether to process the image at the head of its queue locally, or offload it based on the device's estimated completion time, and the completion time of the image if offloaded, assuming the image does not incur any queueing delay on the cloud.

6.2 Simulation Results

In this subsection, first, the performance of the time-heuristic is compared to the optimal solution for a small system. To do this in a controlled environment, we use a simulation. Then, we present the details of the testbed implementation, which is succeeded by results on a larger system via simulation, using values obtained from our experiments. We then proceed with comparisons to the case where the system is heterogeneous. After that we compare to no-filtering stage algorithms. Finally, we show some results when energy consumption is considered.

As a final note, unless stated otherwise, we assume a system of homogeneous devices in the scenarios to follow in this section. Hence, in those cases, we will mostly skip using the index m to simplify the notation.

6.2.1 Comparison to Optimal

We start by comparing our heuristic to the optimal solution for a small system of three devices. Because this is a simulation of a small number of devices and images, we only use the time-heuristic and do not consider energy. The system parameters are as follows: $|M| = 3$, $|I| = \{10, 4, 6\}$, and $h = \{0.2, 0, 0.5\}$ for each device, respectively. The other parameters are $\bar{B}_i = 800$ kB, $r_m = 20$ Mbps, $\alpha = 0.3$ sec/img, $\beta = 0.6$ sec/img, $c_u^p = 0.01$ sec/img, and $c_i^s = 0.05$ sec/img. Assuming 0 is a miss and 1 is a hit, an example of truth values for each device are: Dev-1: {0,0,0,0,1,0,0,0,1,0}, Dev-2: {0,0,0,0}, Dev-3: {0,0,0,1,1,1}. Table 2 illustrates the completion time for all the algorithms.

TABLE 3
Decision Trace

	d#	decisions										[sec]
opt	1	i-off	i-off	i-off	i-off	i-off	f-disc	f-disc	f-disc	i-off	f-disc	1.68
	2	i-off	i-off	i-off	i-off							1.12
	3	i-off	i-off	i-off	i-off	i-off	s-up					1.68
		cloud completion time										1.74
heuristic	1	i-off	i-off	f-disc	i-off	f-off	f-disc	i-off	f-disc	i-off	f-disc	1.75
	2	i-off	i-off	f-disc	f-disc							0.7
	3	i-off	i-off	f-disc	i-off	f-off	s-up					1.83
		cloud completion time										1.75

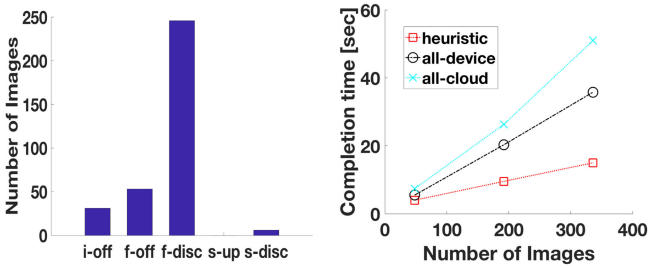
To get more insights, we look at the decisions made by our heuristic and see why it deviates from the optimal. Table 3 shows the decisions each device makes in both the optimal and heuristic. We note that the optimal can be reached using various decision sequences; we only show one of them.

The decisions in the table correspond to *stage-decision*, where *i* refers to the Initial stage, *f* is the Filtering stage, and *s* is the Selection stage. The possible decisions are to offload *-off-*, upload *-up-*, or discard *-disc-*. This notation will be used throughout the remaining of the paper.

For the heuristic, as the cloud is initially empty, and there are available tokens, all devices start by offloading. When the devices reach the third image in their local queues, the local transmission queue starts to grow, pushing devices to start processing locally. Taking a closer look at Dev-3, which is the slowest device in the system, the main cause of the delay is the choice made for image5. When the initial decision is made, the transmission cost is greater than the accelerated processing time, and the device chooses to start local processing. However, given that image5 is a hit, it will not be discarded after the Filtering stage. After that, the transmission queueing delay gets lower and image5 is offloaded to the cloud for further processing. In this scenario, image5 still incurs the transmission delay, causing an unnecessary delay in the process initiation of image6.

