Make Smartphones Last A Day: Pre-processing Based Computer Vision Application Offloading

Jiwei Li†, Zhe Peng†, Bin Xiao†, Yu Hua‡
The Hong Kong Polytechnic University†
Huazhong University of Science and Technology‡

Abstract—**The benefit of offloading applications from smartphones to cloud servers is undermined by the significant energy consumption in data transmission. Most previous approaches attempt to improve the energy efficiency only by choosing a more energy efficient network. However, we find that for computer vision applications, pre-processing the data before offloading can also substantially lower the energy consumption in data transmission at the cost of lower result accuracy. In this paper, we propose a novel online decision making approach to determining the pre-processing level for either higher result accuracy or better energy efficiency in a mobile environment. Different from previous work that maximizes the energy efficiency, our work takes the energy consumption as a constraint. Since people usually charge their smartphones daily, it is unnecessary to extend the battery life to last more than a day. Under both the energy and time constraints, we attempt to solve the problem of maximizing the result accuracy in an online way. Our real-world evaluation shows that the implemented prototype of our approach achieves a near-optimal accuracy for application execution results (nearly 99% correct detection rate for face detection), and sufficiently satisfies the energy constraint.**

I. Introduction

The vision to link together the real and virtual worlds has seen great potentials to come true in the near future due to the increasingly powerful and popular smartphones. A great number of computer vision applications have been developed for smartphones. For example, a British start-up CrowdEmotion [1] develops a technology that can capture people's facial expressions to read their emotions through cameras on smartphones.

One common property of all these computer vision applications is the large amount of energy cost incurred by the intensive computations. Running these applications is prohibitively expensive on battery-limited smartphones. To ameliorate the situation, much work has been done, mostly focused on offloading compute intensive applications to remote cloud servers [2][3]. However, for some applications that involve bulk data transfer, application offloading may not be as energy efficient as expected, as the extra data transmission cost may offset the gains from computation shifting. As a result, how to effectively reduce the data transmission cost becomes a critical yet unsolved issue.

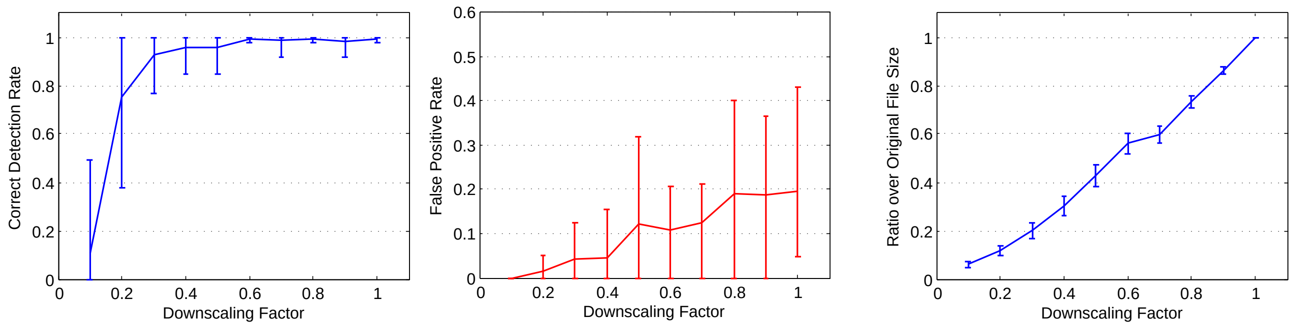
Applications like image processing can provide useful information and one nice property of image processing is that it is not always necessary to input full resolution images so as to yield desirable results. For example, in face detection applications, faces can still be recognized, even though the images go through downscaling or compression before being detected [4]. As the image file size normally drops significantly after being downscaled or compressed, the corresponding overhead of data transfer also decreases substantially. However, correct detection rate for face detection may be adversely affected, since downscaled or compressed images provide less information than the original. This insight provides us with a new way to approach the energy issue.

In this paper, we consider computer vision application offloading in a mobile environment, with focus on the tradeoff between image processing accuracy and energy consumption on smartphones. As the image pre-processing level is correlated with the image process accuracy, we formulate this problem as how to determine the image pre-processing level in order to maximize the image processing accuracy, given specific energy and time constraints. Taking the energy consumption as a constraint is justified by the fact that people normally charge their smartphones daily and thus it is unnecessary to extend the battery life to last more than a day.

However, we are faced with two major challenges in order to solve the problem. First, it is unclear how different image pre-processing levels, i.e. downscaling factor and compression ratio, would impact the ultimate image processing accuracy and energy consumption. We need to determine the right pre-processing level and finish such pre-processing task even before the offloading starts. Second, the proposed approach must not rely on a priori knowledge of future network conditions and incoming workloads in a mobile environment. Any prediction-based techniques should also be avoided, since they cause extra overheads and are highly inaccurate.

To address the first challenge, we conduct real face detection experiments using images fetched from the Internet. Experiment results show that as the downscaling factor drops below 0.3, the correct detection rate drops significantly. We also find that for images compressed in JPEG format, the image file size is almost linearly correlated to the corresponding downscaling factor. In other words, the energy consumption of image uploading is closely related to the downscale factor, as larger images take more time to complete uploading, thus consuming more energy.

Inspired by the Lyapunov optimization framework [5], we propose a decision making approach that utilizes only current network condition and remaining task execution time to tackle the second challenge. Basically, the approach makes



(a) The correct detection rate (b) The false positive rate (c) Ratio over original file size

Figure 1: Downscaling experiments against downscaling factor ranging from 0.1 to 1 with step of 0.1.

two decisions that would affect energy efficiency and result accuracy. First, it decides when the condition check should start based on remaining task execution time, since which it performs condition check periodically by comparing current network condition with a pre-defined value in order to decide whether to start uploading images. Second, it decides the pre-processing level based on current network condition and other information in order to maximize the image processing accuracy while meeting the time and power constraints. We analytically show that an optimal pre-defined value can be found to maximize the image processing accuracy for different condition check starting time. Specifically, we find that earlier condition check starting time results in higher image processing accuracy, but also incurs more extra energy cost.

