Deep Learning Video Analytics Through Edge Computing and Neural Processing Units on Mobile Devices

Tianxiang Tan, *Student Member, IEEE,* and Guohong Cao, *Fellow, IEEE,*

Abstract—Many mobile applications have been developed to apply deep learning for video analytics. Although these advanced deep learning models can provide us with better results, they also suffer from the high computational overhead which means longer delay and more energy consumption when running on mobile devices. To address this issue, we propose a framework called FastVA, which supports deep learning video analytics through edge processing and Neural Processing Unit (NPU) in mobile. The major challenge is to determine when to offload the computation and when to use NPU. Based on the processing time and accuracy requirement of the mobile application, we study three problems: *Max-Accuracy* where the goal is to maximize the accuracy under some time constraints, *Max-Utility* where the goal is to maximize the utility which is a weighted function of processing time and accuracy, and *Min-Energy* where the goal is to minimize the energy under some time and accuracy constraints. We formulate them as integer programming problems and propose heuristics based solutions. We have implemented FastVA on smartphones and demonstrated its effectiveness through extensive evaluations.

Index Terms—deep learning, video analytics, mobile computing

1 Introduction

Deep learning techniques, such as Convolutional Neural Network (CNN), have been successfully applied to various computer vision and natural language processing problems, and some of the advanced models can even outperform human beings in some specific datasets [1]. Over the past few years, many mobile applications have been developed to apply CNN models for video analytics. For example, Samsung Bixby can help users extract texts showing up in the video frames. Although these advanced CNN models can provide us with better results, they also suffer from the high computational overhead which means long delay and more energy consumption when running on mobile devices.

Most existing techniques [2]–[6] address this problem through computation offloading. By offloading data/video to the edge server and letting the edge server run these deep learning models, energy and processing time can be saved. However, this is under the assumption of good network condition and small input data size. In many cases, when the network condition is poor or for applications such as video analytics where a large amount of data is processed, offloading may take longer time, and thus may not be the best option.

Recently, many companies such as Huawei, Qualcomm, and Samsung are developing dedicated *Neural Processing Units (NPUs)* for mobile devices, which can process AI features. With NPU, the running time of these deep learning models can be significantly reduced. For example, the processing time of the ResNet-50 model [1] is about one second using CPU, but only takes about 50 ms with NPU. Although NPUs are limited to advanced phone models at this time, this technique has great potential to be applied to other mobile devices, and even for IoT devices in the future. Although GPUs on mobile devices can also be used to run DNNs, they are not as powerful as those running on desktops and servers. Compared to GPUs on mobile devices, NPUs are much faster and more energy efficient [7].

There are some limitations with NPU. First, NPU uses 16 bits or 8 bits to represent the floating-point numbers instead of 32 bits in CPU. As a result, it runs CNN models much faster but less accurate compared to CPU. Second, NPU has its own memory space and sometimes the CNN models are too large to be loaded into memory. Then, the CNN model has to be compressed in order to be loaded by NPU and then reducing the accuracy. For instance, HUAWEI mate 10 pro has limited memory space for NPU and many advanced CNN models must be compressed at the cost of accuracy.

There is a tradeoff between the offloading based approach and the NPU based approach. Offloading based approach has good accuracy, but may have longer delay under poor network condition. On the other hand, NPU based approach can be faster, but with less accuracy. In this paper, we propose FastVA, a framework that combines these two approaches for real time video analytics on mobile devices. The major challenge is to determine when to offload the computation and when to use NPU based on the network condition, the video processing time, and the accuracy requirement of the application.

Consider an example of a flying drone. The camera on the drone is taking videos which are processed in real time to detect nearby objects to avoid crashing into a building or being trapped by a tree. To ensure no object is missed, the detection result should be as accurate as possible. Here,

• *The authors are with School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA 16802. E-mail: {txt51, gxc27}@psu.edu.*

the time constraint is critical and we should maximize the detection accuracy under such time constraint. For many other mobile applications such as unlocking a smartphone, making a payment through face recognition, or using Google glasses to enhance user experience by recognizing the objects or landmarks and showing related information, accuracy and processing time are both important. Hence, we should achieve a better tradeoff between them. Since mobile devices are powered by battery, reducing energy consumption is critical and we should minimize the energy consumption under time and accuracy constraints.

Based on the accuracy, the processing time and the accuracy requirement of the mobile application, we study three problems: *Max-Accuracy* where the goal is to maximize the accuracy under some time constraints, *Max-Utility* where the goal is to maximize the utility which is a weighted function of accuracy and processing time, and *Min-Energy* where the goal is to minimize the energy under the time and accuracy constraints. To solve these three problems, we have to determine when to offload the computation and when to use NPU. The solution depends on the network condition, the special characteristics of NPU, and the optimization goal. We will formulate them as integer programming problems, and propose heuristics based solutions.

Our contributions are summarized as follows.

- We study the benefits and limitations of using NPU to run CNN models to better understand the characteristics of NPU in mobile.
- We formulate the Max-Accuracy problem for the applications with strict time constraints, and propose a heuristic based solution.
- We formulate the Max-Utility problem to achieve a better tradeoff between accuracy and processing time, and propose an approximation based solution.
- We identify the power, accuracy and processing time tradeoffs between computation offloading based approach and NPU-based approach, formulate and solve the Min-Energy problem.
- We implement FastVA on smartphones and compare it with other techniques through extensive evaluations.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 studies the benefits and limitation of NPU and provides an overview of FastVA. We formulate the *Max-Accuracy Problem* and propose a solution in Section 4. In Section 5, we study the *Max-Utility Problem* and give a heuristic based solution. In Section 6, we study the *Min-Energy Problem* and propose a solution. Section 7 presents the evaluation results and Section 8 concludes the paper.

2 Related Work

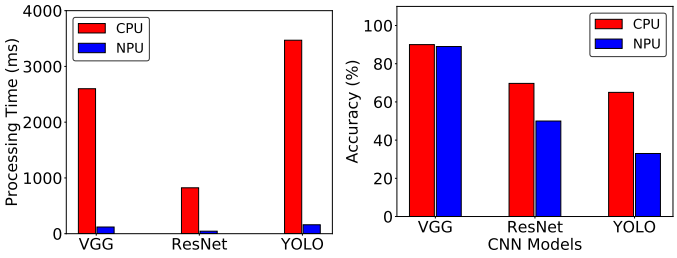
Over past years, there have been significant advances on object recognition with CNN models. For example, GoogleNet [8] and ResNet [9] can achieve high accuracy. However, these CNN models are designed for machines with powerful CPU and GPU, and it is hard to run them on mobile devices due to limited memory space and computation power. To address this issue, various model compression techniques have been developed. For example, in [10], the authors propose to separate convolutional kernels from the convolutional layers and compress the fully-connected layers to reduce the processing time of CNN models. Liu *et al.* [11] optimize the convolutional operations by reducing the redundant parameters in the neural network. FastDeepIoT [12] compresses the CNN models by optimizing the neural network configuration based on the non-linear relationship between the model architecture and the processing time. In [13]–[15], the authors reduce the size and the processing time of CNN models by using less number of bits to represent the parameters in these CNN models. Although the efficiency can be improved through these model compression techniques, the accuracy also drops.

Offloading techniques have been widely used to address the resource limitation of mobile devices. MAUI [16] and many other works [17]–[23] are general offloading frameworks that optimize the energy usage and the computation overhead for mobile applications. However, these techniques have limitations when applied to video analytics where a large amount of video data has to be uploaded to the server. To address this issue, researchers propose offloading framework for deep learning applications. Teerapittayanon et al. [4] distribute the CNN model computations across local device, edge server and cloud, and insert early exit points to reduce the processing time. In Neruosurgeon [24], the CNNs are divided into local processing part and offloading part to reduce the latency and energy consumption. Lu *et al.* [25] proposed a framework which optimizes the video processing by combining the CNN batch processing and offloading technique. Tan *et al.* [26] designed algorithms which can combine offloading and local processing to minimize the processing time for video analytics using multiple CNN models. To reduce the data offloading time, some local processing techniques have been proposed to filter out the less important or redundant data. For example, Glimpse [2] only offloads a frame when the system detects that the scene changes significantly. Similar to Glimpse, MARVEL [3] utilizes the inertial sensors to detect and filter out the redundancy before offloading. However, both MARVEL and Glimpse may not work well when the network condition is poor or when there are many scene changes. To address this issue, other researchers consider how to guarantee a strict time constraint by running different CNN models locally under different network conditions [5], [6].

Some recent work focuses on improving the execution efficiency of CNN models on mobile devices through hardware support. For example, Tan *et al.* [27] developed model partitioning techniques to schedule some neural network layers on CPU while executing other layers on NPU to achieve better tradeoffs between processing time and accuracy Oskouei *et al.* [28] developed an Android library called CNNdroid for running CNN models on mobile GPU. Cappuccino [29] optimizes computation by exploiting imprecise computation on the mobile system-on-chip (SoC). DeepMon [30] leverages GPU for continuous vision analysis on mobile devices. DeepX [31] divides the CNN models into different blocks which can be efficiently run on CPU or GPU. The processing time is reduced by scheduling these blocks on different processors, such as CPU and GPU. Different from

them, we use NPUs.