6.2.2 Testbed Implementation

We implemented PicSys on a testbed of three Samsung Galaxy S9 mobile devices and a server with NVIDIA GTX TITAN X. The devices are connected to the cloud through Wifi. All devices and the cloud are using Caffe models. The expensive CNN is GoogLeNet, and for the accelerated CNN we use SqueezeNet. The Caffe models are pre-installed on the devices and their size is 25 MB and 3 MB, respectively. We queried for all images containing cars. Every device has 112 images, with a total of $|I| = 336$ images in the system. The hit-rate is $h = 0.25$, and the false positive rate $FPR = 0.166$. The transmission rate is 8 Mbps. We run the all-device, all-cloud and the time-heuristic and observe that our proposed heuristic outperforms the other approaches. Namely, our heuristic is $3.4\times$ faster than all-cloud and $2.3\times$ faster than all-device. In the case of all-cloud all 336 images are initially offloaded to the server and processed there,



(a) stage distribution: 336 images (b) varying $|I|$

Fig. 7. Implementation results.

giving an average system completion time of 51.015 sec. For all-device, all images are processed locally on the device and 42 images are uploaded to the server as positive images, with an average completion time of 35.7 sec. The time-heuristic has the smallest average completion time of 14.99 sec.

The number of images in each stage, for the time-heuristic, is shown in Fig. 7a.

In the previous scenario the total number of images in the system was fixed (336). Next, we consider how the makespan of the system changes as a function of the total number of images in the system. The other parameters remain unchanged. Fig. 7b illustrates that dependency for the three possible approaches. By varying the number of images in the system we can see that, as expected, with the increase in the number of images the completion time increases as well. From Fig. 7b it is clear that our heuristic outperforms both all-cloud and all-device. All-cloud has the largest completion time. As the number of images increases, there is a significant increase in the savings provided by the time-heuristic.

It is important to note that the code used in the system implementation is similar to the code used for the experimentation results shown in Section 4.

6.2.3 Larger Systems

In this section we use the measurements from the testbed to simulate a large-scale system in a controlled environment. We first consider the time-heuristic. We generate images with various sizes following a normal distribution $N(\mu, \sigma)$. Based on the results of Section 4.1, we use the following equation to calculate the scaling time of the images: $c_i^s = 0.033 * B_i + 0.031$. The transmission rates between the mobile devices and the cloud are set using a uniform distribution in the range $[\gamma r_m, r_m]$, where $\gamma < 1$. The processing rates using accelerated processing and expensive processing are set uniformly and randomly in the range $[\gamma\alpha, \alpha]$ and $[\gamma\beta, \beta]$, respectively. The processing rate on the cloud is also subject to a uniform distribution in the range $[\gamma c_u^p, c_u^p]$.

To match our implementation in Section 6.2.2, we choose Squeezenet as the accelerated CNN, and GoogLeNet as the expensive CNN.

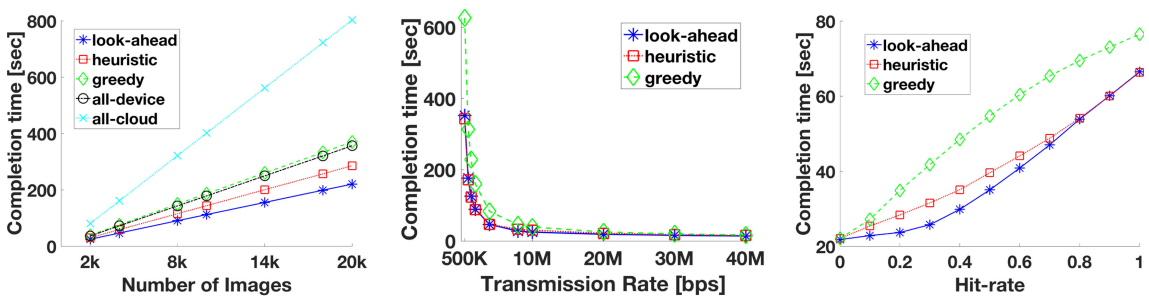
We consider one cloud. Unless stated otherwise, the system parameters for the simulation are: $|M| = 20$, $h = 0.25$, $\mu = 1$ MB, $\sigma = 800$ kB, $r_m = 10$ Mbps, $\alpha = 0.2$ sec/img, $\beta = 0.4$ sec/img, $c_u^p = 0.01$ sec/img, $\gamma = 0.6$. For a choice of $k_1 = 5$, $k_2 = 3$, the corresponding true positive rates are $TPR_\alpha = 0.81$, $TPR_\beta = 0.82$, whereas the false positive rates are $FPR_\alpha = 0.0035$ and $FPR_\beta = 0.0017$, respectively. It is worth mentioning that the processing times, FPR and TPR, are based on experimental results. The results of the simulation are obtained by averaging over 100 independent runs. In the following, we will get more insights by varying different system parameters.