We implemented a prototype of the decision making approach on the Android platform, and developed an OpenCV-based image processing module on the server. We evaluated the prototype extensively in real-world environments, and compared it with other alternative approaches. From experiment results, we show that pre-processing images before uploading them to the server can effectively reduce the energy consumption on smartphones. We also validate our analytical results of how the condition check starting time and the pre-defined value would affect the image processing accuracy. Finally, we show that our approach achieves a near-optimal image processing accuracy (nearly 99% correct detection ratio for face detection), and sufficiently satisfies the energy constraint, whereas other approaches either fail to satisfy the energy constraint, or produce undesirable result accuracy.

We summarize our contributions as follows.

- We reveal that, to counteract the heavy energy consumption of computer vision applications on smartphones, offloading a pre-processed image to a server is a practical and efficient solution, which has been neglected previously. We conduct our motivating experiments to empirically show how the image pre-processing level would affect the image processing accuracy and energy consumption on data transmission.
- We study the problem of how to maximize the image processing accuracy subject to time and power constraints. Our proposed approach is applicable to any dynamic environments without assuming a priori knowledge of workload pattern and network variation. We analytically show that, by putting configurable parameters, our approach yields different maximum image processing accuracy for different power constraints.
- We implemented our proposed approach and conducted extensive experiments in real-world scenarios. Experiment results show that compared with other approaches, our method achieves a near-optimal image processing accuracy and sufficiently satisfies the energy constraint.

The rest of the paper is organized as follows. We introduce our motivating experiments and the problem formulation in Section II. Followed is the client side design detail of the single task offloading decision making in Section III and the multiple task offloading decision making in Section IV. We present how we implement our proposed approach in Section V and evaluate it in Section VI. Related work can be found in Section VII. Finally, we conclude this paper in Section VIII.

II. Motivation & Problem Formulation

In this section, we present our motivating experiment and formulate our studied problem. First, we study how different image pre-processing levels would impact the image processing accuracy and energy consumption. Then, we give the problem formulation.

A. Face Detection Experiment

1) Overview: First, we conduct face detection experiments to study the effect of different image pre-processing levels on image processing accuracy. We choose face detection, because it is well developed and provides stable software tools. In order to detect faces using face detectors with fixed resolution, such as Viola-Jones [4], the same image is scanned multiple times, each time applied with different scaling factors. However, as images are downscaled to a certain extent, some important feature information for face detection may be lost permanently. Intuitively, we argue that the face detection accuracy will be adversely affected when images are over-downscaled. One purpose of the experiment is to validate whether the image pre-processing level affects the image processing accuracy.

Then, we attempt to analyze how image pre-processing would affect the image file size and thus energy consumption of image uploading. Normally, if a raw image is applied with a downscaling factor of 0.5, the file size will become one fourth of the original size. However, images are usually compressed in the JPEG format, which may not follow this rule. Thus, another purpose of the experiment is to find a mapping relationship between downscaling factor and image file size (namely energy consumption of image uploading). Note that we only consider image downscaling, not image compression, to simplify the analysis.

2) Set-up: We downloaded 12 group photos from the Internet as the sample images in our experiment. Each image is applied with downscaling factor ranging from 0.1 to 0.9 with step of 0.1. Thus, we obtain $12 * 10 = 120$ images in all. Then, each image goes through the Viola-Jones face detector. We record the correct detection rate and the false positive rate for each image, where the former refers to the ratio of the number of correctly detected faces over that of the actual faces, and the latter refers to the ratio of the number of incorrectly detected faces over that of all detected faces. These two terms are commonly used as the metrics to measure the performance of face detection. We also record the change of image file size as the donwscaling factor varies, since image file size directly affects the energy consumption in data transmission. For each of these metrics, we plot the average, the 10th percentile and the 90th percentile of 12 images under each downscaling factor, as shown in Figure-1.

3) Result Analysis: Figure-1a shows how the correct detection rate changes with respect to different downscaling factors. As the downscaling factor drops below 0.3, the correct detection rate drops significantly, sometimes even to 0. This validates our hypothesis. We also observe that for the downscaling factor that is below 0.6, the correct detection rate is quite different at the 10th percentile and the 90the percentile, since some images are more sensitive to the downscaling factor than others in face detection applications. We believe that in other image processing applications, such as crowd counting, we expect to see a much sharper variation of the correct detection rate with respect to the downscale factor.

Figure-1b shows how the false positive rate is associated with different downscaling factors. We observe that like the correct detection rate, the average false positive rate increases as the donwscaling factor increases. As we determine the downscaling factor to maximize the correct detection rate, we also need to be aware of the false positive rate.

Figure-1c plots the change of the ratio of downscaled image file size over its original file size as the downscaling factor increases from 0.1 to 1. We find that due to the JPEG format, the file size is not quadratically related to the downscaling factor, but almost linearly. Since image file size directly affects energy consumption of image uploading, we will use this result to analyze data transmission energy consumption for our proposed approach.

Note that we only show the experiment results of varying the downscaling factor. As another pre-processing method, the compression shows similar effects on the correct detection ratio as the downscaling. *To simplify our analysis, we only consider the downscaling as the pre-processing method in this paper.* In fact, the downscaling can be readily replaced by or combined with the compression in our analytical framework and our implemented system.