3 PRELIMINARY



(a) Processing Time Comparison (b) Accuracy Comparison

Fig. 1: Performance Comparison of NPU and CPU.

A video frame can be processed locally by the NPU or offloaded to the server. The decision depends on the application requirements on the tradeoff among accuracy, energy consumption and processing time. More importantly, the decision depends on how various CNN models perform on NPUs in terms of accuracy, processing time and power consumption. In this section, we first show some results on how various CNN models perform on NPUs and local CPUs, and then we give an overview of the proposed FastVA framework.

3.1 Understanding NPU

To have a better understanding of NPU, we compare the accuracy and the processing time of running different CNN models on NPU and CPU. The experiment is conducted on HUAWEI mate 10 pro which has a NPU, and the results are shown in Figure 1. Three CNN models are used in the evaluations and the details are as follows.

- The VGG model [32] which is used for face recognition. In the experiment, we use the face images from the LFW dataset [33].
- The ResNet-50 model [9] which is used for object recognition. The evaluation was based on 4000 object images randomly chosen from the VOC dataset [34], and the results were based on the Top-1 accuracy.
- The YOLO Small model [35] which is designed for detecting objects in an image. The evaluation was based on 100 images randomly chosen from the COCO dataset [36], and the results were based on the F1-score.

As shown in Figure 1(a), compared to CPU, running VGG, ResNet-50, or Yolo small on NPU can significantly reduce the processing time, by 95%. As shown in Figure 1(b), the accuracy loss of using NPU is different for different CNN models. For example, compared to CPU, using NPU has similar accuracy when running VGC, 20% accuracy loss when running ResNet, and the F1-score drops to 0.3 when running YOLO Small.

The accuracy loss is mainly because NPU can only support FP16 operations and store the intermediate result of each layer using FP16, which may result in numerical instability. The numerical instability is caused by floating point overflow or underflow. For some CNN models, the result may even be NaN or 0, which is impossible to interpret. On the other hand, using FP16 in NPU helps achieve an order of magnitude speed and energy improvement with limited memory space.

The accuracy loss also depends on the CNN models. VGG compares the similarity between the two feature vectors $[x_1, x_2, \ldots, x_n]$ and $[y_1, y_2, \ldots, y_n]$, which are extracted from the face images. They belong to the same person if the similarity is below a predefined threshold. The similarity can be measured by the square of the Euclidean distance between two vectors, where $d = \sum_{i=1}^{n}(x_i - y_i)^2$. Since NPU uses less number of bits to represent the floating point numbers, some errors will be introduced. The error introduced by NPU changes d to $d' = \sum_{i=1}^{n}(x_i - y_i + \epsilon_i)^2$. If we consider ϵ_i as noise data with mean value 0, the expected value of d' is equal to $E(d) + \sum_i E(\epsilon_i^2)$. Since ϵ_i is small, it will not change the relationship between d and the threshold too much for most input data, and hence has the same level of accuracy as CPU.

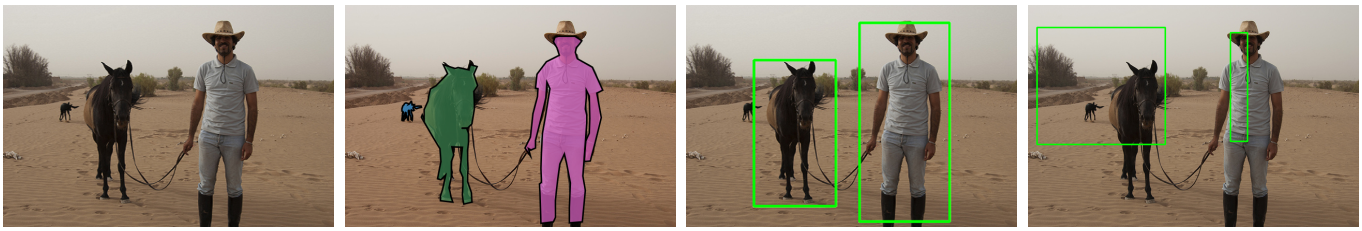
Suppose the feature vector extracted from an image is $\vec{f_1} = [x_1, x_2, \ldots x_n]$, ResNet-50 classifies the image based on the largest element. Assume x_p and x_q are the two largest elements in $\vec{f_1}$, and $x_p > x_q$. Then, the image will be classified as the p^{th} category image. The errors introduced by NPU may change the elements to $x'_p = x_p + \epsilon_1$ and $x'_q = x_q + \epsilon_2$. If the difference between x_p and x_q is small, x'_q may be larger than x'_p due to errors ϵ_1, ϵ_2. Then, the object will be classified as the q^{th} category, and getting a wrong result.

YOLO Small is much more complex than ResNet and VGG. Its feature vector includes information related to location, category and size of the objects, and a small error in the feature vector can change the result. For instance, Figure 2 shows the detection result of a test image using CPU and NPU. Figure 2(c) shows the result with CPU, where the bounding boxes accurately include the objects, and the objects can be correctly recognized as horse and person respectively. Figure 2(d) shows the result with NPU. As can be seen, the detection result only includes part of the objects. The center and the size of the objects are incorrect, leading to wrong detection results.

From these evaluations, we can see that NPU runs much faster than CPU; however, it may not always be the best choice for running CNN models, especially when accuracy is important.

3.2 FastVA Overview

The overview of FastVA is shown in Figure 3. FastVA can process frames through offloading or local processing with NPU. For offloading, FastVA may reduce the frame resolution before transmitting so that more frames can be uploaded at the cost of accuracy. Similarly, multiple CNN models can be used, where a smaller CNN model can reduce the processing time and energy consumption at the cost of accuracy, and a larger model can increase the accuracy at the cost of longer processing time and higher energy consumption. If several CNN models are available, FastVA will choose the proper one for processing under different constraints. To provide real time video analytics, FastVA



(a) The Original Image (b) Ground Truth (c) Detection Result on CPU (d) Detection Result on NPU

Fig. 2: Detection result with YOLO Small

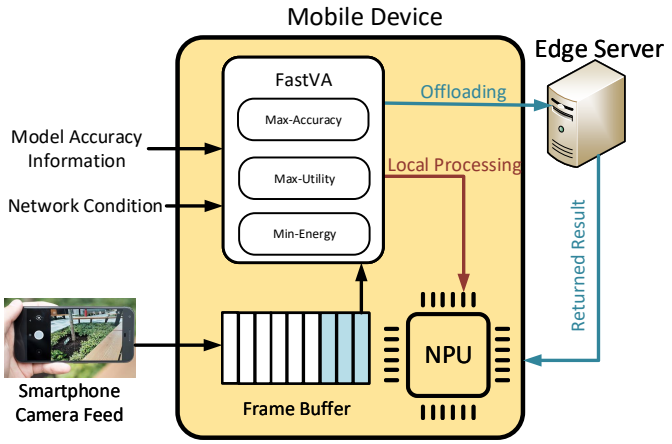


Fig. 3: FastVA overview

ensures that the processing of each video frame is completed within a time constraint.

Different applications have different requirements on accuracy, processing time, and energy. Since mobile devices only have limited resources, i.e., computational power, wireless bandwidth, and battery power, FastVA should be carefully designed to satisfy these application requirements under these resource constraints. In this paper, we study three optimization problems: Max-Accuracy, Max-Utility, and Min-Energy. For AR-based navigation or flying drones which have real time requirements, processing time is more important. Hence, we study the Max-Accuracy problem, where the goal is to maximize accuracy under some time constraint. For applications such as unlocking a smartphone or making payment through face recognition, accuracy is more important than processing time. Hence, we study the Max-Utility problem, where the goal is to maximize the utility which is a weighted function of accuracy and video processing time. For applications which need to run for a long time and recharging is not possible in the near future, energy consumption is more important. Hence, we study a Min-Energy problem, where the goal is to minimize the energy consumption under accuracy and processing time constraints. To solve these three problems, FastVA has to determine when to offload the computation and when to use NPU. The solution depends on the network condition, the special characteristics of NPU, and the optimization goal. In the following sections, we formulate and solve these three optimization problems.

4 The Max-Accuracy Problem

In this section, we study the Max-Accuracy problem which aims to maximize the accuracy under some time constraints. We first formulate the problem and then propose a heuristic based solution.

4.1 Problem Formulation

Notation	Description
I_i	the ith frame
$S(I_i, r)$	the data size of the frame I_i in resolution r
T_j^{npu}	The processing time of j^{th} model on NPU
T_j^o	Processing time using the j^{th} model on the server
$a(j, r)$	The accuracy of the j^{th} model with input images in resolution r
L_i	Network latency of sending frame i between mobile device and server
B_i	upload bandwidth (data rate) of sending frame I_i
f	video frame rate (fps)
γ	the time interval between two consecutive frames
T	the time constraint for each frame
n	the number of video frames that needs to be processed

TABLE 1: Notation.