Fig. 8a shows the comparison among all the algorithms. We can see that as the number of images in the system increases, the completion time increases, too. However, all-cloud has a higher growth rate caused by the larger queueing delays at the cloud. The heuristic has the slowest growth rate and is closest to the look-ahead algorithm.

All-cloud and all-device never do better than the heuristic. When the optimal solution is to perform all-cloud or all-device, the heuristic does so. For better visualization, we exclude these two algorithms from the remainder of the figures.

Interestingly, even at very high transmission rates, the heuristic still results in some images being locally processed on the device. Fig. 8b shows that even at higher transmission rates, the processing rate of the cloud is not high enough to eliminate the cloud from being a bottleneck. On the other hand, at very low transmission rates, the best solution is for the majority of the images stored on mobile devices to be locally processed, and that is what our heuristic does. Although the heuristic and look-ahead curves look the same in the low rate region, they are not. E.g., at 8 Mbps the delay for the former is 32.7 sec, whereas for the latter it is 27.5 sec.

When varying the hit-rate in the system, all-cloud remains constant as all images are initially offloaded regardless of their truth values. All-device increases linearly since there is initially a constant cost of accelerated local processing that takes place regardless of the hit-rate. As the hit-rate increases, the number of images that go to expensive processing and are uploaded to the cloud increases. Since the heuristic integrates the hit-rate into its decision making process, it reaches look-



(a) Varying $|I|$ (b) Varying r_m (c) Varying h

Fig. 8. Simulation results: Time-heuristic.

TABLE 4
Heterogeneous System

bandwidth r	processing α, β	images per stage				device completion time [sec]		
		i-off	f-off	f-disc	s	heuristic	all-device	all-cloud
high	high	19	5	66	10	27.748	35.424	80.07
high	low	31	12	56	1	35.8	63.16	80.07
low	high	4	1	76	19	37.83	42.964	160.07
low	low	15	4	68	13	51.802	63.542	160.07

ahead in both extreme cases of a hit-rate of 0 and 1, as illustrated in Fig. 8c, whereas other algorithms fail to do so.

In general, we see that our heuristic outperforms the baselines and reaches the *look-ahead* in different cases. The number of devices used for the evaluation in this section was chosen to depict clearly the performance differences. Nevertheless, we have also run simulations with larger number of devices with similar conclusions drawn. E.g., in a system with 200 devices PicSys performs better than the other algorithms, and is only 3 percent worse than the look-ahead algorithm. In general, the system is aimed for a large number of users.

Finally, to demonstrate a case of the usefulness of tokens, we use the same setting as before in this section. For 100 devices, each of which contains 4 images, the cloud calculates that there should be around 75 tokens (according to Eq. (18)). The devices with the highest bandwidth receive these tokens (since they can utilize them faster than the other devices). As the tokens limit the initial offload, we obtain a completion time of approximately 4 sec. In the absence of tokens, devices initially see an empty cloud and offload immediately, increasing the completion time on the cloud. In that case, the system completion time is about 6 sec, which shows that in this case using tokens reduces the system completion time by 33 percent.

6.2.4 Heterogeneous Systems

Next, to show that our algorithm is resilient to heterogeneity of devices and variations in parameters, we ran a simulation similar to that of Section 6.2.3 using 20 heterogeneous devices. These devices are grouped into four groups of five devices with the same characteristics. Different groups differ in processing rate and network bandwidth. Table 4 illustrates the results of our simulation. To highlight the differences, we assume a network with new devices and old devices, that are half as fast, as the clock speed for phones has almost doubled in the past years [23], [24]. We assume devices are in different locations, hence a variation in bandwidth. For illustration purposes, we use a factor of two i.e., $\alpha_{low} = 2\alpha$, $\beta_{low} = 2\beta$, $r_{low} = \frac{r_m}{2}$, and $\alpha_{high} = \alpha$, $\beta_{high} = \beta$, $r_{high} = r_m$.

Table 4 shows the average number of images processed at different stages for each device in the four different classes.Devices with higher bandwidth tend to offload more than other devices. Devices with high processing capabilities process more images locally than devices with lower processing rates and similar bandwidth. The devices with low bandwidth and low processing rates become the bottleneck, even as the heuristic reduces their completion times, compared to other algorithms. The completion time of the cloud in the system is 54.57, 64.51 and 160.17 sec for the heuristic, all-device and all-cloud, respectively.

