Adaptive Quality Optimization of Computer Vision Tasks in Resource-Constrained Devices using Edge Computing

Conference Paper · May 2019

DOI: 10.1109/CCGRID.2019.00061

CITATIONS
0

READS
54

4 authors, including:

Anas Toma
An-Najah National University
18 PUBLICATIONS **114** CITATIONS
SEE PROFILE

Jan Eric Lenssen
Technische Universität Dortmund
18 PUBLICATIONS **121** CITATIONS
SEE PROFILE

Jian-Jia Chen
Karlsruhe Institute of Technology
152 PUBLICATIONS **2,813** CITATIONS
SEE PROFILE

Adaptive Quality Optimization of Computer Vision Tasks in Resource-Constrained Devices using Edge Computing

Anas Toma, Juri Wenner, Jan Eric Lenssen and Jian-Jia Chen
Department of Computer Science, TU Dortmund University, Dortmund, Germany
E-mail: firstname.lastname@tu-dortmund.de

***Abstract*—This paper presents an approach to optimize the quality of computer vision tasks in resource-constrained devices by using different execution versions of the same task. The execution versions are generated by dropping irrelevant contents of the input images or other contents that have marginal effect on the quality of the result. Our execution model is designed to support the edge computing paradigm, where the tasks can be executed remotely on edge nodes either to improve the quality or to reduce the workload of the local device. We also propose an algorithm that selects the suitable execution versions, which includes selecting the configuration and the location of the execution, in order to maximize the total quality of the tasks based on the available resources. The proposed approach provides reliable and adaptive task execution by using several execution versions with various performance and quality trade-offs. Therefore, it is very beneficial for systems with resource and timing constraints such as portable medical devices, surveillance video cameras, wearable systems, etc. The proposed algorithm is evaluated using different computer vision benchmarks.**

I. Introduction

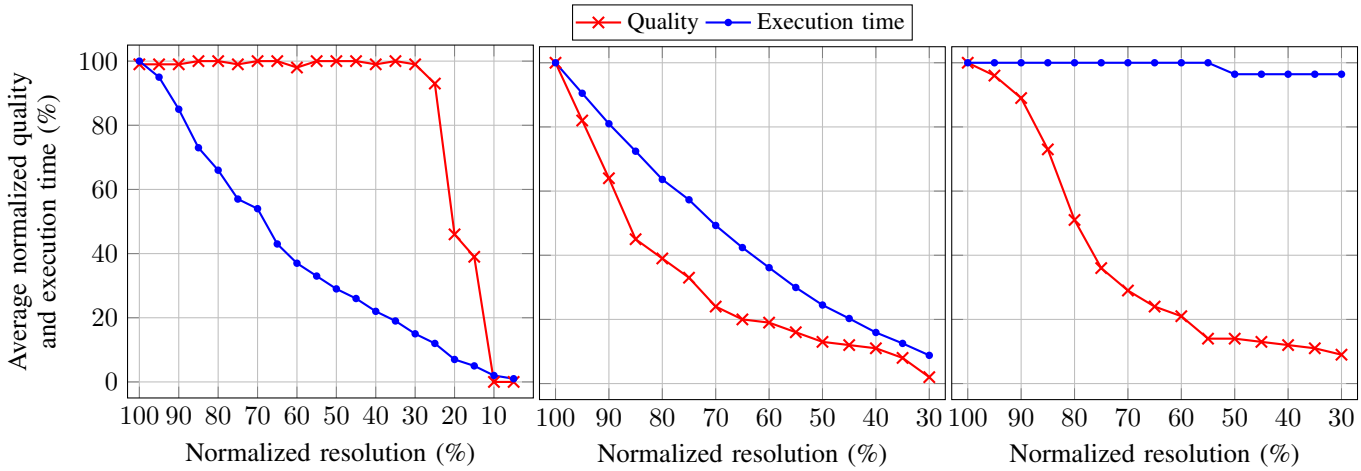
In recent years, there has been an increasing interest in integrating computer vision applications into mobile and sensor devices such as smartphones, tablets and surveillance video cameras. Typical applications include augmented and virtual reality, face recognition, optical character recognition, interactive gaming, etc [1]. The main tasks in these applications are image/video processing and machine learning inference tasks, which should usually be executed iteratively within a given time frame or a deadline, e.g., the period of time between two consecutive video frames. However, some devices might not be able to finish all the required tasks within the deadline due to temporary or permanent resource limitations. In this case, partially computed results may become useless, wasting the resources used to compute them.

It has been shown in the literature that the critical video resolution for computer vision tasks, i.e. the minimum resolution required to ensure an acceptable accuracy, is much lower than the minimum video resolution that is acceptable for human vision [2]. Therefore, we can drop some contents of the input image, such as reducing the resolution, to make it possible for the resource-constrained devices to finish all the workload tasks in time. This reduction of input information introduces a quality vs. resource consumption trade-off. Also, for some tasks, the quality of the result does not decrease at all if all dropped information is irrelevant to the objective task. For instance, changing the resolution will generate several execution versions with corresponding execution times and quality levels. Although this may affect the total quality of the results, it can be maintained within acceptable levels. The term *quality* is used in this paper to refer to the quality of the result such as the accuracy of a recognition task or any other metric that describes the goodness of the result. Figure 1a shows that the execution time of a face recognition task can be reduced by 85% of the original time while we sacrifice only 1% points of quality, i.e., the accuracy.