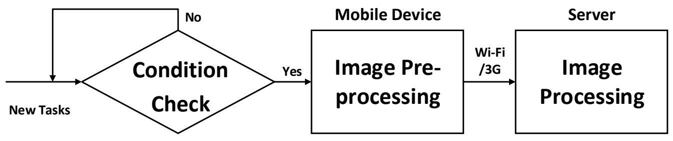


Figure 2: The workflow of the single task offloading.

B. Problem Formulation

We consider a mobile cloud offloading model where images on the smartphone are uploaded to and processed on the server while the result is sent back from the server to the smartphone. In this model, we focus on the energy consumption on the smartphone, the image processing accuracy and the response time. The energy consumption mainly comes from using the network interfaces, while the image processing accuracy is closely related to the image quality. Normally, there is a tradeoff between these two metrics. The response time is usually regarded as a time constraint for image uploading. Based on this model, we formulate our studied problem as an optimization problem, i.e., maximizing the image processing accuracy while satisfying certain response time and power constraints.

III. Single Task Offloading Decision Making

In this section, we focus on a decision making problem for the single task offloading. Two important decisions need to be made in order to maximize the image process accuracy while satisfying time and power constraints. One is to decide when to start data transmission via which network interface, which has a direct impact on energy efficiency, since using network with different type or quality results in completely different energy consumption on data transmission. The other is to determine how to pre-process the image before uploading it to the server, which empirically shows an impact on the image processing accuracy based on our motivating experiments.

In order to understand the problem more thoroughly, we present the basic workflow under the single task scenario, as shown in Figure-2. Basically, as a new task arrives, it will first go through a condition checking process. If it passes, the image in the task will be pre-processed accordingly, and then will be uploaded to a server for further processing via a chosen network. Otherwise, it has to go through the condition checking process again after a fixed amount of time.

A. Methodology

Now, we present our methodology in terms of how to make decisions of when to start data transmission via which network and how to pre-process images. Inspired by the Lyapunov optimization framework, we only utilize current channel status and task delay time to make decisions, without assuming a priori knowledge of future network conditions or using

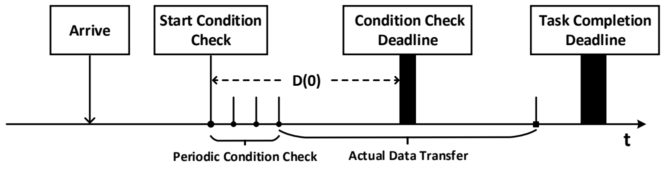


Figure 3: One illustrating example of making decisions for the single task offloading.

predictive methods. We also allow users to specify the time and power constraints to achieve different maximum image processing accuracy. The specific procedures are described in more details as follows, with an illustrating example shown in Figure-3.

1) Estimate the largest image pre-processing level: To ensure a minimum image processing accuracy, we roughly estimate the largest image pre-processing level θ_{max} and the corresponding image file size S_{min} under θ_{max}, based on our findings of the motivating experiments. θ_{max} varies with different types of applications. For example, people counting may require a minimum of 95% accuracy, indicating that $\theta_{max} \approx 0.5$.

2) Calculate the condition check deadline: To ensure that image uploading is completed before its deadline, we calculate the maximum time needed to upload an image with the size of S_{min} using cellular network. Then, we can calculate the condition check deadline before which condition check must performed, as well as the remaining time D between current time and the condition check deadline.

3) Condition Check: Condition check is applied on two variables, namely current Wi-Fi bandwidth B_w and the remaining time D before the condition check deadline. If $D \leq 0$, the task must be executed now via cellular network with the maximum pre-processing level θ applied. Otherwise, it will turn on Wi-Fi and test the bandwidth of the associated AP every fixed amount of time t_{check} until it either passes the condition check or reaches the condition check deadline. If the tested bandwidth B_w is greater than β times the cellular bandwidth, i.e., $B_w \geq \beta B_c$, the task passes the condition check.

4) Determining the Image Pre-processing level: To maximize the image processing accuracy, we need to maximize the image file size S, since images with higher resolution come in larger file size and usually result in better image processing accuracy. Let S' denote the minimum file size that yields a 100% accuracy. Based on $min(S, S')$, we can determine the corresponding image pre-processing factor θ.

Specifically, we can formulate the problem of maximizing S as an optimization problem, i.e.,

$$\begin{aligned} \underset{S}{\text{maximize}} \quad & S \\ \text{subject to} \quad & \frac{S}{B_w} \leq \frac{S_{min}}{B_c} + D, && (1) \\ & P_w \cdot \frac{S}{B_w} \leq P_c \left(\frac{S_{min}}{B_c} + t_{tail} \right), && (2) \end{aligned}$$

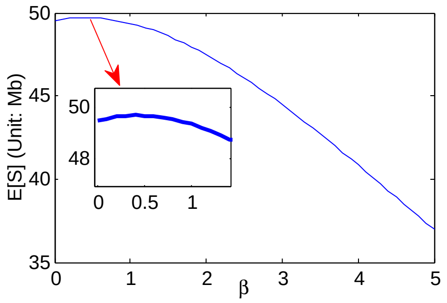


Figure 4: The change of $E[S]$ as β increases for Case 1.

where (1) is the time constraint that image uploading under current Wi-Fi must finish before the task completion deadline, and (2) is the power constraint that the incurred energy cost must not be larger than uploading the image with the size of S_{min} using cellular network. t_{tail} represents the tail time of cellular network that remains in high power state after the last active data transmission. P_w and P_c represent the power consumption of Wi-Fi and cellular network in active transmission state, respectively. B_c is the current bandwidth for cellular network.