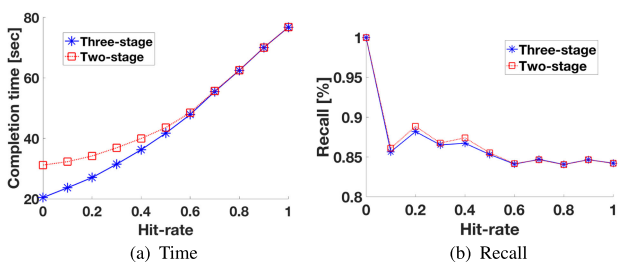
6.2.5 Comparison to no-Filtering Stage

One of our main contributions is the addition of an intermediate, accelerated-processing, stage that acts as a filtering stage. In this set of simulations, we study the difference in completion times between our three-stage model and a "normal" two stage model, where there is no Filtering stage. In the second case, a mobile device initially makes the decision based on the network conditions, cloud backlog and hit-rate, and that decision is to either offload the image or process it locally. An image, if locally processed, has to go through all the layers of the CNN, since there is no accelerated CNN. To study the benefits of adding a Filtering stage, we run the same set of simulations by varying the system parameters. Our model outperforms the standard two-stage model in all cases.

Fig. 9 shows that when using three-stages the completion time reduces by 35 percent; that decrease comes at a cost of less than 1 percent in recall. The gap between the two models increases as the hit-rate decreases. At low hit-rates, more images are processed locally. Many images are discarded in the Filtering stage and do not go through the expensive CNN, which results in time savings, as well as leads to the small decrease in recall.

Similarly, when the number of images in the system increases, given a fixed hit-rate, the number of negative images is going to increase. As a result, those images can be discarded after the Filtering Stage, leading to an increase in savings when using our three-stage model. From these results, *we can clearly see the advantage of adding an intermediate decision stage*, and having the option to offload later if the network or cloud conditions change.

The system in [25] has goals similar to ours but is geared towards video processing. It proposes an adaptive algorithm that attempts to reduce the completion time of processing videos across devices by assigning some videos to be processed in a cloud, and others locally on the devices. We adapted this system to process images instead of videos to provide a fair comparison. The key idea in the algorithm in [25] is to determine which devices have the longest expected completion



(a) Time (b) Recall

Fig. 9. Three versus two stage.

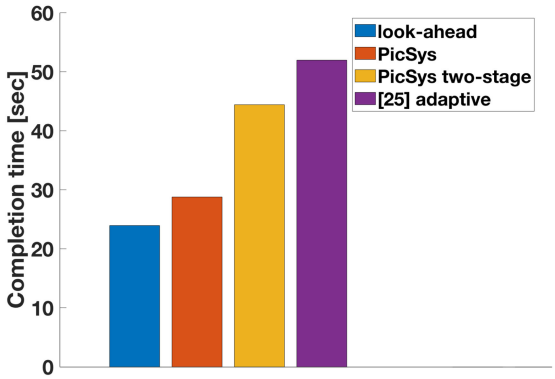


Fig. 10. Comparison of PicSys to [25].

time, and assign these to the cloud to reduce the overall system completion time. Unlike PicSys, the system in [25] does not consider the hit rate, the fact that images that have the object of interest must be uploaded to the cloud even if they are processed locally on the mobile device, or energy. From Fig. 10, it can be seen that PicSys does 44 percent better than [25]. This reduction in completion time is due to accounting for the hit-rate and the introduction of a higher speed filtering stage. We also show that even in the absence of a filtering stage the two-stage algorithm of PicSys still performs better than [25] by about 15 percent.

6.2.6 Simulation Results - Energy

In this set of simulations we analyze our energy-heuristic algorithm and highlight the important results. We consider user thresholds of 1, 2 and 3 percent of total energy to be used for this application. Unless stated otherwise, the distributions and parameters for the simulations are similar to Section 6.2.3. The new parameters are: $e^s = 0.00016\%/\text{img}$, $e^t = 0.0006\%/\text{img}$ for a 12 Mbps Wifi connection, $e^t = 0.0032\%/\text{img}$ for LTE with the same throughput, $e_\alpha = 0.0037\%/\text{img}$ and $e_\beta = 0.0103\%/\text{img}$. There are ten devices ($|M| = 10$), and $|I| = 10,000$.