However, this technique still suffers from two main problems. First, it is not suitable for all computer vision tasks. For some tasks, the quality may decrease almost linearly with the execution time by reducing the resolution as shown in Figure 1b. In this case, the technique should be used carefully and to a certain degree. For other tasks, the quality may strongly decline or drop once we start reducing the resolution without a significant reduction in the execution time as shown in Figure 1c. We call tasks whose quality can not be optimized, or those which have a high threshold of critical quality, *quality-sensitive* tasks. Second, one application may include many tasks that should be executed within a deadline. Selecting an inappropriate execution version (or quality) of a task with a relatively long execution time may affect the quality of the remaining tasks or makes it difficult to finish them in time. In this work, we utilize edge computing to solve both issues as described in the following.

Edge computing is a computing paradigm or architecture where cloud computing services are provided at the edge of the network, i.e. end user devices [4]. This paradigm is used to reduce the workload of cloud servers and their network traffic [5], because it is expected that 94% of computing workloads will be processed by cloud servers and the cloud IP traffic will reach 19.5 ZB by 2021 [6, 7]. Edge computing provides the possibility to use remote and nearby devices as supporting resources by offloading the quality-sensitive tasks to them. Figure 2 shows the average execution time and the average quality of a convolutional neural network performing nanoparticle classification [3], where executing the task remotely or using lower resolution may lead to a significant decrease in the execution time with only up to 1.08% points of quality reduction. However, an appropriate edge node should be selected for each task individually, because the execution time of the offloaded task and the quality of the result depend on the computation capabilities of the selected node. Having more than one location to execute the offloaded tasks generates different execution versions as well,

Published in the 19th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGrid 2019) Larnaca, Cyprus, May 2019.



(a) Face recognition (b) Face detection (c) Eye detection

Fig. 1: Average normalized quality and execution time of three computer vision tasks for different normalized resolutions.

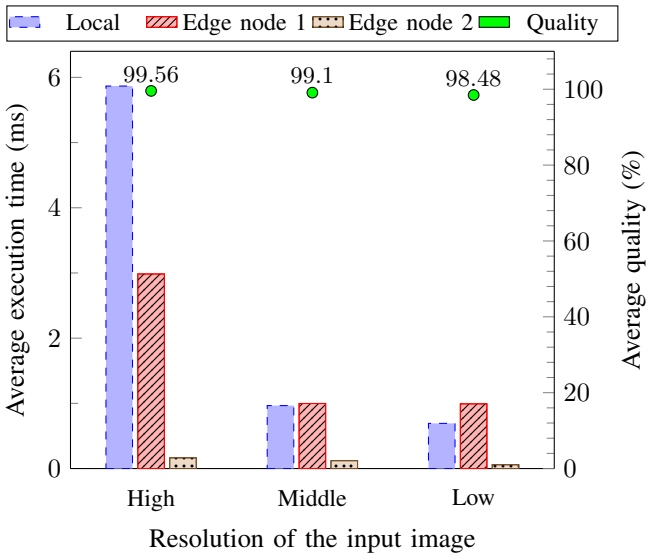


Fig. 2: Average execution time and average quality for a nanoparticle classification task using a mobile biosensor [3].

which needs to be considered by automated schedulers.

In this paper, we propose an adaptive algorithm based on dynamic programming, which maximizes the total quality for computer vision tasks under resource and time constraints. This includes selecting the execution configuration (e.g. the resolution of the input image) and the execution location (i.e. on local machine or on a specific edge node) of each task based on the available resources such as the computation capabilities of the local device, the available edge nodes and the network bandwidth. The algorithm can also be used for any type of tasks that have different execution versions and it is not specifically limited to computer vision tasks. The contribution of this paper can be summarized as follows:

- We conducted extensive experiments using different computer vision tasks to generate many execution versions and to show that in many cases the execution time can be reduced significantly while sacrificing only a very small amount of quality.
- We adopt edge computing to reduce the workload of the local device or to obtain results with higher quality.
- We provide a formal framework for tasks that can be executed using different versions, which is also compatible with the edge computing model.
- An adaptive dynamic programming algorithm is proposed to find the execution versions that maximize the total quality based on the available resources.
- The algorithm is evaluated using different benchmarks that include computer vision tasks.

TABLE I: Quality Evaluation of a Television Picture [8]

Quality		Description
Value	Text	
1	Excellent	Highest desired quality
2	Fine	Quality with enjoyable viewing
3	Passable	Acceptable quality
4	Marginal	Poor quality with a desire to be improved
5	Inferior	Very poor with possibility to be watched
6	Unusable	Cannot be watched

II. Background and Literature Review

A. Quality Metrics

Although any quality metric can be used to evaluate the result of a task execution, we provide several of them for computer vision tasks in this section. Quality evaluation can be either objective based on mathematical calculations or subjective based on human observation. Some examples of objective metrics are root-mean-square (rms) error, mean-square Signal-to-Soise Ratio (SNR_{ms}), Structural Similarity

(SSIM) and accuracy. The subjective metrics are usually used to evaluate the quality of the image used for human vision. Table I shows quality evaluation of a television picture, where the quality is described by humans and then categorized into 6 levels. Furthermore, each level can be described more precisely using different scales such as $\{+1 = \text{high}, 0 = \text{normal}, -1 = \text{low}\}$ [8].