Basically, D determines which constraint dominates in this optimization problem. We can obtain the solution as

$$S = \begin{cases} \left(\frac{S_{min}}{B_c} + D \right) B_w, & \text{if } D < D_m \\ \frac{P_c}{P_w} \left(\frac{S_{min}}{B_c} + t_{tail} \right) B_w, & \text{if } D \geq D_m. \end{cases} \quad (3)$$

where $D_m = \frac{P_c - P_w}{P_w} \cdot \frac{S_{min}}{B_c} + \frac{P_c}{P_w} \cdot t_{tail}$.

B. Analytical Framework

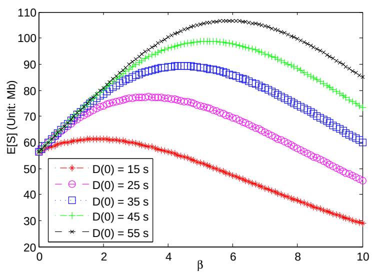
Now, we give a mathematical analysis on how D and β would combine to affect S. Assume that B_w follows an exponential distribution with rate λ. Thus, we have $Pr(B_w \geq \beta B_c) = e^{-\lambda \beta B_c}$, denoted as p. In other words, each time the condition check is performed, the probability that the task will pass is $e^{-\lambda \beta B_c}$. Provided that it passes the condition check, we calculate the conditional expected bandwidth for Wi-Fi, i.e.,

$$E[B_w | B_w \geq \beta B_c] = \int_{\beta B_c}^{+\infty} B_w \lambda e^{-\lambda B_w} \, \mathrm{d}B_w.$$

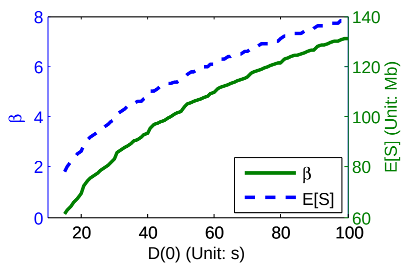
Now, we need to calculate the expected image file size $E[S]$ in two cases according to the equation 3, i.e., $D < D_m$ and $D \geq D_m$.

Case 1: It starts condition check when $D < D_m$. Obviously, $S = [\frac{S_{min}}{B_c} + D] \cdot B_w$, where D decreases by t_{check} each time condition check is performed. Considering that it passes condition check with probability p, it resembles a geometric distribution, except that there is a termination condition. Thus, we can calculate the expected image file size as

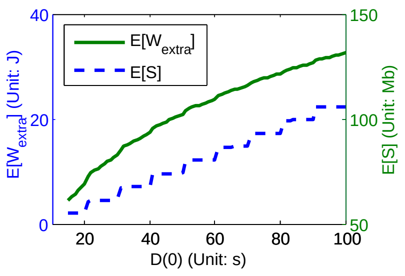
$$\begin{aligned} E[S] = \sum_{i=0}^{l} (1-p)^i p \left(\frac{S_{min}}{B_c} + D(0) - i t_{check} \right) \\ E[B_w | B_w \geq \beta B_c] + (1-p)^{l+1} S_{min}, \end{aligned} \quad (4)$$



(a) The change of $E[S]$ as β increases where $D(0)$ ranges from 15 s to 55 s with step of 10 s for Case 2.



(b) The change of β_m as $D(0)$ increases, as compared to the corresponding maximum $E[S]$ for Case 2.



(c) The change of $E[W_{extra}]$ as $D(0)$ increases, as compared to the corresponding maximum $E[S]$.

Figure 5: Analysis results for Case 2 and the practical consideration.

where $D(0)$ represents the time length between the starting time and deadline for condition check, and $l = \left\lfloor \frac{D(0)}{t_{check}} \right\rfloor$.

Now, we show how $E[S]$ changes with respect to β according to Equation 4. The power consumption in active state of Wi-Fi P_w and 3G B_c is set to 1450 mW and 1400 mW, respectively. The tail time of 3G network is set to 12 seconds. The bandwidth of 3G network is set to 1 Mbps, which we assume remains stable throughout the time. We assume that the bandwidth of Wi-Fi follows an exponential distribution with an average of 4.5 Mbps. Assume $S_{min} = 1$ Mb, and $t_{check} = 10$ seconds. Based on above settings, D_m is calculated as 11.6 seconds, indicating that $D(0)$ must be less than 11.6 seconds in Case 1.

Figure-4 shows the curve of $E[S]$ with respect to β ranging from 0 to 5 with step of 0.1, where we set $D(0)$ to 10.5 seconds, a value between t_{check} and D_m. We enlarge a part in the graph to emphasize the change of $E[S]$, where we can observe the maximum $E[S]$ is achieved when β is approximately equal to 0.5.

Case 2: It starts condition check when $D \geq D_m$. Now, we need to divide the calculation of $E[S]$ into two parts, since D is larger than D_m in the beginning and becomes smaller than D_m as time goes by. First, we calculate the first half when D remains larger than D_m, which is

$$E_1 = \sum_{i=0}^{k} (1-p)^i p \frac{P_c}{P_w} \left(\frac{S_{min}}{B_c} + t_{tail} \right) E[B_w | B_w \geq \beta B_c],$$

where $k = \left\lfloor \frac{D(0)-D_m}{t_{check}} \right\rfloor$.

Then, we calculate the second half when D falls under D_m, i.e.,

$$E_2 = \sum_{i=1}^{n-1} (1-p)^{k+i} p \left(\frac{S_{min}}{B_c} + D(0) - (k+i) t_{check} \right) E[B_w | B_w \geq \beta B_c] + (1-p)^{k+n} S_{min},$$

where $n = \left\lceil \frac{D}{t_{check}} - k \right\rceil$.