Assume the incoming frame rate is f, the time interval between two consecutive video frames is $\gamma = \frac{1}{f}$. For the i^{th} frame in the video, assume its arrival time is $i\gamma$ and FastVA needs to process it before $T + i\gamma$, where T is the time constraint. For each frame, it can either be processed locally by the NPU or offloaded to the server. Multiple CNN models are used on the edge server and the mobile device to process these frames. If the frame I_i is processed by the j^{th} model locally, the corresponding processing time is T_j^{npu}. If I_i is processed at the edge server, the data can be offloaded with the original resolution or reduce the resolution to r before uploading to save bandwidth. Let B_i denote the upload bandwidth of sending I_i and let L_i denote the network latency between the edge server and the mobile device. Then, it takes $\frac{S(I_i,r)}{B_i} + T_j^o + L_i$ to transmit the i^{th} frame in resolution r and receive the result from the server. Although the transmission time can be reduced by reducing the frame to a lower resolution, the accuracy is lower.

The notations used in the problem formulation and the algorithm design are shown in Table 1. The Max-Accuracy problem can be formulated as an integer programming in the following way.

$$\max \quad \frac{1}{n}\sum_{i=0}^{n}\sum_{j}\sum_{r} a(j,r)X_i^j Y_i^r \tag{1}$$

$$\text{s.t.} \quad \sum_{k\le i}\sum_{j} T_j^{npu} X_k^j \le T + (i-k)*\gamma,\ \forall i,k \tag{2}$$

$$D(k) \le T + (i-k)*\gamma,\ \forall i,k \tag{3}$$

$$\sum_{j} X_i^j = 1,\ \forall i \tag{4}$$

$$\sum_{r} Y_i^r = 1,\ \forall i \tag{5}$$

$$Y_i^r, X_i^j \in \{0,1\}\ \forall i,j \tag{6}$$

Where $D(k) = \sum_j (\sum_r \sum_{k\le i} \frac{S(k,r)Y_k^r}{B_k} + T_j^o X_k^j) + L_i$ is the offloading time for the frames that arrive between I_k and I_i. X_i^j is a variable to show which model is used to process the frame and Y_i^r is a variable to show which resolution the frame is resized to before offloading. If $X_i^j = 0$, the frame I_i is not processed by the j^{th} model. If $X_i^j = 1$, the frame I_i is run by the j^{th} model. If $Y_i^r = 1$, the frame I_i is resized to resolution r before offloading.

Objective (1) is to maximize the accuracy of the processed frames in the time window. Constraint (2) specifies that all local processed frames should be completed before the deadline, and constraint (3) specifies that the results of the offloaded frames should be returned within the time constraint.

Theorem 1. *The Max-Accuracy problem is NP-hard.*

Proof. We reduce a well known NP-hard problem, the unbounded knapsack problem to the Max-Accuracy problem. In the unbounded knapsack problem, there are n items $(p_1, p_2, p_3, \ldots, p_n)$, each item p_i with weight w_i and a value v_i. Given a knapsack with a weight limit W, the goal is to find a subset of items such that their total weight is no more than W and their total value is maximized. Different from the 0-1 knapsack problem, we can pick unlimited copies of an item. For example, p_1 can only be picked at most once in the 0-1 knapsack problem, but it can be picked multiple times in the unbounded knapsack problem.

For an arbitrary instance of knapsack problem, we can construct an instance of our Max-Accuracy problem as follows. We first construct a video clip with n' frames, where $n' = \lceil \max_i \frac{W}{w_i} \rceil$. For each frame, it can only be offloaded to the server for processing. There are $(n+1)$ possible resolution options and the data size of the frame I_k is defined as

$$S(I_k, r_i) = \begin{cases} 0 & \text{if } i = 0 \\ Bw_i & \text{if } i \in [1,n] \end{cases}$$

All frames arrive at the frame buffer at time 0, and $L_i, T_{j_0}^o$ are set to be 0. The time constraint T is set to be W. There is only one CNN model j_0 running on the server and its accuracy is defined as

$$a(j_0, r_i) = \begin{cases} 0 & \text{if } i = 0 \\ v_i & \text{if } i \in [1,n] \end{cases}$$

A solution to this instance of Max-Accuracy problem maximizes $\sum_{k=1}^{n'} \sum_{i=1}^{n} \sum_j a(j, r_i) X_k^j Y_k^{r_i}$. Since there is only one model j_0 is used, $\sum_j X_k^j = X_k^{j_0} = 1(k \in [1, n'])$. $\sum_{k=1}^{n'} Y_k^{r_i}$ can be seen as the number of frames which are offloaded in resolution r_i. Let $Z_i = \sum_{k=1}^{n'} Y_k^{r_i}$, the Max-Accuracy actually maximizes $\sum_{i=1}^{n} v_i Z_i$, which is the goal of unbounded knapsack problem. Moreover, the time constraint must be satisfied, we have

$$\sum_{k=1}^{n'} \sum_{i=1}^{n} \frac{S(I_k, r_i)}{B_k} Y_k^{r_i} \le T$$

$$\sum_{i=1}^{n} w_i Z_i \le W$$

Which is also the weight constraint in the unbounded knapsack problem. Thus, the solution is also a solution to the unbounded knapsack problem. This completes the reduction and hence the proof. □

4.2 Max-Accuracy Algorithm

Algorithm 1: Max-Accuracy Algorithm

Data: Video frames in the buffer
Result: Scheduling decision

The frame schedule list $S \leftarrow \{\}$
$A \leftarrow 0$, $n_s \leftarrow$ the number of models on the server side
for *each possible resolution* r **do**
  Resize the I_0 to the resolution r
  $A' \leftarrow 0, S' \leftarrow \{\}$
  Sort the remote models in the descending order based on their accuracy $a(j, r)$.
  for j *from 1 to* n_s **do**
    if $t + T_j^o + L_0 \le T$ **then**
      Add $(0, j, r)$ to S'
      $A' \leftarrow A' + a(j, r)$
      break
  $n_l \leftarrow \lfloor \frac{S(I_0, r)}{B_0 * \gamma} \rfloor$
  Compute $H(i,t)$ according to Equation 7 and 8 for $i \in [1, n_l]$ and $t \in [\gamma, n_l * \gamma + T]$
  $h' \leftarrow \max_t H(n_l, t)$
  $t' \leftarrow \arg\max_t H(n_l, t)$
  for i *from* n_l *to* 1 **do**
    for *each local model* j **do**
      if $H(i-1, t' - T_j^{npu}) = h'$ **then**
        Add (i, j, r_{max}) to S'
        $t' \leftarrow t' - T_j^{npu}, h' \leftarrow h' - a(j, r_{max}), A' \leftarrow A' + a(i, j)$
        break
  if $\frac{A'}{n_l+1} > A$ **then**
    $A \leftarrow \frac{A'}{n_l+1}, S \leftarrow S'$
return S

A brute force method to solve the Max Accuracy Problem is to try all the possible scheduling options, and it takes $O((n_c * n_r)^n)$, where n_c is the number of CNN models available for processing the frames and n_r is the number of resolution options. Since the brute force method is impractical, we propose a heuristic solution. The basic idea is as follows. Since offloading based approach can achieve better accuracy than NPU based approach for the same CNN model, the arriving video frame should be

offloaded as long as there is available bandwidth. Due to limited bandwidth, some frames cannot be offloaded and will be processed by the NPU locally. More specifically, our Max-Accuracy algorithm consists of multiple rounds. In each round, there are two phases: offload scheduling phase and local scheduling phase. In both phases, the right CNN model is selected to process the video frame within the time constraint and maximize the accuracy.

4.2.1 Offload Scheduling

In this phase, the goal is to find out the CNN model that can be used for processing the offloaded video frame within time constraint and maximize the accuracy. Assume that the network interface is idle and I_0 is the new frame arriving at the buffer. I_0 will be resized to resolution r and offloaded to the server. On the server side, the only requirement for selecting the CNN model is the time constraint, which requires the result must be returned in time. In other words, the constraint $\frac{S(I_0,r)}{B_0} + L_0 + T_j^o \leq T$ must be satisfied for the uploaded frame I_0. A CNN model will be selected if it can satisfy the time constraint and has the highest accuracy on images with resolution r. Since video frames arrive at a certain interval γ, $n_l = \lfloor \frac{S(I_0,r)}{B_0 * \gamma} \rfloor$ frames will be buffered while I_0 is being transmitted. These frames will be processed locally, and the local scheduling phase will be used to determine their optimal scheduling decision.

4.2.2 Local Scheduling

In this phase, the goal is to find out the CNN model that can be used for processing the video frame within time constraint and maximize the accuracy. For each CNN model, the video processing time and the accuracy vary. A simple dynamic programming algorithm is used to find an optimal scheduling decision. More specifically, let $H(k,t)$ denote the optimal accuracy for processing the first k frames with time constraint t, where $k \in [0, n_l]$. Then, frame I_1 arrives at the frame buffer at time γ and the last frame I_{n_l} must be processed before time $n_l * \gamma + T$, thus $t \in [\gamma, n_l * \gamma + T]$. If it is impossible to process all k frames within t, $H(k,t) = -\infty$. Initially, since the frame I_0 is offloaded to the server, $H(0,t)$ can be computed as follow:

$$H(0,t) = \begin{cases} -\infty, & \text{if } t < T^{idle} \\ 0 & \text{Otherwise} \end{cases} \tag{7}$$

where T^{idle} is the queuing time for I_1.