B. Literature Review

A real-time object recognition system for mobile devices was proposed by Chen et al. [9]. The system performs detection, recognition, labeling and tracking of an object in the captured video. It executes these computation-intensive tasks remotely on server machines to improve the performance. To avoid the degradation in the accuracy of object recognition tasks due to potential network delays, the system uses an active cache of video frames to perform object tracking on the local device with the help of some hints from the server. Korshunov and Ooi [2] study face analysis algorithms to find the minimum video quality required to ensure an acceptable accuracy. It also investigates video quality metrics in order to find the critical quality level. It has been shown that the critical video quality for computer vision has much lower bitrate than the minimum video quality that is acceptable for human vision. Both studies above are limited to specific domains.

Hu and Cao [10] propose a quality-aware traffic offloading (QATO) framework to save energy and reduce delay for mobile devices. The main idea is to offload tasks from the mobile device with low service quality to neighboring devices with better service quality through direct communication techniques. The paper also provides analysis for the factors that affect the service quality for different wireless carriers. Toma et al. [11] present a middleware to save energy in mobile computing devices that use cloud applications. Energy saving comes from reducing the energy consumed during the communication with the remote server in the cloud. Any available nearby device can be used as an auxiliary server either to forward the data to the remote server, or to execute the offloaded tasks directly. The operating mode of the auxiliary server depends on its available resources, the workload of the mobile device, the availability of the remote server, and the bandwidth of the connection between those three devices. However, the middleware can offload tasks only for one nearby device.

Taken together, the limitations of the related studies can be summarized as follows: Some approaches are dedicated for specific tasks such as object or face recognition [2, 9]. The system proposed by Chen et al. [9] always offloads the computation-intensive tasks to a remote server. However, this service may be influenced by any fluctuations in the network connection or by the availability of the remote server. The work in [11] tries to solve this problem by using a nearby device as an auxiliary or surrogate server. But, it is only possible to offload tasks to one nearby server in this work. The framework presented by Hu and Cao [10] considers mainly the service quality of the wireless carriers for offloading decision. Moreover, most of state-of-the art approaches in this area either improve the performance or reduce the energy consumption without improving or considering the quality of the result. Our proposed algorithm can be used for any type of tasks, normal tasks (i.e. tasks with only one execution version) or tasks with different execution versions. It is designed to be compatible with edge computing paradigm so that the tasks can be offloaded to more than one edge node or remote server. It is also compatible with the former computation offloading algorithms where the tasks have only two versions of execution, on the local device or on the remote server. The algorithm also allows the system to adapt to any changes in the available resources or to uncertain conditions by providing many execution alternatives. Task execution can be moved either to other edge nodes if the current one is overloaded, or to the local device with an acceptable quality if the network connection is weak or unavailable.

TABLE II: Techniques used to reduce the size of the input image [8].

	Component	Technique
1	Color space	Converting colored images to gray scale images
2	Color depth	Reducing the number of color levels
3	Resolution	Reducing the number of total pixels that represent the image
4	Video frames	Reducing the number of frames in a video

III. Adaptive Quality Optimization

A. System Structure

We adopt the model of edge computing as system structure. It consists of three main components: a local device, edge nodes/servers and a cloud of computers. The tasks to be executed belong to the local device and can be executed on any of these three components. The local device can be any computing device such as a mobile device, a wearable device, a video surveillance camera, a portable medical device, a mobile phone, etc. Both edge nodes and cloud servers represent remote servers to the local device. The execution of the tasks on the edge nodes has lower network overhead than executing them on the cloud servers, because the edge nodes are usually closer to the local device. However, cloud servers are usually more powerful.

B. Execution Versions

Most of computer vision tasks are computation-intensive while it may be required to run them on resource-constrained devices. Instead of improving these devices, which also results in additional costs, we try to reduce the workload by reducing the size of the input images. For the sake of completeness, we study data redundancy in image representation to find the irrelevant components that can be dropped without affecting the quality of image processing tasks. Other relevant components can be dropped as well if the effect on the quality can be tolerated. Table II explains briefly some of these techniques [8]. We try also to categorize the most commonly used image processing tasks in computer vision according to the techniques that can be applied on them from Table II as shown in Table III.

The technique of reducing the resolution of the input image was used to generate different execution versions from many

TABLE III: General image processing categories.

Category	Description	Technique from Table II
Detection	Detecting and locating the object of interest	1, 2, 3
Recognition	Identifying objects	1, 3
Motion detection and analysis	Detecting a change in the location of an object and describing it	1, 3, 4
Monitoring	Examining a video or an image by naked eye	2, 3, 4

computer vision tasks as described in Subsection IV-A. The tasks can be classified into three categories according to the trade-off between their quality and execution time as follows:

1) Quality-tolerant: the execution time can be reduced significantly without affecting the quality, or by sacrificing only a very small amount of it as shown in Figure 1a.
2) Quality-equivalent: the reduction in both quality and execution time is nearly equivalent as shown in Figure 1b.
3) Quality-sensitive: quality degradation without much impact on the execution time as shown in Figure 1c

C. System Model

The local device executes a set of tasks that arrive periodically at the same time, have the same period, and require execution within a common time frame. The period is the minimum time frame between the arrival of two consecutive jobs of the same task. The time frame can be a relative deadline for soft-real time tasks [12], an execution horizon [13], an execution window [14], etc. The same task can be executed under different configurations, which generates different execution versions as well. The task can be also executed remotely, mostly on a more powerful server, to obtain results with higher quality than the local execution or to reduce the workload of the local device. The execution time of the task and the quality of the result depend on the corresponding version.