Summing up E_1 and E_2, we can get the expected image file size $E[S]$ for Case 2. Keeping the same parameters as in Case 1, Figure-5a shows the curves of E[S] when $D(0)$ ranges from 15 to 55 seconds with step of 10 seconds. For different $D(0)$, different β leads to different maximum $E[S]$. For example, the maximum of $E[S]$ is 61.2499 Mb when β is 1.8 and $D(0)$ is 15 seconds, while the maximum of $E[S]$ is 88.4541 Mb when β is 4.4 and $D(0)$ is 35 seconds. Figure-5b plots how $D(0)$ would affect β_m and the maximum $E[S]$, where β_m represents β that maximizes $E[S]$ under a specific $D(0)$.

C. Practical Consideration in Implementation

In practice, we must consider the overheads of switching on and off the Wi-Fi interface, idle listening and scanning, as well as bandwidth testing. Thus, it is important to determine when to start condition check, since starting condition check earlier does not only yield larger expected image file size, but also incurs more extra overheads. In other words, there exists a tradeoff between $E[S]$ and the overall energy consumption $W_{all} = W_{tran} + W_{switch} + W_{idle} + W_{scan} + W_{test}$, where each item is self-explanatory with its index name. As W_{tran} and W_{switch} are not closely related to the condition check starting time, we focus mainly on W_{idle}, W_{scan} and W_{test} here. Let $W_{extra} = W_{idle} + W_{scan} + W_{test}$. Similar with the derivation of $E[S]$, now we derive the expected extra energy cost $E[W_{extra}]$ with respect to $D(0)$ as follows.

$$E[W_{extra}] = \sum_{i=0}^{m-1} (1-p)^i p \Big(i t_{check} P_{idle} + W_{scan} \left\lfloor \frac{i t_{check}}{t_{scan}} \right\rfloor + (i-1) W_{test} \Big) + (1-p)^m W_c,$$

where $m = \left\lceil \frac{D(0)}{t_{check}} \right\rceil$ and $W_c = P_{idle} D(0) + W_{scan} \left\lfloor \frac{D(0)}{t_{scan}} \right\rfloor$.

We compare $E[W_{extra}]$ against the corresponding maximum $E[S]$ when $D(0)$ ranges from 15 to 100 seconds with step of 1 second, as shown in Figure-5c. We observe that, as $D(0)$ increases, $E[W_{extra}]$ increases almost linearly, while $E[S]$ increases with a slowing rate. To achieve a large $E[S]$ while keeping $E[W_{extra}]$ low, we can specify the condition check starting time as 20 to 30 seconds earlier than the condition check deadline.

IV. Multiple Task Offloading Decision Making

A. Overview

We introduce a scheduler to coordinate the execution time of multiple tasks, aiming at further reducing the energy consumption of Wi-Fi idle listening, scanning and bandwidth testing. The scheduler bundles multiple tasks for execution at once, which can also effectively avoid unnecessary tail energy of using cellular network. The basic idea is stated as follows. As new tasks arrive, the scheduler does not schedule any condition check until it reaches a specified starting time for condition check. Once started, the condition check repeats at a fixed amount of time until it either passes or reaches a new condition check deadline for all existing tasks in the queue. One major difference from the single task offloading is to calculate this new condition check deadline, which must ensure that every task in the queue can be completed before their individual completion deadline. We describe our proposed methodology for the multiple task offloading in more detail as follows.

B. Methodology

To begin with, we describe how the scheduler accepts new tasks and schedules existing tasks for condition check and execution. Then, we present a simple algorithm that iteratively calculates the new condition check deadline.

1) Task execution order: Normally, the scheduler is either in sleeping mode or in running mode. In sleeping mode, the scheduler does not turn on the Wi-Fi interface in order to save energy. It re-calculates the new condition check deadline each time a new task arrives, based on which the new starting time for condition check is specified. When it switches to running mode, the scheduler adopts a FIFO policy to schedule the task execution order. As one task completes, the subsequent task will first start condition check if it has not reached its own condition check deadline. The single task decision making rule still applies here in the sense that the task must start data transfer immediately using cellular network when it reaches its own condition check deadline. When the scheduler is in running mode, new arriving tasks will be added in the queue and wait for condition check and execution. If no tasks exist in the queue, the scheduler will switch back to sleeping mode.

2) Condition check: Now, we presents a simple algorithm to calculate the new condition check deadline. Assume that initially $T_1, ..., T_q$ have arrived in the queue (smaller index means earlier arriving time) and the scheduler is in sleeping mode. The new condition check deadline will be calculated based on these q existing tasks. Basically, the algorithm iteratively compares T_i's condition check deadline with T_{i-1}'s completion deadline as $i = q, ..., 2$, where the earlier one will be regarded as the temporary new condition check deadline against which T_{i-1}'s condition check deadline will be calculated. Note that T_q's condition check deadline is calculated against its own completion deadline. As a result, the new condition check deadline can be calculated, which is T_1's re-calculated condition check deadline.

We define a new term t_{min} to specify the starting time for condition check. In other words, the scheduler must start condition check t_{min} earlier than the new condition check deadline. The choice of t_{min} may affect the number of tasks existing in the queue before the scheduler switches to running mode. In practice, we can tune the parameter t_{min} to achieve

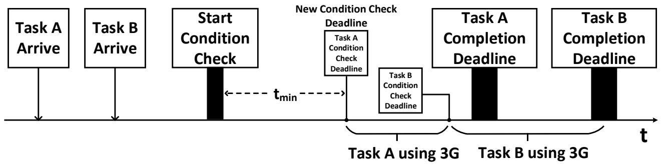

Task A Arrive
Task B Arrive
Start Condition Check
New Condition Check Deadline
Task A Condition Check Deadline
t_{min}
Task B Condition Check Deadline
Task A Completion Deadline
Task B Completion Deadline
t
Task A using 3G
Task B using 3G

Figure 6: One illustrating example of making decisions for the multiple task offloading.

a desirable tradeoff between image processing accuracy and energy consumption on smartphones, just as how we tune $D(0)$ in single task decision making.