For frame $I_k (k > 0)$, it can be processed on one of the local CNN model j. $H(k,t)$ can be computed as follow:

$$H(k,t) = \begin{cases} -\infty, & \text{if } \forall j, k\gamma + T_j^{npu} < t \\ \max_j(H(k-1, t - T_j^{npu}) + a(j, r_{max})), & \text{Otherwise} \end{cases} \tag{8}$$

Theorem 2. *The proposed dynamic programming algorithm has an optimal substructure.*

Proof. Let $H^*(n_l, t)$ denote the optimal accuracy for processing the n_l frames with time constraint t. To show the optimal substructure property, we prove the substructure $H^*(k', t')$, where $k' < n_l, t' < t$, is optimal for the subproblems.

Let A^* denote the optimal accuracy that can be achieved by processing the frames $I_{k'+1}, I_{k'+2}, \dots I_{n_l}$ within time $t - t'$. $H^*(n_l, t)$ can be represented as $H^*(k', t') + A^*$. Assume that the substructure $H^*(k', t')$ is not optimal which means that there exists $H(k', t')$ so that $H^*(k', t') < H(k', t')$. Then, we have

$$\begin{aligned} H^*(n_l, t) &= H^*(k', t') + A^* \\ &< H(k', t') + A^* \\ &= H(n_l, t) \end{aligned}$$

Which contradicts the assumption that $H^*(n_l, t)$ is optimal (i.e., has the maximum accuracy). □

Based on the computed $H(k,t)$, the scheduling decision can be made by backtracking. The Max-Accuracy algorithm is summarized in Algorithm 1. Lines 4-11 are the offloading scheduling phase and Lines 12-21 are for the local scheduling phase. In the algorithm, a variable A is used for tracking the maximum accuracy that is found so far, and its corresponding schedule decision is maintained in S'. The frame schedule list S' is a list of pair (i, j, r), which means that frame I_i is processed by the j^{th} model with resolution r. The running time of our algorithm is $O(n_r * n_c * n)$.

5 Max-Utility Problem

In this section, we study the Max-Utility problem. The goal is to maximize the utility which is a weighted function of accuracy and video processing time. We first formulate the problem and then propose an approximated based solution.

5.1 Problem Formulation

With time constraint, FastVA may not be able to process all frames using the CNN model with the highest accuracy. To achieve high accuracy with limited resources, FastVA may skip some frames whose queuing time is already close to its time constraint. With the notations used in the last section, the length of the video is $n\gamma$ and $\sum_i \sum_j X_i^j$ is the total number of frames to be processed. Then, the video is processed at a real frame rate of $\frac{\sum_i \sum_j X_i^j}{n\gamma}$. The average accuracy can be computed as $\frac{\sum_i \sum_j a(j,r) X_i^j Y_i^r}{\sum_i \sum_j X_i^j}$. Let α denote the tradeoff parameter between accuracy and processing time (measured with the frame processing rate). Then, the utility can be computed as $\sum_i \sum_j \frac{X_i^j}{n\gamma} + \alpha \frac{\sum_i \sum_j a(j,r) X_i^j Y_i^r}{\sum_i \sum_j X_i^j}$.

Similar to the Max-Accuracy problem, the Max-Utility Problem can be formulated as an integer programming in the following way.

$$\max \quad \sum_{i=0}^{n} \sum_j \frac{X_i^j}{n\gamma} + \alpha \frac{\sum_i \sum_j a(j,r) X_i^j Y_i^r}{\sum_i \sum_j X_i^j} \tag{9}$$

$$\text{s.t.} \quad \sum_{k \leq i} \sum_j T_j^{npu} X_k^j \leq T + (i-k) * \gamma, \ \forall i, k \tag{10}$$

$$D(k) \leq T + (i-k) * \gamma, \ \forall i, k \tag{11}$$

$$\sum_j X_i^j \leq 1, \ \forall i \tag{12}$$

$$\sum_r Y_i^r \leq \sum_j X_i^j, \forall i \tag{13}$$

$$Y_i^r, X_i^j \in \{0,1\}, \forall i,j \tag{14}$$

Objective (9) maximizes the utility. Constraints (10) and (11) specify that the frames must be processed within the time requirement. Constraint (13) specifies that each frame can at most be processed by a CNN model either remotely or locally. Constraint (14) specifies that each image can only be resized to a certain resolution.

Theorem 3. *The Max-Utility problem is NP-hard.*

Proof. We can reduce the Max-Accuracy problem to the Max-Utility problem. The major difference between Max-Accuracy and Max-Utility is the frame processing rate and the parameter α in the objective function. For an arbitrary instance of Max-Accuracy problem, we can select the parameter α properly so that the frame processing rate $\sum_i \sum_j \frac{X_i^j}{n\gamma}$ has no impact on the objective (9).

More specifically, given any two solutions of the Max-Utility problem S and S' with frame processing rate $f_S, f_{S'}$ and accuracy $A_S, A_{S'}$. If the frame processing rate has no impact on the objective (9) and $A_S > A_{S'}$, then we have

$$f_S + \alpha A_S > f_{S'} + \alpha A_{S'}$$
$$\alpha > \frac{f_{S'} - f_S}{A_S - A_{S'}}$$

Since $f_{S'}, f_S \in [1, f]$, $f_{S'} - f_S \leq f$. Let d to be $\min_{\forall j,j',r,r'} |a(j,r) - a(j',r')|$. $A_S - A_{S'} \geq d$. Therefore, we have

$$\alpha \geq \frac{f}{d} > \frac{f_{S'} - f_S}{A_S - A_{S'}}$$

In the Max-Utility problem, set α to be $\frac{f}{d} + 1$, the frame processing rate has no impact on objective (9). The optimal solution of the Max-Utility problem is actually maximizing the accuracy which is the goal of the Max-Accuracy problem. This completes the reduction and hence the proof. □

5.2 Max-Utility Algorithm

Since the problem is NP-hard, we propose a heuristic based algorithm (called the Max-Utility Algorithm) to solve it. The basic idea of the algorithm is as follows. Since the offloading based approach can achieve better accuracy than NPU based approach for the same CNN model, our algorithm first maximizes the utility by offloading the arriving video frame with the available bandwidth. Due to the limited bandwidth, some frames will not be offloaded and our Max-Utility algorithm further improves the utility using a dynamic programming algorithm to decide which frames should be skipped and which frames should be processed locally.

Assume the network interface is idle when a new frame I_0 arrives in the buffer. I_0 will be resized to a resolution r and offloaded to the server. The offloading time for this frame is $\frac{S(I_0,r)}{B_0}$, which means the frames are offloaded at the frame rate $\frac{B_0}{S(I_0,r)}$. The schedule decision for I_0 is made by solving $\max_{r,j} \frac{B_0}{S(I_0,r)} + \alpha \times a(j,r)$. Since the result from the server should be received within the time limitation, the constraint $T \geq \frac{S(I_0,r)}{B_0} + L_0 + T_j^o$ should be satisfied. $n_l = \lfloor \frac{S(I_0,r)}{B_0} \rfloor$ frames will be buffered while I_0 is being transmitted. These frames will be processed locally, and a dynamic programming algorithm is used to find out the optimal solution.

Algorithm 2: Max-Utility Algorithm

Data: Video frames in the buffer
Result: Scheduling decisions

The frame schedule list $S \leftarrow \{\}$
$u \leftarrow 0$
for j *from 1 to* n_s **do**
  for *each possible resolution* r **do**
    $u' \leftarrow \frac{B_0}{S(I_0,r)} + \alpha \times a(j,r)$
    if $\frac{S(I_0,r)}{B_0} + T_j^o + L_0 \leq T$ *and* $u < u'$ **then**
      $p \leftarrow (0, j, r), u \leftarrow u'$
Add p to S
$n_l \leftarrow \lfloor \frac{S(I_0,r)}{B_0} \rfloor$, $U(0) \leftarrow \{(T^{idle}, 0, 0)\}$
for $i \leftarrow 1$ *to* n_l **do**
  for *each* $(t, u, m) \in U(i-1)$ **do**
    Add (t, u, m) to $U(i)$
    for *each local model* j **do**
      $t' \leftarrow \max(t, i\gamma) + T_j^{npu}$
      if $t' \leq T + i\gamma$ *and* $i\gamma < t$ **then**
        $A \leftarrow \frac{m}{m+1}(u - \frac{m}{n_l * \gamma}) + \alpha \frac{a(j, r_{max})}{m+1}$
        Add $(t', A + \frac{m+1}{n_l * \gamma}, m+1)$ to $U(i)$
  Remove the dominated pairs from U(i)
$(t', u', m') \leftarrow \arg\max_{(t,u,m) \in U(n_l)} u$
for i *from* $n_l - 1$ *to* 0 **do**
  for *each pair* (t, u, m) *in* $U(i)$ **do**
    for *each local model* j **do**
      $A \leftarrow \frac{m}{m'}(u - \frac{m}{n_l * \gamma}) + \alpha \frac{a(j, r_{max})}{m'}$
      if $t + T_j^{npu} = t'$ *and* $A + \frac{m'}{n_l * \gamma} = u'$ **then**
        Add $(i+1, j, r_{max})$ to S
        $t' \leftarrow t, u' \leftarrow u, m' \leftarrow m$
        break
return S