Fig. 11 illustrates the completion times for the energy-heuristic in three different cases: Wifi network with devices using 1 percent of the total battery for crowd-sourcing each, Wifi network with devices using 3 percent battery for crowd-sourcing, and an LTE network with devices using 3 percent battery for the same purpose. From Fig. 11 it can be observed that in the Wifi scenario of 3 percent for

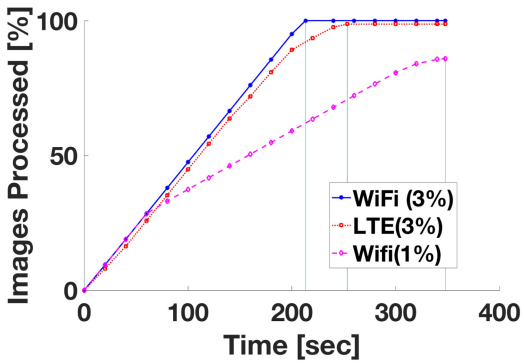


Fig. 11. Energy-heuristic: varying a_m.

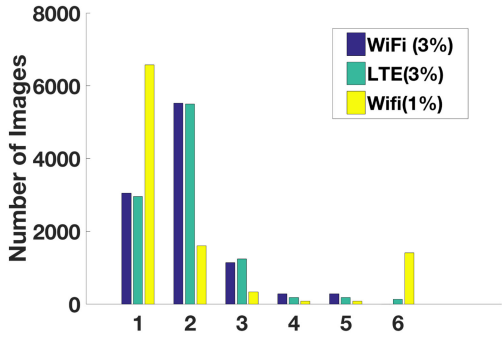


Fig. 12. Decision stage distribution.

crowd-sourcing the energy is sufficient and the devices follow the time-heuristic.

Having considered the effect of the maximum energy for crowd-sourcing, we proceed with looking at the distribution of stages at which the decisions are being made. In that direction, Fig. 12 depicts the number of images that go through each of the system's decision stages. The stages in the x-axis are as follows: 1:Initial-offload, 2:Filtering-discard, 3:Filtering-offload, 4:Selection-discard, 5:Selection-upload, 6:Image-not-processed.

As expected, the time-heuristic with no power budget has the shortest completion time. The higher the available energy, the closer the energy-heuristic is to the time-heuristic. When using Wifi, the stricter the energy constraint, the more initial-offloading, as shown in Fig. 12. With the strict energy constraint (1 percent), the system completes 86 percent of the images. When using the time-heuristic with the 1 percent energy constraint, only 51 percent of the images are processed, because the time-heuristic does not consider the energy budget. This illustrates the effectiveness of the energy-heuristic. We have also observed that in the 2 percent-case (not shown in Fig. 12), the device is able to process all the images, but is $1.3\times$ slower than the time-heuristic.

On the other hand, when using LTE, the cost of offloading becomes expensive and a 3 percent energy constraint is not sufficient to follow the time-heuristic, as in the case of Wifi. In Fig. 12 we see that at a restrictive energy constraint, LTE does more local processing than initial-offload, as opposed to Wifi.

To summarize, in energy-constrained cases, there is a tradeoff between completion time and the amount of images processed.

7 Related Work

Systems and techniques for computation offload have been long researched due to their promise in minimizing either mobile power consumption, or total computation time, or both. In that direction, Hermes [26] formulates an NP-hard problem to minimize the application latency while meeting prescribed resource utilization constraints and proposes a novel fully polynomial-time approximation scheme. In [27], authors optimize computation offload via distributing workload among multiple servers in a cloud. Glimpse [28] is a real-time object recognition system; it does local tracking and video labeling, and only sends frames that are likely to have new information to the server for object recognition.

Some video and image distributed crowd-sourcing systems have been proposed. Vigil [29] builds a video surveillance system that analyzes images from a number of cameras, leverages edge computing and only uploads a fraction of non-redundant videos, thereby optimizing bandwidth consumption. SmartEye [30] does image search and mines images to find image similarities and only uploads unique images via in-network deduplication. Neurosurgeon [31] dynamically partitions DNNs to move part of the processing to the edge, aiming to achieve low latency or energy consumption. The minimization is done for a single device only. A similar setting as ours is considered in [25] and [32], but the choices of processing is restricted to either local processing or offloading.

In [33] and [34], the block mining process in exchange for monetary rewards is considered, where different incentive mechanisms that preserve privacy in crowdsensing applications are proposed. We assume voluntary participation in our work, but we can capture the case with incentives as well. This is beyond the scope of our work.