Next, we give an illustrating example to show how this algorithm works, as shown in Figure-6.

a) At first, Task A arrives, with its condition check deadline calculated against its task completion deadline. Then, the starting time for condition check is specified simply t_{min} earlier than the new condition check deadline (i.e., Task A's condition check deadline).

b) After a while, Task B arrives before the starting time for condition check. Now, the new condition check deadline for existing tasks in the queue needs to be re-calculated. First, Task B's condition check deadline is calculated against its own task completion deadline. Then, Task A's condition check deadline is re-calculated against whichever is earlier, Task B's condition check deadline or Task A's completion deadline. In this case, Task B's condition check deadline is earlier. As a result, the new condition check deadline and starting time are updated accordingly.

V. IMPLEMENTATION

We implemented our prototype of the decision making framework on the KitKat branch of the Android OS. We tested it on a commercially available smartphone called RedMi, which is equipped with fully functional Wi-Fi and 3G interfaces. We also implemented an face detection module and deployed it on a server with a 3.7GHz i7 CPU, running Windows Server 2012 R2 with an externally accessible IP address. All these implementations comprised approximately 3,000 lines of Java codes.

We implemented the decision maker as a background service on Android. Basically, it listens on incoming tasks submitted from any other third-party app via an `Intent`, a feature provided by Android to enable inter-app communication. We adopted a `LinkedList` to temporarily store these submitted tasks in a FIFO policy, and used a separate thread to handle the actual task execution. We invoked `Thread.sleep` to specify how long the Wi-Fi interface on the smartphone should remain off. The exact sleeping time is calculated based on a function called `getConditionCheckDeadline(Task)`, which is recursively calculated on all existing tasks.

To perform condition check, we included a snippet of codes to test the bandwidth of current associated AP. The idea is that it sends out a HTTP request to a remote server for fetching a test image, which is of fixed file size. By marking the starting time and finishing time of the image fetching, the transmission time can be estimated. Hence, the network bandwidth is roughly estimated as the ratio of the test image file size over the time elapsed during the transmission.

To pre-process images, we used `BitmapFactory` to decode an image into a `Bitmap`, and invoked `createScaledBitmap` to downscale the image. The downscaling factor is calculated based on Equation 3 in Section III. The actual image uploading is handled via the HTTP protocol. In case of network disconnection during uploading, we used a `if` statement to check whether the image uploading is successful or not. The image needs to be uploaded again if its previous attempt fails.

We implemented the face deteion module using OpenCV Java API and Java Servlet. We used a stump-based 20x20 gentle AdaBoost [6] frontal face detector to detect images uploaded via a servlet. Detected faces are marked with rectangles so that we can track the correct detection rate and false positive rate. The client is usually notified via a HTTP response when the image processing finishes on the server.

VI. Real-world Experiments

In this section, we evaluate our proposed decision making approach in real-world experiments. We first describe our experiment methodology both in static and dynamic environments, and then discuss our experiment results.

A. *Methodology*

1) Static environment: In order to obtain a fine-grained measurement of energy consumption on smartphones, we set up a static environment where we use a wireless router to limit the maximum achievable Wi-Fi bandwidth and a power meter called Power Monitor [7] to directly measure the voltage and current of smartphones. Figure-7 shows the setup of power consumption measurement of the RedMi Android phone using Power Monitor.

First, we compare the energy consumption of applying different pre-processing levels and uploading images under the Wi-Fi environment. We adopt the same set of 12 sample images as in our motivating experiment. On the RedMi phone, each of these sample images is applied with the downscaling factor ranging from 0.2 to 1 with step of 0.1 before being uploaded to the server. Note that when the downscaling factor is 1, it means that the image is not pre-processed. For each task execution, we use Power Monitor to measure the energy consumption between the starting time of image downscaling and the finishing time of image uploading. We also manually configure the wireless router to set up a maximum achievable Wi-Fi bandwidth in order to show how Wi-Fi quality would affect energy consumption. We simulate three types of Wi-Fi environment, i.e., low quality (2,000 kbps), medium quality (5,000 kbps) and high quality (10,000 kbps). Under each type of Wi-Fi environment, we conduct the same power measurement experiment on the same set of images.

Second, we compare the energy consumption of pre-processing and uploading images under Wi-Fi and 3G environments. Specifically, we consider four execution methods,

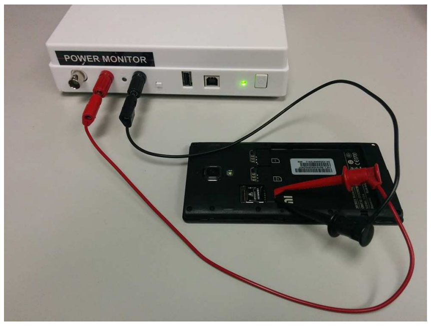
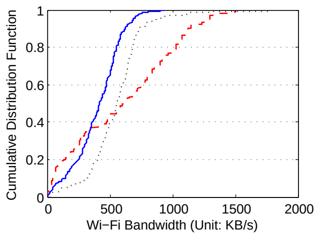
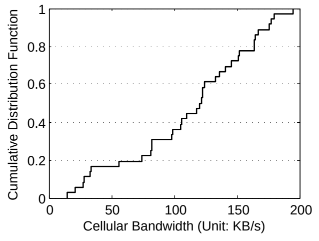


Figure 7: Power consumption measurement using Power Monitor.