In the algorithm, an array $U(k)$ ($k \in [0, n_l]$) is maintained to find the schedule for maximizing the utility. $U(k)$ is a list of triples, and each triple is denoted as (t, u, m), where utility u is gained by processing m out of the k frames locally within time t. Notice that not all possible triples are maintained in $U(k)$, and only the most efficient ones (i.e., with more utility and less processing time) are kept. More specifically, a triple (t', u', m') is said to dominate another triple (t, u, m) if and only if $t' \leq t, u' \geq u$. Obviously, triple (t', u', m') is more efficient than triple (t, u, m) and all dominated triples will be removed from the list of $U(k)$. Assume that T^{idle} is the queuing time for I_1. Initially, $U(0) = \{(T^{idle}, 0, 0)\}$. To add triples to the list of $U(k)$, we consider two cases: no processing, local processing.

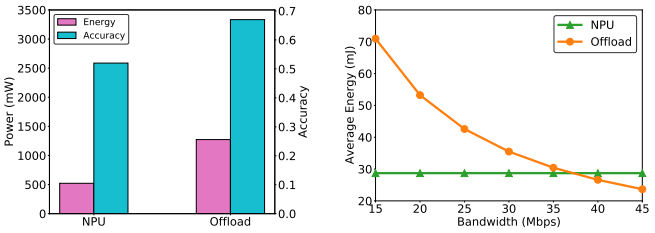
No processing: In this case, the k^{th} frame will not be processed. Processing more frames may require a faster local CNN model to be used for processing. In such cases, the average accuracy decreases and the utility may also decrease. A better solution is to skip this frame. Therefore, we will add all the triples in $U(k-1)$ to $U(k)$.

Local Processing: In this case, it requires T_j^{npu} time to process the frame using the j^{th} model locally. Since the frame does not need to be resized, it is processed with the maximum resolution r_{max}. The new average accuracy can be computed as $A = \frac{m}{m+1}(u - \frac{m}{n_l * \gamma}) + \alpha\frac{a(j,r_{max})}{m+1}$. For each triple $(t, u, m) \in U(k-1)$, a new triple $(\max(t, k\gamma) + T_j^{npu}, A + \frac{m+1}{n_l * \gamma}, m+1)$ is added to the list of $U(k)$. Notice that all local processed frames should be finished within the time constraint. Therefore, $\max(t, k\gamma) + T_j^{npu} \leq k\gamma + T$ should be satisfied for all new triples.

With the list of $U(k)$, we can find a schedule to maximize the utility. The complete description of our algorithm is shown in Algorithm 2. In Lines 2-8, the algorithm maximizes the utility for the offloaded frame, and the schedule decision is determined for the local processing frames in Lines 9-27. The running time of the algorithm is $O(n^2 * n_c)$.

6 Min-Energy Problem

In this section, we study the Min-Energy problem, where the goal is to minimize the energy consumption under the time and accuracy constraints. We first identify the power, accuracy and processing time tradeoffs between computation offloading based approach and NPU-based approach, and then formulate and solve the Min-Energy problem.



(a) Accuracy and Power Consumption (b) Energy consumption under different network conditions

Fig. 4: Accuracy and energy tradeoffs between NPU based appraoch and offload based approach.

6.1 Motivation

To better understand the power, accuracy, and processing time tradeoffs between NPU based approach and offloading based approach, we compare the energy consumption of both approaches. The measurement was done using the HUAWEI Mate 10 pro and we utilized the battery fuel gauge on smartphones to get the power information (i.e., the battery voltage and current). Such information can be obtained through the BatteryManager Class in Android SDK and we developed an Android application which runs as a background process to record the instantaneous voltage and current every 100ms.

To measure the power consumption of the offload based approach, the smartphone continuously offload data to the server through the wireless interface. To measure the power consumption of the NPU based approach, ResNet-50 has been used to continuously process 1000 different images on NPU. As shown in Figure 4(a), the power consumption of NPU based approach is about 60% less than the offload based approach. However, processing data only on NPU may not be the best option for the following reasons. First, when the bandwidth is high, offloading can save more energy since the the data transmitting time is much shorter than the processing time on NPU. We perform an experiment by measuring average energy consumption of offloading 1000 images under different network conditions. The images are offloaded in 224x224 pixels, which is the input size of ResNet-50. As shown in Figure 4(b), offloading saves more energy when the bandwidth is above 40 Mpbs. Second, as shown in Figure 4(a), running ResNet-50 on NPU has much lower accuracy than offloading based approach. Although processing images on NPU is faster and more energy efficient when the wireless bandwidth is low, some images should be offloaded to satisfy the accuracy requirement. Therefore, it is a challenge to determine which image should be offloaded and which one should be processed locally to achieve a better tradeoff between energy and accuracy.

6.2 Problem Formulation

Let P_{npu} denote the power consumption of NPU and let P_{tran} denote the power consumption of transmitting data through the wireless interface. If a frame I_i is processed by the j^{th} model locally on NPU, the energy consumption can be computed as $P_{npu}T_j^{npu}$. If a frame I_i is reduced to resolution r and is offloaded to the edge server for processing, the energy consumption can be computed as $P_{tran}\frac{S(I_i,r)}{B_i}$. Our goal is to minimize the average energy consumption of processing each frame and it can be computed as $\frac{1}{n}\sum_i\sum_j(P_{npu}T_j^{npu}X_i^j + \sum_r P_{tran}\frac{S(I_i,r)X_i^jY_i^r}{B_i})$. In addition to energy, FastVA needs to guarantee that the accuracy is above a predefined threshold set by the users. Let θ_a denote the accuracy threshold and the accuracy constraint $\frac{1}{n}\sum_i\sum_j\sum_r a(j,r)X_i^jY_i^r \geq \theta_a$ must be satisfied.

The Min-Energy problem can be formulated as an integer programming in the following way.

$$\min \quad \frac{1}{n}\sum_{i=0}^{n}\sum_j(P_{npu}T_j^{npu}X_i^j + \sum_r P_{tran}\frac{S(I_i,r)X_i^jY_i^r}{B_i}) \tag{15}$$

$$\text{s.t.} \quad \sum_{k\leq i}\sum_j T_j^{npu}X_k^j \leq T + (i-k)*\gamma,\ \forall i,k \tag{16}$$

$$D(k) \leq T + (i-k)*\gamma,\ \forall i,k \tag{17}$$

$$\frac{1}{n}\sum_i\sum_j\sum_r a(j,r)X_i^jY_i^r \geq \theta_a \tag{18}$$

$$\sum_j X_i^j \leq 1,\ \forall i \tag{19}$$

$$\sum_r Y_i^r \leq \sum_j X_i^j,\ \forall i \tag{20}$$

$$Y_i^r, X_i^j \in \{0,1\},\ \forall i,j \tag{21}$$

Objective (15) minimizes the energy consumption. Constraints (16) and (17) specify that the frames must be processed within the time requirement. Constraint (18) specify that the accuracy constraint must be satisfied. Constraint (19) specifies that each frame can at most be processed by a CNN model either remotely or locally. Constraint (20)

specifies that each image can only be resized to a certain resolution.

Theorem 4. *The Min-Energy problem is NP-hard.*

Proof. We reduce a well known NP-hard problem, the partition problem to the Min-Energy problem. In the partition problem, there is a set U which includes n numbers $(p_1, p_2, p_3, \ldots, p_n)$ and the goal is to partition the set into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals to the sum of the numbers in S_2.

For an arbitrary instance of the partition problem, we can construct an instance of Min-Energy problem as follows. A frame I_i is created for each number p_i and the arrival times of the frames are set to be 0. The time constraint T is set to be $\frac{1}{2}\sum_{p_i \in U} p_i$ for all frames. In the constructed Min-Energy problem, all frames must be offloaded to the server and there are two possible resolution options r_0 and r_1. The data size of the frame I_i is defined as

$$S(I_i, r_k) = \begin{cases} 0 & \text{if } k = 0 \\ p_i * B_i & \text{if } k = 1 \end{cases}$$

There is only one CNN model j_0 running on the server. P_{tran} is set to be $-n$. $L_i, T^o_{j_0}, \theta_a$ are set to be 0.