1) Task Model: Given a set $\mathcal{T}$ of n frame-based real-time tasks. Each task $\tau_i \in \mathcal{T}$ can be executed under configuration j, $j = 1, 2, \ldots, m_i$, and is associated with the following average parameters:

- Local execution time c_{ij}: The amount of time required to finish the task on the local device.
- Sending time s_{ij}: The amount of time required to send the data from the local device to the server. It also includes any potential post processing.
- Remote response time p_{ij}: The interval length starting from the time when the task arrives to the server until the time when the local device starts receiving the result from the server.
- Receiving time r_{ij}: The amount of time required to receive the result from the server.
- Quality q_{ij}: The quality of the result.

In case of local execution under configuration j, the CPU of the local device is allocated for task τ_i for c_{ij} amount of time. In case of remote execution, the CPU of the local device is allocated for s_{ij} amount of time for sending the data to the server and r_{ij} amount of time for receiving the result. Each execution version j has a corresponding quality q_{ij}. Tasks can be represented by the following tuple: $\tau_{ij} \equiv (c_{ij}, s_{ij}, p_{ij}, r_{ij}, q_{ij}, v_{ij})$, where $1 \leq j \leq m_i$ and $1 \leq i \leq n$. Furthermore, $s_{ij} = p_{ij} = r_{ij} = \infty$ for local execution and $c_{ij} = \infty$ for offloading. The binary value v_{ij} equals 1 if the version j of the task i is selected for execution, otherwise 0.

2) Server Model: The remote server may have many virtual servers to handle the requests from many devices, or to execute many offloaded tasks at the same time. For the sake of simplicity, we will use the term *remote server* to denote the virtual server regardless of its location on the edge node or on the cloud. Any resource reservation technique can be used to manage the sharing of the server's processor and to provide response time guarantee for the offloaded tasks as shown in the middleware proposed in [12].

D. Problem Definition

Given a set $\mathcal{T}$ of n real-time tasks required to be executed on a resource-constrained device within a common frame of time, e.g. the deadline D. The execution of the tasks can be also performed on a remote server, however, with additional communication overhead. Each task has m_i execution versions, including the execution versions on the server, where each version has a corresponding execution time and result's quality. *The MAximum Quality (**MAQ**) The problem is to find an execution version for each task so that the total obtained quality is maximized and all the tasks finish within the deadline.* In other words, the objective is to maximize the total *quality rate*, i.e. the total quality within a given time frame. Suppose that the sequence of the tasks is given. The problem can be represented as follows:

$$\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{m_i} v_{ij} q_{ij} \tag{1a}$$

$$\text{such that} \sum_{i=1}^{n} \sum_{j=1}^{m_i} v_{ij}(c_{ij} + s_{ij} + r_{ij}) \leq D \tag{1b}$$

$$\sum_{j=1}^{m_k} v_{kj} p_{kj} + \sum_{i=1}^{k} \sum_{j=1}^{m_i} v_{ij}(s_{ij} + r_{ij}) \leq D, \quad \forall k = 1, 2, \ldots, n \tag{1c}$$

$$\sum_{j=1}^{m_i} v_{ij} = 1, \quad \forall i = 1, 2, \ldots, n \tag{1d}$$

$$v_{ij} \in \{0, 1\}. \tag{1e}$$

Where $s_{ij} = p_{ij} = r_{ij} = 0$ for local execution and $c_{ij} = 0$ for offloading in the equations above. This problem is an $\mathcal{NP}$-hard problem, where the proof can be reduced from the Multiple-Choice Knapsack problem [15].

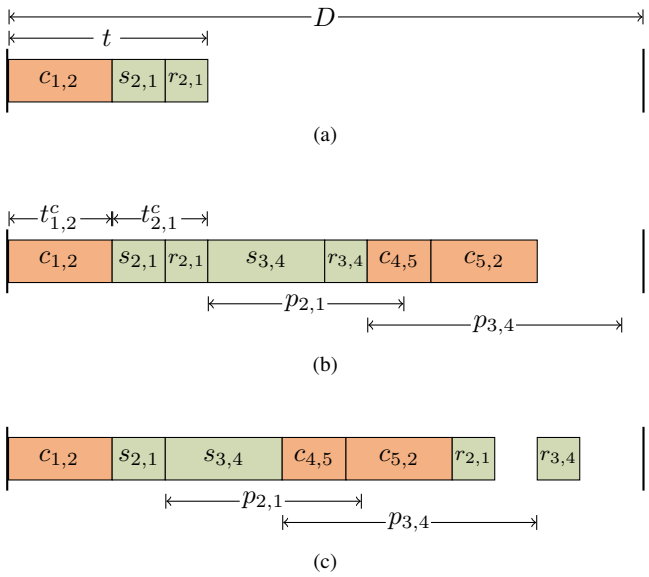


Fig. 3: An example of five tasks showing the the parameters of the dynamic programming algorithm and the main steps.