(a)



(b)

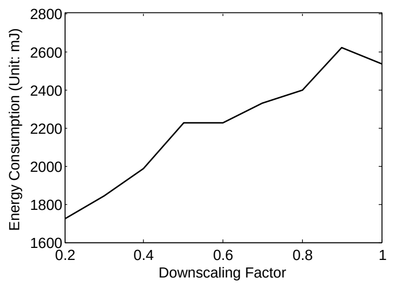
Figure 8: The CDF of (a) Wi-Fi bandwidth (at three different locations)and (b) cellular bandwidth.

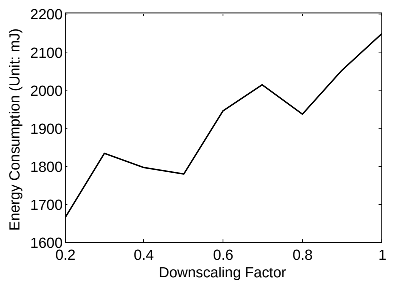
i.e., using Wi-Fi with pre-processing, using Wi-Fi with no pre-processing, using 3G with pre-processing and using 3G with no pre-processing. The pre-processing level is chosen as a downscaling factor of 0.5, while Wi-Fi is configured to medium quality. For each execution method, we supply the same set of 12 sample images and measure the energy consumption for each task execution using Power Monitor.

2) Dynamic environment: In order to evaluate the performance of our proposed decision making approach, we test it under dynamic environments where the phone holder walks around and the wireless network conditions vary.

We choose three different locations to conduct our experiments. To profile the network environment, we collect the bandwidth information of both Wi-Fi and 3G networks at each location and plot the CDFs in Figure-8. We observe that Wi-Fi conditions vary with different locations, while 3G bandwidth falls within the range of 30 KB/s to 200 KB/s. Note that we only plot one curve of CDF of 3G, since the other two curves are very similar with current drawn one. We regard these three locations as low quality, medium quality and high quality Wi-Fi environments, respectively.

We conduct the experiments by walking in three locations and uploading those 12 sample images in a pre-defined sequence. We adjust two parameters, i.e., condition check duration and β, to demonstrate how they affect the performance of our approach. When experimenting on condition check duration, we keep β fixed, and vice versa. We set the *completion deadline* for all tasks to 60 seconds and the *energy constraint* to 5 mAh on the RedMi phone based on our static experiment results. We also compare our approach against other alternative approaches, i.e., instant check and instant


(a)


(b)

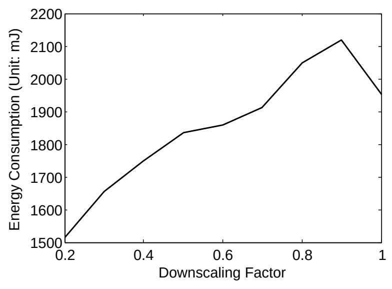

(c)

Figure 9: Energy consumption measurement of pre-processing and uploading images at different downscaling factors under three different Wi-Fi environments. The maximum Wi-Fi bandwidth is set to (a) 2,000 kbps, (b) 5,000 kbps and (c) 10,000 kbps, respectively.

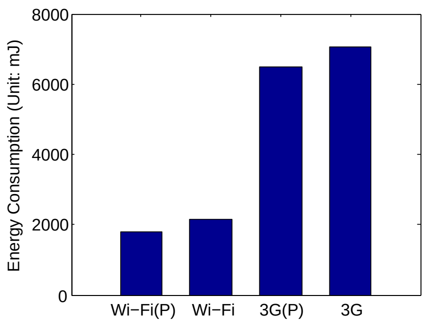
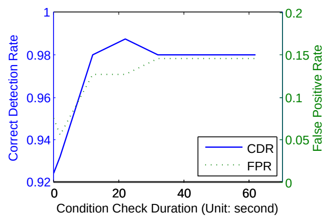


Figure 10: Energy consumption comparison of using Wi-Fi and 3G.


(a)

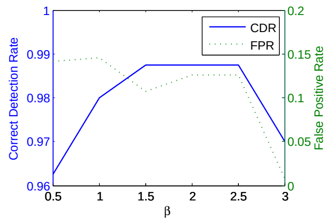
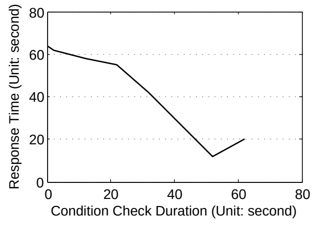

(b)

Figure 11: The correct detection rate and false positive rate with respect to (a) condition check duration and (b) β.


(a)

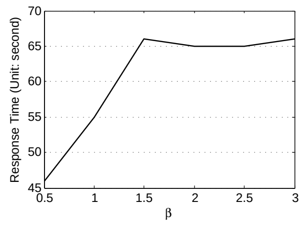

(b)

Figure 12: The response time with respect to (a) condition check duration and (b) β.

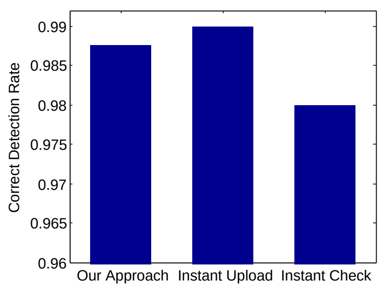
upload, in terms of image processing accuracy, energy consumption and response time. Since the use of Power Monitor is limited in the lab environment, we use the built-in tool in Android called `dumpsys batterystats` to observe the energy consumption for our application.

The *instant check* method is the extreme version of our approach. It starts condition check right after arrival, and uploads images when it either passes condition check or reaches the deadline. It is equivalent to our approach when the specified condition check duration is larger than current remaining time till the deadline. The *instant upload* method uploads images right after arrival using available network. It adopts a pre-processing level that yields the best image processing accuracy. This method ignores the energy constraint, but is still much better than previous work that does not leverage pre-processing in terms of energy efficiency.