Since all frames are offloaded, the solution of Min-Energy problem minimizes $\frac{1}{n}\sum_i \sum_r P_{tran}\frac{S(I_i,r)Y_i^r}{B_i}$, and it actually maximizes $\sum_{Y_i^{r_1}=1} p_i$. Since the $\theta_a = 0$, the accuracy constraint is satisfied in all solutions. Since the time constraint must be satisfied, we have $\sum_i \sum_r \frac{S(I_i,r)Y_i^r}{B_i} \leq T$ which is equivalent to $\sum_{Y_i^{r_1}=1} p_i \leq \frac{1}{2}\sum_{p_k \in U} p_k$.

A solution to this instance of Min-Energy problem maximizes $\sum_{Y_i^{r_1}=1} p_i$. If $\sum_{Y_i^{r_1}=1} p_i = \frac{1}{2}\sum_{p_k \in U} p_k$, the feasible solution of the partition problem can be constructed as follow. The number p_i will be put into S_1 if $Y_i^{r_1} = 1$; otherwise it is put into S_2. Thus, the solution to the Min-Energy problem is also the solution to the partition problem. This completes the reduction and hence the proof. □

6.3 Min-Energy Algorithm

Since the Min-Energy problem is NP-hard, we propose a heuristic based algorithm. The basic idea is as follows. With two constraints (processing time and accuracy) in the problem, it is difficult to minimize the energy consumption while satisfying both constraints at the same time. Thus, in our algorithm, these two constraints are handled in two phases, initial scheduling and frame rescheduling. The initial scheduling phase focuses on generating a schedule with minimum energy consumption under the accuracy constraint. Since the time constraint may not be satisfied, in the frame rescheduling phase, some frames are rescheduled to guarantee that both constraints are satisfied.

6.3.1 Initial Scheduling

In this phase, our goal is to minimize the energy consumption under the accuracy constraint. Assume that the a new frame arrives at the buffer and there are n_b frames in the buffer to be processed. For these n_b frames, a dynamic programming algorithm is used to find an optimal schedule decision. More specifically, an array $G(k)(k \in [0, n_b])$ is maintained to find the schedule for minimizing the energy consumption. $G(k)$ is a list of pairs, and each pair is denoted as (A, E); i.e., it costs energy E to achieve accuracy A by processing the first k frames. Similar to the Max-Utility algorithm, only the most efficient pairs are kept in the list (i.e., with higher accuracy and less energy consumption). A pair (A, E) is more efficient than (A', E') if and only if $A > A', E < E'$. Initially, $G(0) = \{(0, 0)\}$. To add the pairs to the list of $G(k)$, we consider two cases: offloading and local processing.

Offloading: If a frame I_i is processed by the j^{th} model on the server, it takes $\frac{S(I_i,r)}{B_i}$ time to transmit the frame. Therefore, for each pair (A, E) in $G(k-1)$, a new pair $(A + a(j, r), E + P_{tran}\frac{S(I_i,r)}{B_i})$ is added to $G(k)$.

Local Processing: If a frame I_i is processed by the j^{th} model locally on NPU, for each pair (A, E) in $G(k-1)$, a new pair $(A + a(j, r), E + P_{npu}T_j^{npu})$ is added to $G(k)$.

With the list of $G(k)$, we can find a schedule to minimize the energy consumption under the accuracy constraint. However, the schedule may not be feasible due to the time constraint and the frame rescheduling phase will be used to address this problem.

6.3.2 Frame Rescheduling

In this phase, the goal is to modify the schedule found in the initial phase to satisfy the time constraint and minimize the energy consumption. More specifically, Let S denote the schedule found in the initial scheduling phase. Our algorithm checks the time constraint for the frames one by one in the ascending order of their arrival time. If the time constraint is not satisfied for frame I_i, the frame will be rescheduled and the following two cases are considered.

Case 1: If the frame is processed by the j^{th} model locally on NPU, it will be offloaded to the server for processing. Since the accuracy constraint is already satisfied in the Initial Scheduling phase, the n_b frame should not be processed by a model with lower accuracy after rescheduling. In other words, if I_i is resized to resolution r' and processed by the model j' on the server, $a(j', r') \geq a(j, r_{max})$ must be satisfied. Moreover, the time constraint (Eq. 17) must be satisfied for the first i frames. A CNN model j' and resolution r' will be selected if it can satisfy the time and accuracy constraint and minimizes energy consumption for frame I_i.

In this solution, we did not consider the option of using a different local model to process the frame I_i since such an option could not satisfy the accuracy constraint. More specifically, a local CNN model used in S has higher accuracy if its processing time is longer, since the list $G(k)$ only keeps the most efficient pairs. To satisfy the time constraint, I_i must be processed by another model j with lower accuracy and shorter processing time on NPU. However, this rescheduling will not satisfy the accuracy constraint and it can be proved by contradiction. Assume that this rescheduling is feasible and we can generate a new schedule S' based on S by only rescheduling I_i to be processed the j^{th} model on NPU. S' consumes less energy than S and the accuracy constraint is satisfied. However, it contradicts the fact that S is the optimal solution in the initial scheduling phase and thus S' cannot satisfy the accuracy constraint.

Case 2: If the frame is offloaded in resolution r and processed by the j^{th} model on the server, it will be moved

to NPU for processing. To guarantee that the accuracy constraint is still satisfied after rescheduling, the constraint $a(j', r_{max}) \geq a(j, r)$ must be satisfied if I_i is processed by j'^{th} model on NPU. Moreover, the time constraint (Eq. 16) must be satisfied for the first i frames. A local CNN model j' will be selected if it has the lowest energy consumption and satisfies the time and accuracy constraints. Similar to Case 1, we do not consider the option of reducing the frame resolution or processing it with a lower accuracy model on the server since such an option will not satisfy the accuracy constraint.

Algorithm 3: Min-Energy Algorithm

Data: Video frames in the buffer
Result: Scheduling decision
1 The frame schedule list $S \leftarrow \{\}$
2 $G(0) \leftarrow \{(0, 0, S)\}$
3 **for** *i from 1 to n_b* **do**
4 **for** *each $(A, E, S) \in G(i-1)$* **do**
5 **for** *each local model j* **do**
6 $A' \leftarrow A + a(j, r_{max})$
7 $E' \leftarrow E + P_{npu} T_j^{npu}$
8 $S' \leftarrow \{(i, j, r_{max})\} \cup S$
9 Add (A', E', S') to $G(i)$
10 Remove the dominated pairs from $G(i)$
11 Remove pair (A, E, S) from $G(n_b)$ if $A < \theta_a$
12 $(A, E, S) \leftarrow \text{argmin}_{(A,E,S) \in G(n_b)} E$
13 **for** $(i, j, r) \in S$ **do**
14 **if** *Time constraint is not satisfied for I_i* **then**
15 $E_{min} \leftarrow +\infty$
16 **if** *model j is on the server* **then**
17 **for** *each local model j'* **do**
18 $E \leftarrow P_{npu} T_j^{npu}$
19 **if** $a(j, r) \leq a(j', r_{max})$ *and* $E_{min} > E$ *and Constraint (16) is satisfied for* $I_k (k \in [1, i])$ **then**
20 Replace (i, j, r) by (i, j', r_{max}) in S
21 $E_{min} \leftarrow E$
22 **else**
23 **for** *each possible resolution r'* **do**
24 **for** *each model j on the server* **do**
25 $E \leftarrow P_{tran} \frac{S(I_i, r')}{B_i}$
26 **if** $a(j, r) \leq a(j', r')$ *and* $E_{min} > E$ *and Constraint (17) is satisfied for* $I_k (k \in [1, i])$ **then**
27 Replace (i, j, r) by (i, j', r') in S
28 $E_{min} \leftarrow E$
29 **return** S

The complete description of our algorithm is shown in Algorithm 3. Lines 1-12 are for the initial scheduling phase and Lines 13-28 are for the frame rescheduling phase. The running time of our algorithm is $O(n_b * n_r * n_c)$.

7 Performance Evaluations

In this section, we evaluate the performance of the proposed algorithms, Max-Accuracy, Max-Utility and Min-Energy, and compare them with other approaches.

7.1 Experiment Setup

Currently, there are only a few smartphones on the market with dedicated NPUs. In the evaluation, we use HUAWEI Mate 10 pro smartphone because it is equipped with NPU and it has a published HUAWEI DDK [37] for developers. Since NPU has a different architecture from CPU, the existing CNN models have to be optimized before running on NPU. The HUAWEI DDK includes toolsets to do such optimization for NPU from CNN models trained by the deep learning frameworks Caffe [38]. The HUAWEI DDK also includes the APIs to run the CNN models, and a few Java Native Interface (JNI) functions are provided to use the APIs on Android. Since these JNI functions are hard coded for running a specific model, we have implemented more flexible JNI functions which can run different CNN models.

In FastVA, the frames are offloaded in the lossless PNG format. FastVA periodically estimates the wireless bandwidth based on the harmonic mean of the uploading data rate of the past several frames [39]. The harmonic mean is robust to large outliers and is used to minimize the impact of wireless channel variations. Other estimation methods can be found in [40], [41]. The edge server is a desktop with AMD Ryzen 7 1700 CPU, GeForce GTX1070 Ti graphics card and 16 GB RAM. We have installed the Caffe framework to run the CNN models on GPU.