Early classification was studied in previous work such as Branchynet [35], where the authors propose using dynamic graphs with the addition of side branches to a linear CNN to allow early exit, based on a confidence metric to reduce inference cost. In [36], the system named Blockdrop is proposed to speed up ResNet by dynamically choosing which blocks to execute, while maintaining similar accuracy to the original ResNet CNN. Techniques such as pruning are also proposed to reduce network sizes and enabling them to run faster and lighter on mobile devices. In [37], there is a layer-by-layer pruning, which reduces the network size, using energy estimation. Energy estimation is performed based on data movement in the memory and the number of multiply and accumulate (MAC) operations, as well as data sparsity.

It is not straightforward to compare these systems in a unified network setup. Our system is not built for a specific CNN to be used, and as more research is developed, new techniques such as [8], [35], [36] and [37] to save on both energy and time are suitable for use. In our paper we chose to use GoogLeNet [7] and partition it, as an example of early exit and Squeezenet [8] as an example of a compressed model. Similar to BranchyNet [35], we utilize the early learned features to get a prediction from early stages. As GoogLenet is a well-developed CNN, by using the caffe [11] framework it is easy for us to control our experiments and deploy the models on different platforms. We also show the effects of choosing the top-k values at each stage, and how that choice affects both accuracy and latency of the system.

More related works to ours are MAUI [38], Odessa [39], CloneCloud [40], ThinkAir [41], which perform fine-grained partitioning and code offload to minimize energy consumption and total execution time. Their primary focus is on enabling code-offload with minimal changes to applications.

The principal goal in [38] is to minimize energy consumption of mobile applications subject to latency constraints. The authors have shown that when there is a good network connection, there is an improvement in execution time. However, the system incurs a large overhead in 3G. The main feature of the algorithm is that the programmer marks the portions of the code that can be run remotely. The server has a MAUI coordinator, the mobile device and the server both have a proxy, profiler and solver. The profiler calculates the runtime and energy cost for each method, as well as the other network parameters. The main limitation of [38] is that it aims to optimize for a single mobile device only. As opposed to [38], PicSys is run on a large number of devices where both Wifi and 4G interfaces can be used to send the images to the server. The other difference is in the objective function; the objective in [38] is to minimize energy consumption, while in PicSys we minimize the system completion time, given energy constraints.

Another work similar to ours is [39]. It uses code offloading, pipelining, and data parallelism to optimize for the makespan (execution time for a frame) and throughput (the rate at which frames are processed). This system operates in the following way: It divides tasks into the compute stage or network stage. In the former, the system calculates the estimated cost of offloading the stage or the estimated cost of spawning more works. In the latter, the estimated cost of offloading the source stage and the estimated cost of offloading the destination stage are calculated. The evaluation is performed on a netbook, laptop and a server, and the evaluation algorithms used are *all-offload*, *domain specific* (where the application specifications and developer's input are used to identify compute-intensive stages that can be offloaded to the server), and an *offline optimizer* (this is an oracle similar to our look-ahead algorithm). The objective in [39] is to minimize the completion time of a single device with no energy consideration. As opposed to [39], which does not consider energy consumption, PicSys is energy-aware, and minimizes the completion time of the entire system.

CloneCloud [40], built on [38], uses application VM clones at the server and partitions applications automatically. The objective is to minimize the completion time of the device. The main difference with PicSys is that this system focuses only on minimizing the completion time of only one device, whereas PicSys considers the entire system.

The system in [41], named ThinkAir, is used for similar applications. The main objective is to improve the performance through cloud computing, and dynamic scaling of computational power. The experiments were run based on an execution time policy with a boundary input value, which is the minimum value for input parameters for which offloading would be beneficial. The scenarios used are local processing on the phone, and sending to the cloud through Wifi and 3G. The evaluation is performed on a single device and on a single cloud. Again, as in the other related works, this system is run on a single mobile phone and optimizes the completion time of that single phone, as opposed to PicSys in which even large systems (with hundreds of phones) can be used and the overall performance optimized.

While PicSys uses offloading for crowdsourcing applications, as several of the other references, there are some key differences. Most importantly, PicSys considers the entire system, not a single device. It also collects all positively identified data, as opposed to simply labeling or processing data. PicSys optimizes for latency subject to energy constraints. If we were to optimize for a single device in a setting where the offload incurs a large cost, the results would lean towards processing all or most data locally. However, since in PicSys the positively classified data have to be *uploaded* to the cloud, PicSys finds better solutions, which are not found by other systems mentioned here. Similarly, in cases where devices have high bandwidth, systems optimizing for a single mobile

device result in *all-offload* ; we have shown that such choice is not always optimal, given the backlog created on the server.