E. Version Selection Algorithm

Consider the sub-problem of the first i tasks $\{\tau_1, \tau_2, \ldots, \tau_i\}$. For the tasks in $\{\tau_1, \tau_2, \ldots, \tau_i\}$, let $\sum_{k=1}^{i} \sum_{j=1}^{m_k} v_{kj}(c_{kj} + s_{kj} + r_{kj})$ be the *total execution time of the local segments*, i.e. the local execution time of the local tasks and the summation of the sending and receiving times of the remote tasks. Suppose that $\mathcal{Q}(i,t)$ is the *maximum quality* for the tasks in $\{\tau_1, \tau_2, \ldots, \tau_i\}$, such that the total execution time of their local segments is less than or equal to t. Figure 3 shows the parameters of the algorithm using an example of five tasks. A dynamic programming table is constructed for all the possible integer values of i and t, where $0 \leq i \leq n$ and $0 \leq t \leq D$. We start by initializing all the elements of $Q(0,t)$ to zeros. Then, the following recursion is used to fill the table:

$$\mathcal{Q}(i,t) = \begin{cases} \max_{j:1,\ldots,m_i}\{q_{ij} + \mathcal{Q}(i-1, t-t^c_{ij})\} \\ \qquad\qquad\qquad\qquad \text{if } t^c_{ij} \leq t \leq D - p_{ij} \\ -\infty \qquad\qquad\qquad\qquad \text{otherwise.} \end{cases} \tag{2}$$

where $t^c_{ij} = c_{ij} + s_{ij} + r_{ij}$ is the local execution segment of the task τ_{ij}. The algorithm chooses the execution version of task τ_i that maximizes the total quality $Q(i,t)$ for each sub-problem $\{\tau_1, \tau_2, \ldots, \tau_i\}$, and for all the possible values of t. The local execution segment t^c_{ij} of the selected version is deducted from the available time t of the previous sub-problem as shown in Figure 3a. The inequation $t^c_{ij} \leq t \leq D - p_{ij}$ in the recursion maintains the feasibility conditions in Equations (1b) and (1c), where the part $t^c_{ij} \leq t$ checks if the local segment of the task can be executed within the deadline, and the part $t \leq D - p_{ij}$ checks if the result of the remote task can return before the deadline as shown in Figure 3b. The algorithm also maintains the selected version of each task for backtracking afterwards. After filling the table, we search for the maximum of $Q(n,t)$ for all the possible values of t, which represents the objective function in Equation (1a). Then, we backtrack the table starting from the location of the maximum value described above to the value of $\mathcal{Q}(i-1, t-t^c_{ij})$ in order to find the selected version for each task, where $i = n, n-1, \ldots, 2, 1$. Figure 3c illustrates the actual execution of the selected versions. Because the number of the versions is usually limited and constant, the time complexity of the dynamic programming algorithm is $O(nD)$. If any input parameter changes (e.g. the number of the tasks, the deadline, network bandwidth, the remote response time, etc), the algorithms can be easily executed again to adapt to the new parameters and to maintain system reliability.

IV. Experimental Evaluations and Simulations

The proposed algorithm was compared to two other algorithms, *Heuristic* and *Greedy* algorithms. In the Heuristic algorithm, the execution version j of a tasks τ_h is selected based on the highest Heuristic value of q_{hj}/t^c_{hj}. The algorithm tries to maximize the quality and minimize the local execution time simultaneously. In other words, it selects the versions with the highest possible qualities that do not consume too much time on the local device in order to meet the deadline. The Greedy algorithm selects the version of the task with the highest quality. If there is more than one version with the same highest quality, the algorithm selects the one with the minimum execution time. We use the abbreviation *DP* to denote our proposed Dynamic Programming algorithm.

A. Setup

The algorithms were evaluated using two ODROID-XU4 devices and a Thinkpad T420 computer. The two ODROID devices were used as a local device and as an edge node. The computer where used as a remote server on the cloud. ODROID-XU4 has a Samsung Exynos-5422 processor (Coretex-A15 Quad-Core 2 GHz and Cortex-A7 Quad-core 1.4 GHz), 2GB of RAM and a Mali-T628 MP6 GPU [16]. The same processor is used in Samsung Galaxy S5 mobile device as well [17]. The remote server has an Intel Core i5-2410M processor (2.30 GHz, 3MB L3), 8GB of RAM and Intel HD 3000 graphic chip. Ubuntu and Ubuntu Mate were used as operating systems on the the server and the ODROID devices respectively. The ODROID used as a local device was configured to run on a low-performance mode to emulate a resource-constrained device.

The following benchmarks were used for the the evaluation of the algorithms:

- Motion Detection and analysis (**MD**)
- Face Detection (**FD**)
- Digit Recognition (**DR**)
- Face Recognition (**FR**)
- Eye Detection (**ED**)
- Human Detection (**HD**)
- Car Detection (**CD**)

We executed the benchmarks using OpenCV library (Open Source Computer Vision Library), which is an open source library that includes many algorithms for computer vision, machine learning and image processing techniques [18]. The databases and files below were used as inputs for the benchmarks:

- Yale Face Database: It was used for face detection, eye detection and face recognition. The database contains 165 grayscale images of 15 people in GIF format. Each person has 11 images with different facial expressions and light configurations [19].
- Heatmap video: An application called motion heatmap [20] was used to detect and analyze movement patterns over time using this video [21]. The same video was also used for human detection. It has a resolution of 768×576 pixels, 10 frames per second and a total length of 1 minute and 19 seconds.
- GRAZ-02 database: A video from this database was used for the car detection benchmark. It has a length of 33 seconds and 204×240 pixels.
- OpenCV example images: They were used for digit recognition. The used images contains 5000 handwritten numbers segmented into 5000 single frames [22].