In the experiment, object classifications are performed on mobile devices, which are common computer vision tasks for many mobile applications. In our experiment, two different object recognition CNN models are used, ResNet-50 [9] and SqueezeNet [42], which are well known and are widely used. Moreover, SqueezeNet has a compact structure and it is much smaller than ResNet. It can be considered as a compressed model that runs faster than ResNet at the cost of accuracy. This allows the application to achieve tradeoffs between accuracy and processing time under different network condition and time constraint.

In the evaluation, we use a subset of videos from the FCVID dataset [43], which includes many real-world videos. These videos have been used for training models related to object classification and activity recognition. In our experiment, we focus on object classification, and thus activity recognition clips are not used. Since the dataset is very large, about 1.9 TB, we randomly select 40 videos from the dataset and filter out the noisy data. Since the labels of FCVID and ImageNet are different, we map the labels produced by the CNN models to that used by the FCVID dataset.

We evaluate the proposed algorithms with different frame rates. Most videos in the dataset use 30 fps, and thus we have to change their frame rate by decoding/encoding. For both CNN models (ResNet-50 and SqueezeNet), the maximum resolution of the input image is 224x224 pixels. This resolution can be downsized for some offloading images, and we consider 5 different resolutions: 45x45, 90x90, 134x134, 179x179 and 224x224 pixels. The time constraint for each frame is set to be 200 ms in all the experiments. The running time of Max-Accuracy and Max-Utility algorithm is less than 1 ms on the smartphone and it is negligible compared to the time constraint (100 ms level).

CNN model		Processing Time (ms)	Transmission Time (ms)	Top-1 Accuracy
ResNet	Local	52	0	0.52
	Server	69	39 - 242	0.67
SqueezeNet	Local	17	0	0.41
	Server	9	39 - 242	0.51

TABLE 2: The performance of the CNN models.

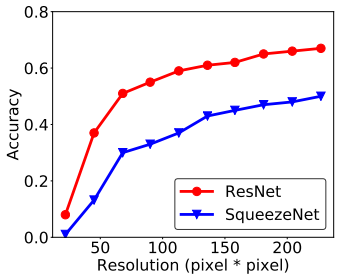
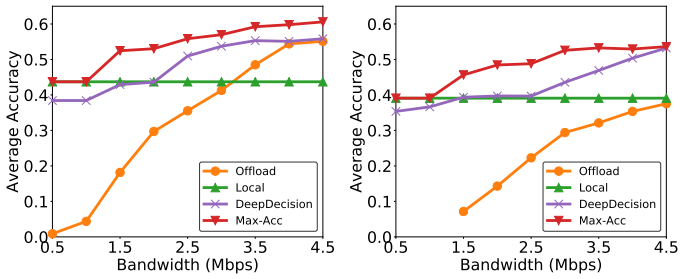


Fig. 5: Accuracy vs. Resolution.



(a) Frame Rate 30 fps (b) Frame Rate 50 fps

Fig. 6: The performance of different methods under different network conditions.

7.2 The Effects of CNN models

To have a better understanding of how the CNN models perform, we run ResNet-50 and SqueezeNet on the edge server and NPU with randomly selected 4000 images from the VOC dataset. As shown in table 2, the accuracy of ResNet-50 is about 30% better than SqueezeNet on the server and it is about 25% better than SqueezeNet on the NPU. However, SqueezeNet is 700% faster than ResNet-50 on the server and it is 300% faster than ResNet-50 on the NPU. Although running these CNN models has high accuracy on the server, there is a network latency between the server and mobile device. As shown in the table, the transmission time can range from tens of milliseconds to hundreds of milliseconds based on the network condition and frame data size. When the network condition is poor, offloading may take much longer time than running on NPU.

Figure 5 shows the tradeoff between accuracy and resolution. We note that the accuracy does not scale linearly with the resolution. The data in Table 2 and Figure 5 are used for making scheduling decisions in FastVA.

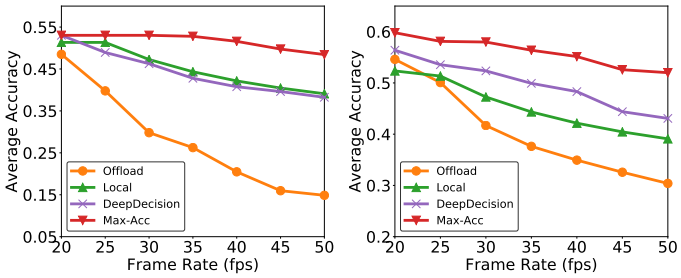
7.3 The Performance of Max-Accuracy

We compare the performance of Max-Accuracy with the following schedule algorithms.

- **Offload:** In this method, all frames must be offloaded to the edge server for processing. Each frame will be resized to a resolution so that it can be offloaded before the next frame arrives, and the server chooses the most accurate model that can process the frames and return the result within the time constraint.
- **Local:** In this method, all frames are processed locally. It uses the proposed dynamic programming technique to find the optimal schedule decision for local processing.
- **DeepDecision:** This is a simplified version of DeepDecision [5] which optimizes the accuracy and utility within the time constraint. DeepDecision divides time into windows of equal size. At the beginning of each time window, it picks a specific resolution and CNN model to process all the frames within a time window.
- **Optimal:** This shows the upper bound for all methods. It tries all possible combinations and chooses the schedule that maximizes the accuracy. Notice that this method cannot be used for processing videos in real time since it takes too much time to search all possible schedules. We can only find the optimal solution offline by replaying the data trace.

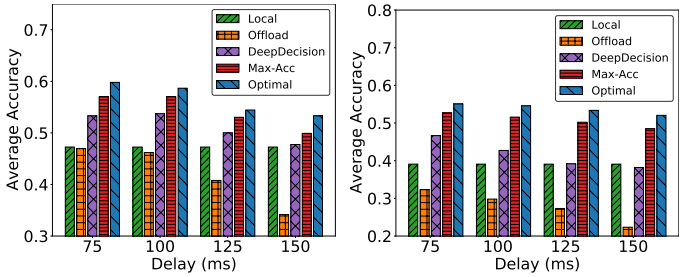
The performance of the schedule algorithms depends on several factors, the bandwidth, latency and the processing time requirement specified by the applications.

In Figure 6, we compare Max-Accuracy with the Local, Offload and DeepDecision method under different network conditions. In the evaluation, we set the network latency to be 100 ms. The Local method does not offload any frames, and thus its performance remains the same under different network conditions. The Local method can achieve the same accuracy as the Max-Accuracy algorithm when the bandwidth is low, since most of the video frames will be processed locally and the Local method can find an optimal solution. Notice that the Local method performs better than DeepDecision when the bandwidth is low. The reason is as follows. DeepDecision makes the same schedule decision for frames within a time slot and NPU may not be fully utilized if only SqueezeNet is used. In contrast, the Local method achieves higher accuracy by using ResNet to process some of the frames within the time slot. In Figure 6(b), the Offload method is not capable of processing all frames when the bandwidth is lower than 1.5 Mbps. When the network bandwidth is low, the Offload method performs poorly since it has to resized video frames into an extremely small size and then reduce the accuracy. As shown in Figure 5, even with an advanced CNN model, the accuracy is till low with these low resolution images. As the network bandwidth increases, the differences among the Max-Accuracy, DeepDecision and



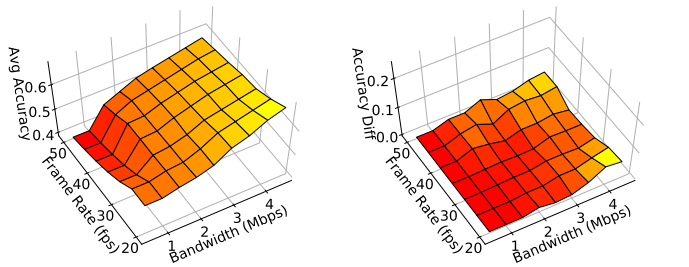
(a) $B = 2$ Mbps (b) $B = 3$ Mbps

Fig. 7: The performance of different methods under different frame rate requirements.



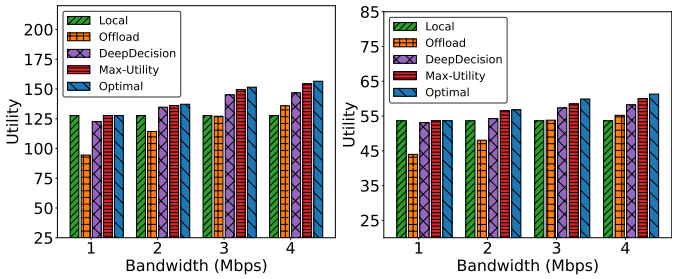
(a) Frame Rate 30 fps (b) Frame Rate 50 fps

Fig. 9: The performance of different methods under different network latency.



(a) Performance of the Optimal (b) Difference between Max-Accuracy and Optimal

Fig. 8: Comparison between optimal and Max-Accuracy



(a) $\alpha = 200$ (b) $\alpha = 50$

Fig. 10: The impact of network bandwidth.

Offload become smaller since the mobile device can offload most of the frames in high resolution and achieve better accuracy.