8 Conclusion

This paper presents the design of PicSys, a system for efficiently searching for images over a set of mobile cameras. We formulate two problems: one that is energy-oblivious and the other that is energy-aware. Since the problems are NP-hard, we propose low-complexity distributed online heuristics to optimize for the makespan. We show using experiments and simulations that the performance of the heuristics approaches that of a look-ahead oracle and performs better than other standard algorithms. In future, we plan to consider the system design when user mobility is involved. Potential future work includes the incorporation of caching mechanisms and CNNs that are not pre-trained.

Acknowledgments

This work was supported by the Army Research Laboratory and accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] X. Zhang, Z. Yang, Y. Liu, and S. Tang, "On reliable task assignment for spatial crowdsourcing," *IEEE Trans. Emerging Topics Comput.*, vol. 7, no. 1, pp. 174–186, Jan.-Mar. 2019.

[2] R. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: Current state and future challenges," *IEEE Commun. Magazine*, vol. 49, no. 11, pp. 32–39, Nov. 2011.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.

[5] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 4489–4497.

[6] D. Yang, G. Xue, X. Fang, and J. Tiang, "Incentive mechanisms for crowdsensing: Crowdsourcing with smartphones," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1732–1744, Jun. 2016.

[7] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[8] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size," 2016, *arXiv:1602.07360*.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[11] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*.

[12] sh1r0, "caffe-android-lib," Accessed: Jan. 20, 2018. [Online]. Available: https://github.com/sh1r0/caffe-android-lib

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[14] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, pp. 303–338, 2010.

[15] C. M. Zoo, Accessed: Jan. 15, 2018. [Online]. Available: https://github.com/bvlc/caffe/wiki

[16] F. Bellard *et al.*, "Ffmpeg," Accessed: Jan. 15, 2018. [Online]. Available: http://ffmpeg.org

[17] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.

[18] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.

[19] A. Ignatov *et al.*, "Ai benchmark: Running deep neural networks on android smartphones," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 288–314.

[20] Z. Lu, S. Rallapalli, K. Chan, and T. La Porta, "Modeling the resource requirements of convolutional neural networks on mobile devices," in *Prof. 25th ACM Int. Conf. Multimedia*, 2017, pp. 1663–1671.

[21] B. H. Google, Accessed: Nov. 29, 2018. [Online]. Available: https://github.com/google/battery-historian

[22] A. Arisha, P. Young, and M. El Baradie, "Job shop scheduling problem: An overview," in *Proc. Int. Conf. Flexible Automat. Intell. Manuf.*, 2001, pp. 682–693.

[23] Samsung, Galaxy s4, Accessed: Aug. 27, 2019. [Online]. Available: https://www.samsung.com/uk/smartphones/galaxy-s4-i9505/GT-I9505ZKABTU/

[24] Samsung, Galaxy s9, Accessed: Aug. 27, 2019. [Online]. Available: https://www.samsung.com/uk/smartphones/galaxy-s9/specs/

[25] Z. Lu, K. S. Chan, R. Urgaonkar, and T. La Porta, "On-demand video processing in wireless networks," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, 2016, pp. 1–10.

[26] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3056–3069, Nov. 2017.

[27] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[28] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "GLIMPSE: Continuous, real-time object recognition on mobile devices," in *Proc. 3rd ACM Conf. Embedded Netw. Sensor Syst.*, 2015, pp. 155–168.

[29] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 426–438.

[30] Y. Hua, W. He, X. Liu, and D. Feng, "SmartEye: Real-time and efficient cloud image sharing for disaster environments," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1616–1624.

[31] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2017, pp. 615–629.

[32] Z. Lu, K. Chan, and T. La Porta, "A computing platform for video crowdprocessing using deep learning," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1430–1438.

[33] M. Li *et al.*"Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1251–1266, Jun. 2018.

[34] J. Wang, M. Li, Y. He, H. Li, K. Xiao, and C. Wang, "A blockchain based privacy-preserving incentive mechanism in crowdsensing applications," *IEEE Access*, vol. 6, pp. 17 545–17 556, 2018.

[35] S. Teerapittayanon, B. McDanel, and H. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.

[36] Z. Wu *et al.*, "Blockdrop: Dynamic inference paths in residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8817–8826.