To generate the execution versions, the resolution of the input images were reduced from 100% to 5% with a step size of 5%. The evaluation was performed by combining the four modes below:

- **Mode 1** (Only local execution): The usage of edge computing is not possible due to problems in network connection or server availability.
- **Mode 2** (Local execution with edge computing): Edge computing is used to improve the quality or reduce the workload of the client.
- **Mode 3** (Normal execution without timing constraints).
- **Mode 4** (Task execution within a given deadline). The deadline was assigned according to the following equation:

$$D = \lambda \cdot \sum_{i=1}^{n} \max_{j:1,\ldots,m_i} \{c_{ij}\}, \tag{3}$$

where $\lambda = [1, 0.1]$ with a step size of 0.1. λ is the percentage of the deadline from the total longest execution time of the tasks if they are executed locally. Lower λ values correspond to tighter deadlines.

B. Results

Total quality of a task set is the summation of the results' quality for all the tasks in the set. *Total execution time* of a task set is the summation of the execution time for all the tasks on the local device. Figure 4 shows the average total quality and the average total execution time for all the algorithms by executing the tasks without any timing constraints. The DP and the Greedy algorithms have the same behavior, where they were able to achieve the maximum quality. Although the tasks selected by the Heuristic algorithm have relatively lower total quality (88%), their total execution time is just about 29% of the tasks selected by other two algorithms. This is because the Heuristic algorithm tries to balance between maximizing the quality and minimizing the total local execution time.

The algorithms were also evaluated under timing constraints, where a deadline of 1448.8 ms ($\lambda = 0.8$) was assigned. The results are shown in Figure 5. The DP algorithm was able to adapt to the new time frame by sacrificing only about 2% points of the total quality. The decisions of the Heuristic and Greedy algorithms are fixed, because they don't consider the available time frame. It was not possible to execute the tasks based on the decision of the Greedy algorithm, because their total execution time is longer than the deadline. The solution of the Heuristic algorithm is feasible under the assigned deadline, but the total quality of this solution is 88% which is less than the quality of the DP solution. To have a closer look, the qualities of the selected tasks are shown in Figure 6. The DP algorithm executes just one task, the face detection task, with a lower quality to leave enough space for the remaining tasks. The decision of the Heuristic algorithm is taken locally for each task. Therefore, a wrong decision for the first two tasks affected the quality of the remaining ones. The greedy algorithm was not able to execute the last two tasks at all, because the first five tasks were executed with the maximum quality.

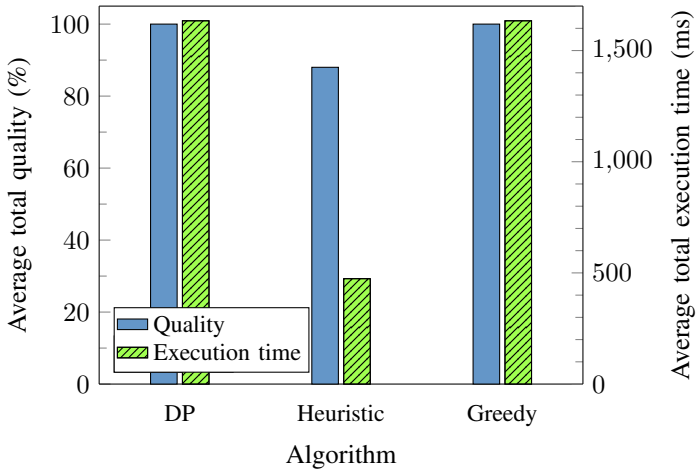


Fig. 4: Average total quality and total execution time of the tasks in case of only local execution and without timing constraints.

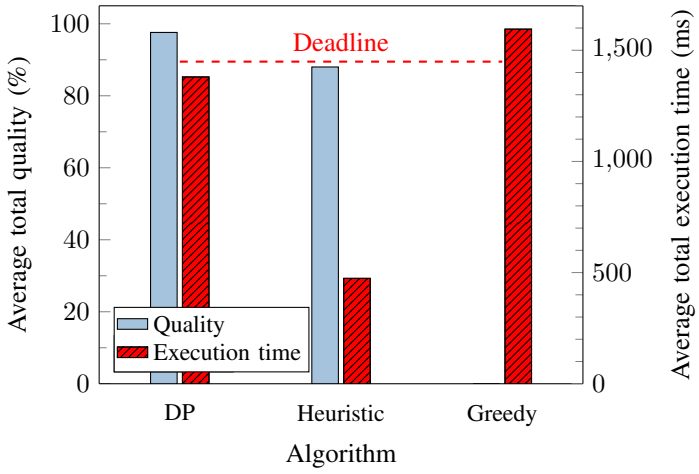


Fig. 5: Average total quality and total execution time of the tasks in case of only local execution and $\lambda = 0.8$.

To show the advantage of using edge computing, we compare it to the case where the tasks are executed only on the local device using different deadlines as presented in Figure 7.