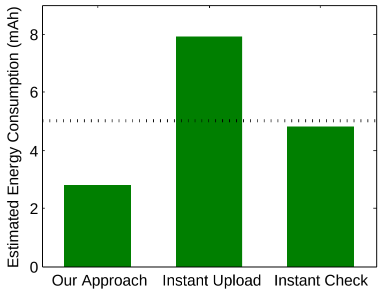
B. Results

1) Static environment: Figure-9 shows the energy consumption of pre-processing and uploading images at different downscaling factors under three different Wi-Fi environments. We observe that as the downscaling factor increases from 0.2 to 0.9, the corresponding energy consumption also increases. However, we also notice that the energy consumption drops when the downscaling factor changes from 0.9 to 1 in Figure-9a and from 0.8 to 1 in Figure-9c. On the other hand, the improvement in Wi-Fi quality shows reduction on energy consumption, as it takes less time to transmit data using faster Wi-Fi. Figure-10 plots the energy consumption comparision of using Wi-Fi and 3G with/without pre-processing. The most energy efficient method is to use Wi-Fi with pre-processing, while the most energy consuming method is to use 3G with no pre-processing. It shows that pre-processing helps save energy on the smartphone, and that Wi-Fi is far more energy efficient than 3G.

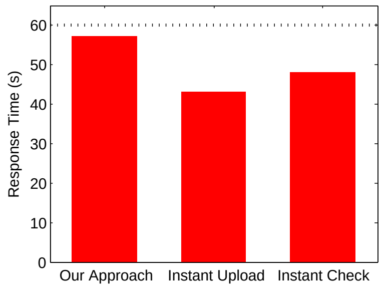
2) Dynamic environment: Figure-11 shows how the correct detection rate and false positive rate are affected by condition check duration and β. We observe that both metrices increase, as the condition check duration increases from 0 to 20 seconds, and both drop to a stable value when the condition check duration is larger than 20 seconds. As β increases, the curve first goes up, then stays flat and finally goes down, which fits nicely with our analytical model. Figure-12 charts the variation of response time with respect to condition check duration and β. Longer condition check duration leads to earlier starting time for condition check, thus resulting in shorter response time, while larger β lowers the chances that the condition check passes, thus resulting in longer response time.



(a)



(b)



(c)

Figure 13: Comparison results of our approach, instant check and instant upload in terms of (a) correct detection rate, (b) energy consumption and (c) response time.

Figure-13 shows the comparison results of our approach, instant check and instant upload. The instant check approach satifies both the energy and time constraints, but achieves the lowest correct detection rate among all. The instant upload approach delivers the best result in correct detection rate and response time, but fails to satisfy the energy constraint. Our approach achieves a nearly 99% of correct detection rate, very close to the instant upload approach. It only incurs less than 3 mAh of energy consumption, which sufficiently satisfies the energy constraint. It also limits the overall response time under 60 seconds, which satisfies the time constraint.

VII. Related Work

Recently, much work has been done in the area of application offloading. MAUI [2] and CloneCloud [3] are among the first systems that offload applications from smartphones to servers in order to save energy, as these offloaded applications are usually compute-intensive. ThinkAir [8] extends MAUI and CloneCloud by offloading to multiple VM images, with more focus on system's scalability. Odessa [9] is intended for interactive perception applications, targeting at performance improvement. However, these systems do not consider any optimization on data transmission. Our approach utilizes the idea of pre-processing images before uploading them to servers, which can greatly reduce the energy consumption.

Link selection is another hot research topic that arises recently. TailEnder [10] smartly schedules data transfers to Wi-Fi so as to minimize the high tail energy overheads incurred by cellular network. Based on the Lyapunov optimization framework, SALSA [11] utilizes channel conditions and local information to make link selection decisions, achieving an near-optimal energy-delay tradeoff on smartphones. BreadCrumbs [12] and Wiffler [13] strive to delay data transfers so as to offload more data on Wi-Fi, based on its prediction on future Wi-Fi connectivity. Our work borrows the link selection idea from above work, but goes beyond that by using image pre-processing to further reduce the energy consumption.

VIII. Conclusion

We propose an online decision making approach to determining the pre-processing level for computer vision application offloading. We solve the problem of maximizing the image processing accuracy while satisfying the energy and response time constraints. Our approach does not require any a priori knowledge of future network variation, or use any predictive techniques. Our analytic results suggest the settings for obtaining optimal results under specific constraints. We implemented a prototype of our approach and evaluated it extensively in real-world scenarios. Experiment results show that as compared with other alternative approaches, our approach achieves a near-optimal image processing accuracy while sufficiently satisfying the energy constraint.

References

[1] Crowdemotion enables reading facial expressions. [Online]. Available: http://www.crowdemotion.co.uk/

[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. of ACM MobiSys*, 2010.

[3] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. of ACM EuroSys*, 2011.

[4] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE CVPR*, vol. 1, 2001.

[5] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[6] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*. Springer, 1995, pp. 23–37.

[7] Monsoon power monitor. [Online]. Available: https://www.msoon.com/

[8] 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 *IEEE INFOCOM*, 2012, pp. 945–953.

[9] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the ACM 9th Mobisys*, 2011, pp. 43–56.

[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM IMC*, 2009, pp. 280–293.

[11] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proc. of ACM MobiSys*, 2010.

[12] A. J. Nicholson and B. D. Noble, "Breadcrumbs: forecasting mobile connectivity," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*. ACM, 2008, pp. 46–57.

[13] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3g using wifi," in *Proc. of ACM MobiSys*, 2010.