[37] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5687–5695.

[38] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Services*, 2010, pp. 49–62.

[39] R. Moo-Ryong, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proc. 9th Int. Conf. Mobile Syst. Appl. Services*, 2011, pp. 43–56.

[40] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.
[41] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, 2012, pp. 945–953.

Noor Felemban received the BS degree in computer engineering from Prince Mohammad bin Fahd University, Saudi Arabia, in 2012, and the MS degree in computer science and engineering from the Pennsylvania State University, University Park, PA, in 2016. She is currently working toward the PhD degree in computer science and engineering at the Pennsylvania State University, University Park, PA, State College, PA, as a member of the Institute for Networking and Security Research. Her research interests include mobile networks, resource allocation and machine learning.

Fidan Mehmeti received the graduate degree in electrical and computer engineering from the University of Prishtina, Kosovo, in 2009, and the PhD degree from Institute Eurecom/Telecom ParisTech, France, in 2015. After that, he was a post-doctoral fellow at the University of Waterloo, Canada. He is currently working as a postdoctoral scholar at Penn State University. His research interests include performance modeling and analysis of wireless networks in general.

Hana Khamfroush is currently an assistant professor with the Computer Science Department, University of Kentucky. Before joining University of Kentucky, she was a research associate in the Department of Computer Science and Engineering, Penn State University. Her research interests include complex networks, mobile edge and cloud computing, wireless communications, online social networks, and cyber-physical systems. She was named a rising star in EECS by MIT and CMU, in 2015 and 2016. She was selected as one of the 200 young researchers for the Heidelberg Laureate Forum, Germany. She is a reviewer for prestigious IEEE Journals and Conferences, including IEEE JSAC, the *IEEE Transactions on Information Theory*, *IEEE Transactions on Communication*, and *IEEE Transactions on Vehicular Technology*. She is served as a TPC member of many conferences including INFOCOM, ICC, GLOBECOM, IFIP Networking, LCN, and IFIP WWIC.

Zongqing Lu received the BS and MS degrees from Southeast University, China, and the PhD degree from Nanyang Technological University, Singapore, 2014. He is currently an assistant professor with the Department of Computer Science, Peking University. Prior to joining Peking University in 2017, he worked as a postdoc in the Department of Computer Science and Engineering, Pennsylvania State University. His current research interests include mobile/edge computing and learning to cooperate.

Swati Rallapalli received the PhD degree from the University of Texas, Austin, in 2014. She is currently a research scientist at Facebook Inc., working on recommender systems and ads ranking. Prior to this, she was a research staff member at the IBM T. J. Watson Research Center, where her work is focused on distributed video analytics (using CNNs/RNNs) on mobile systems. She also served as a principal investigator on the Quality Aware Semantic Video Analytics under US Army Research Lab funded Network Science Collaborative Technology Alliance, that involved inferring multi-resolution spatial and temporal features from sensed data.

Kevin Chan (S'02–M'09–SM'18) received the BS degree in electrical and computer engineering and engineering public policy from Carnegie Mellon University, Pittsburgh, PA, and the MSECE and PhD degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA. He is currently a research scientist with the Computational and Information Sciences Directorate at the U.S. Army Combat Capabilities Development Command Army Research Laboratory (Adelphi, MD). He is actively involved in research on network science, distributed analytics and cybersecurity. He has received multiple best paper awards and the NATO Scientific Achievement Award. He currently is the co-editor for the IEEE Communications Magazine Military Communications and Networks Series.

Thomas La Porta received the BSEE and MSEE degrees from the Cooper Union, NY, and the PhD degree in electrical engineering from Columbia University, NY. He is currently the director of the School of Electrical Engineering and Computer Science at Penn State University. He is an Evan Pugh professor and the William E. Leonhard chair professor in the Computer Science and Engineering Department and the Electrical Engineering Department. He joined Penn State in 2002. He was the founding director of the Institute of Networking and Security Research at Penn State. Prior to joining Penn State, he was with Bell Laboratories for 17 years. He was the director of the Mobile Networking Research Department in Bell Laboratories, Lucent Technologies where he led various projects in wireless and mobile networking. He is an IEEE fellow, Bell Labs fellow, and received the Bell Labs distinguished Technical Staff Award. He also won two Thomas Alva Edison patent awards. He was the founding editor-in-chief of the *IEEE Transactions on Mobile Computing*. He has published numerous papers and holds 39 patents.