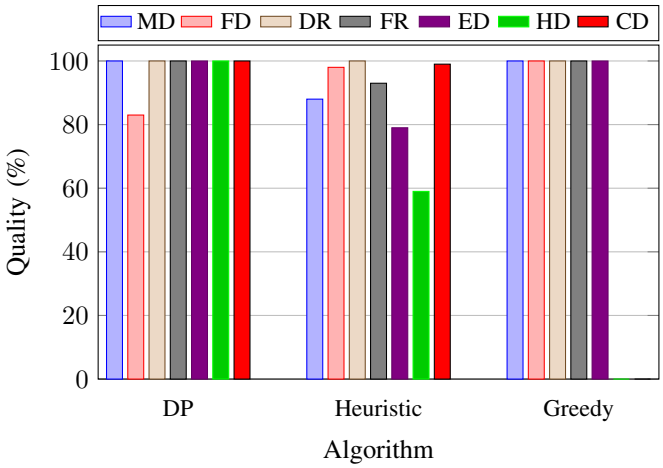


Fig. 6: Quality of the tasks for $\lambda = 0.8$ in case of only local execution.

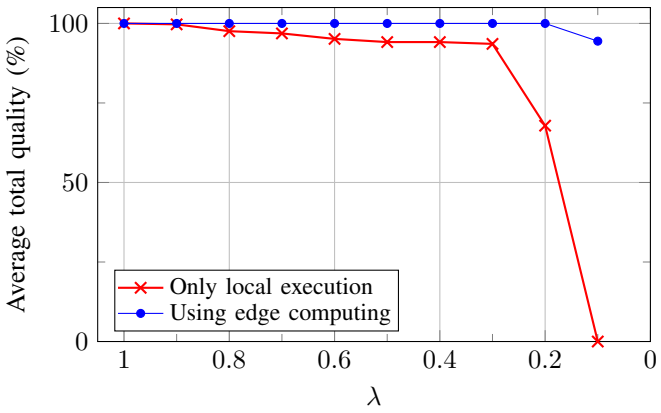


Fig. 7: Average total quality for different λ values using DP algorithm.

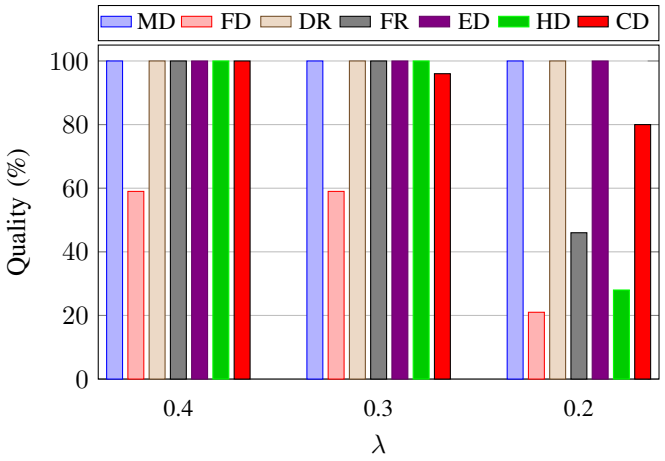


Fig. 8: Quality of the tasks using DP algorithm in case of only local execution.

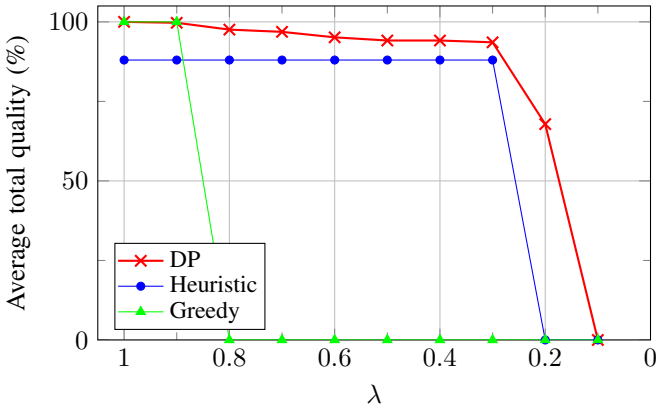


Fig. 9: Average total quality for different λ values in case of only local execution.

It is clear that the total quality in both cases decreases as the deadline decreases. However, the quality is higher in case of using edge computing than only executing the tasks on the local device, especially for tighter deadlines, because the edge nodes usually provide higher execution capabilities. The DP algorithm was not able to find a solution for a deadline of 181 ms ($\lambda = 0.1$) without using edge computing. The adaptation of the DP algorithm to the variation in the timing constraints is shown in Figure 8.

Figure 9 presents a comparison between the three algorithms for different deadlines in case of only local execution. The total quality of the solutions provided by the Greedy algorithm is equal to 100% for $\lambda = 1$ and $\lambda = 0.9$. Otherwise, the quality is degraded because the available time is not enough to execute all the tasks with the maximum quality. Although the solution provided by the Heuristic algorithm has a relatively lower quality of 88%, it was able to finish executing all the tasks within tight deadlines (until $\lambda = 0.3$). However, the solutions provided by the DP algorithm achieve quality values that are always higher than or equal to the ones achieved by other algorithms. Figure 10 presents the same comparison but with edge computing. The Greedy algorithm competes the DP algorithms until $\lambda = 0.3$. This is due to the fact that the edge computing nodes usually provide higher computation capabilities, where the execution time on these nodes is much shorter than the execution time on the local device. If the difference in the execution time between the local device and the edge nodes is big, the algorithm tends to execute the tasks remotely. Hence, the majority of the workload on the local device will be sending the data to the remote server and receiving the results from it, which is relatively not that long. The Heuristic algorithm was able to execute all the tasks even within very short deadlines, but with a low quality of about 70%. The figure shows that the DP algorithm was able to execute all the tasks with high quality for any deadline in the evaluation, which increases the reliability of the system.

V. Conclusion

This paper addresses the problem of executing computation-intensive tasks on resource-constrained devices such as portable medical devices, surveillance video cameras, wearable systems, etc. We consider computer vision tasks specifically in this paper, because a small reduction in the quality of their results is usually affordable. This property is

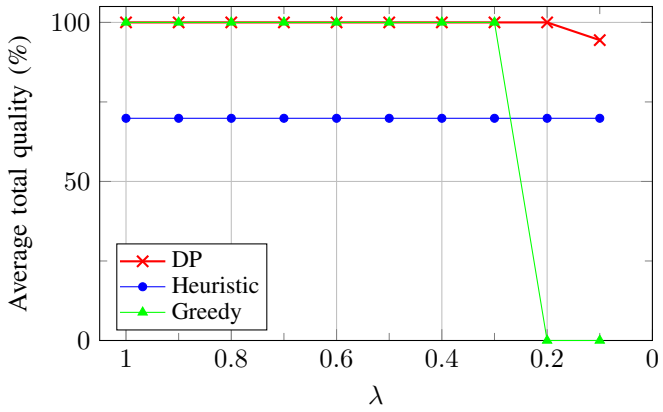


Fig. 10: Average total quality for different λ values in case of edge computing.

used to generate various execution versions of each task with quality and performance trade-offs. We adopt edge computing to improve the quality of the tasks or reduce the workload of the resource-constrained device. An algorithm based on dynamic programming is proposed to select the execution versions that maximize the quality under timing and resource constraints. This also includes determining the execution location of each task, i.e., on the local device or on one of the edge nodes. The evaluation shows that the execution time of the tasks can be reduced significantly by sacrificing only a small amount of quality. Moreover, the proposed algorithm provides adaptive task execution by selecting the suitable execution versions based on the available resources, which improves the reliability of the system.

VI. Acknowledgements

This work is supported by the German Research Foundation (DFG) as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/), subproject B2.

References

[1] Kenji Kanai, Kentaro Imagane, and Jiro Katto. Overview of multimedia mobile edge computing. *ITE Transactions on Media Technology and Applications*, 6(1):46–52, 2018.

[2] Pavel Korshunov and Wei Tsang Ooi. Video quality for face detection, recognition, and tracking. *ACM Trans. Multimedia Comput. Commun. Appl.*, 7(3):14:1–14:21, September 2011.

[3] Jan Eric Lenssen, Anas Toma, Albert Seebold, Victoria Shpacovitch, Pascal Libuschewski, Frank Weichert, Jian-Jia Chen, and Roland Hergenröder. Real-time low snr signal processing for nanoparticle analysis with deep neural networks. In *the 11th International Conference on Bio-Inspired Systems and Signal Processing (BIOSIGNALS 2018)*, Funchal, Portugal, January 2018.

[4] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018.

[5] Kashif Bilal, Osman Khalid, Aiman Erbad, and Samee U Khan. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130:94–120, 2018.

[6] Cisco. Cisco global cloud index: Forecast and methodology, 2016-2021 white paper. November 2018.

[7] Cisco. Cisco visual networking index: Forecast and methodology, 20162021. September 2017.

[8] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., NJ, USA, 2006.

[9] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 155–168, New York, NY, USA, 2015. ACM.

[10] W. Hu and G. Cao. Quality-aware traffic offloading in wireless networks. *IEEE Transactions on Mobile Computing*, 16(11):3182–3195, Nov 2017.

[11] Anas Toma, Alexander Starinow, Jan Eric Lenssen, and Jian-Jia Chen. Saving energy for cloud applications in mobile devices using nearby resources. In *the 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2018)*, Cambridge, UK, March 2018.

[12] Anas Toma, Santiago Pagani, Jian-Jia Chen, Wolfgang Karl, and Jörg Henkel. An energy-efficient middleware for computation offloading in real-time embedded systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*, pages 228–237. IEEE, 2016.

[13] Jaehyun Park, Ganapati Bhat, Cemil Geyik, Hyung Gyu Lee, and Umit Ogras. Energy-optimal gesture recognition using self-powered wearable devices. In *Biomedical Circuits and Systems Conference (BioCAS 2018)*, Cleveland, USA, October 2018.

[14] Kristin Krüger, Marcus Völp, and Gerhard Fohler. Vulnerability Analysis and Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Systems. In Sebastian Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[15] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[16] Hardkernel. ODROID-XU4. http://www.hardkernel.com/main/products/prdt_info.php.

[17] Samsung. Exynos 5 octa (5422) mobile processor. http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_5_Octa_5422.html.

[18] Gary R. Bradski and Adrian Kaehler. *Learning OpenCV - computer vision with the OpenCV library: software that sees*. O'Reilly Media Inc., 2008.

[19] Department of Computer Science, Yale University. Yale Face Database. http://cvc.cs.yale.edu/cvc/projects/yalefaces/yalefaces.html.

[20] Intel Corporation. Motion heatmap, . https://github.com/intel-iot-devkit/python-cv-samples/tree/master/examples/motion-heatmap.

[21] Motion heatmap video. https://github.com/opencv/opencv/blob/master/samples/data/vtest.avi.

[22] Intel Corporation. Open Source Computer Vision Library (OpenCV), . https://opencv.org.