In Figure 7, we evaluate the impact of frame rate for different methods. As can be seen from the figures, the performance of all methods drops when the frame rate is high. As the frame rate requirement increases, more frames have to be resized to lower resolutions. That is why the Offload method suffers a 30% accuracy drop in the experiments. In contrast, there is no significant accuracy drop in Max-Accuracy, since they can avoid reducing the solution by processing the video frames on NPU.

In Figure 8, we compare Max-Accuracy with the Optimal method under various frame rates and network conditions. As shown in Figure 8(a), the accuracy of Optimal increases when the network bandwidth increases, because the mobile device can upload more frames with higher resolution. To support a higher frame rate, more frames must be processed within the time constraint and the optimal method has to use fast CNN models with low resolution or low accuracy, resulting in low accuracy.

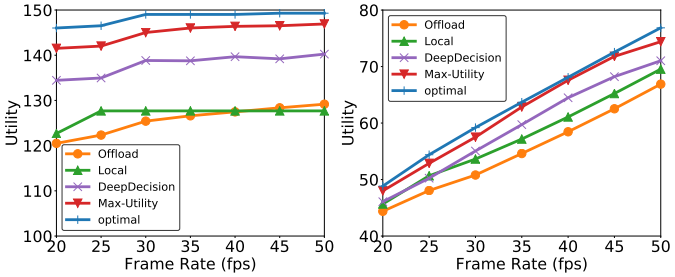
In figure 8(b), we plot the accuracy difference between Optimal and Max-Accuracy. The accuracy difference is computed using the accuracy of the Optimal method minus that of Max-Accuracy. As can be seen from the figure, the difference is almost 0 in most cases, which indicates that Max-Accuracy is close to Optimal.

In Figure 9, we evaluate the impact of network latency on accuracy. We set the uplink network bandwidth to be 3 Mbps and set the frame rate to be 30 and 50 fps. Since the Local method does not offload any frames, its performance remains the same. A longer delay means that less frames can be offloaded to the server for processing, since the result must be returned within the time constraint. Therefore, for the Offload, DeepDecision, Optimal and Max-Accuracy algorithms, the performance drops as the network latency increases. Compared to DeepDecision, Optimal and Max-Accuracy, a significant accuracy drop can be observed in the Offload method, when the network latency becomes larger. This is because DeepDecision, Optimal and Max-Accuracy can schedule frames to be processed locally at this time to deal with the long network latency. Although the accuracy is also dropped by processing the frames on NPU, the impact is not as significant as that in the Offload method.

7.4 The Performance of Max-Utility

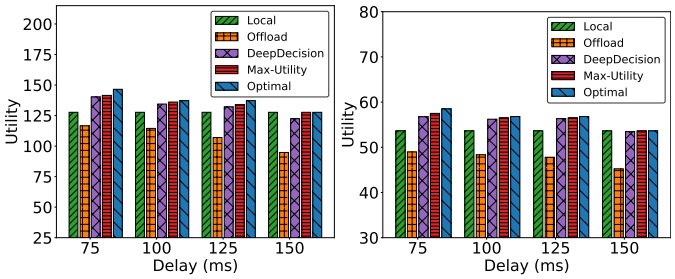
To evaluate the performance of Max-utility, we still compare it to Offload, Local, and Optimal. Since we focus on utility instead of accuracy in this subsection, these algorithms are also modified to maximize utility instead of accuracy.

In Figure 10, we evaluate the impact of network bandwidth for different methods. In the comparison, the frame rate and the network latency are set to be 30 fps and 100 ms respectively, and the tradeoff parameter α is set to be 50 and 200. The Local approach cannot offload any video frames to the server, and thus its utility remains the same in different network conditions. When the bandwidth is low, the Offload method may have to upload low resolution frames, resulting in low utility, and thus it underperforms the Local method. As the network bandwidth increases, the performance difference among Offload, DeepDecision, Max-Utility, and Optimal becomes smaller, since most frames



(a) $\alpha = 200$ (b) $\alpha = 50$

Fig. 11: The effects of frame rates.



(a) $\alpha = 200$ (b) $\alpha = 50$

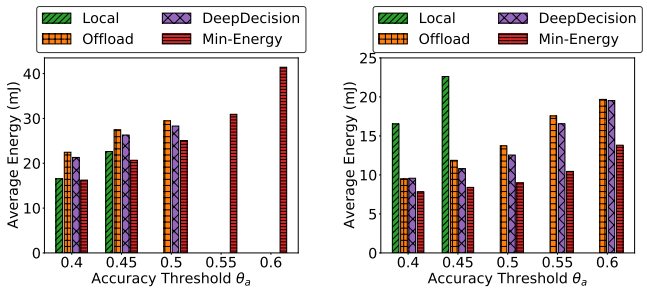
Fig. 12: The effects of network latency

can be transmitted with higher resolution to achieve better accuracy.

As shown in the figure, when the network bandwidth decreases, the performance of Offload, DeepDecision, Max-Utility and Optical all drops. However, the performance of DeepDecision, Max-Utility and Optimal drops much slower than Offload. The reason is as follows. When α is large, as shown Figure 10(a), the accuracy has more weight in calculating the utility. Max-Utility achieves high accuracy and then high utility by offloading when network bandwidth is high and by local execution when the network bandwidth is low. When α is small, as shown Figure 10(b), the processing time (frame rate) has more weight in calculating the utility. Max-Utility supports high frame rate and then achieves high utility by offloading when network bandwidth is high and by local execution when the network bandwidth is low.

Figure 11 shows the impacts of frame rate for different methods. In the evaluation, we set the network bandwidth to be 2.5 Mbps and the network latency to be 100 ms. As shown in the figure, Max-Utility outperforms Offload, Local and DeepDecision methods. When α is small, as shown Figure 11(b), the processing time (frame rate) has more weight in calculating the utility, and thus the utility of all methods increases when the frame rate increases. When α is large, as shown Figure 11(a), the accuracy has more weight in calculating the utility, and thus the utility of all methods does not increases too much when the frame rate increases.

Figure 12 shows the impact of network latency for different methods. We set the frame rate to be 30 fps and the network bandwidth to be 2 Mbps. Since the Local method does not offload any video frames to the server, its performance remains the same. As the network latency increases, less video frames can be offloaded to the server due to time



(a) $B = 5Mbps$ (b) $B = 15Mbps$

Fig. 13: The effects of the accuracy threshold θ_a.

constraints, and hence degrading the performance of the Offload, DeepDecision, Max-Utility, and Optimal algorithms.

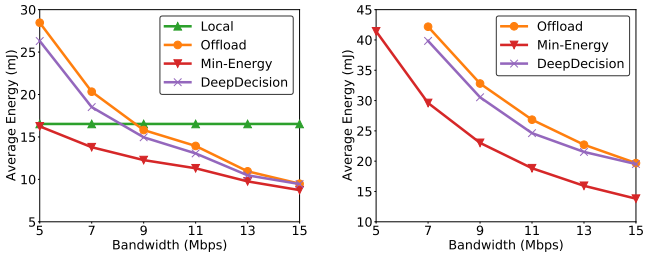
7.5 The Performance of Min-Energy

To evaluate the performance of Min-Energy, we compare it to Local, Offload, DeepDecision and Optimal, where the performance is measured in terms of energy consumption under various accuracy and time constraints.

In Figure 13, we evaluate the impact of the accuracy threshold θ_a for different methods. In the comparison, the frame rate and the network latency are set to be 30 fps and 100 ms respectively, and the bandwidth is set to be 5 Mbps and 15 Mbps. As the accuracy threshold increases, more frames have to be offloaded in a higher resolution or processed by a local CNN model with higher accuracy and longer processing time. This explains why the energy consumption of all methods increases when the accuracy threshold increases.

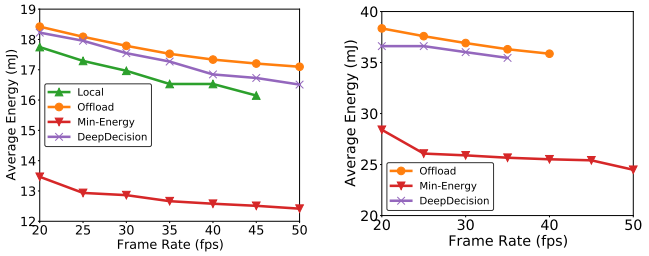
As shown in Figure 13(a), when the accuracy threshold is above 0.45, the Local method is not capable of processing all frames and thus not shown in the figure; when the accuracy threshold is above 0.5, the Offload and DeepDecision method cannot process all frames and thus not shown in the figure. When the accuracy threshold and the bandwidth is low, as shown in Figure 13(a), the Local method has similar energy consumption to the Min-Energy method since most frames are processed locally. Notice that the Local method performs better than DeepDecision when the accuracy threshold is low in Figure 13(a). This is because DeepDecision makes the same schedule decision for frames within a time slot and the energy consumption is higher if the frames are only processed by the ResNet model on NPU. In contrast, the Local method can further reduce the energy consumption by using SqueezeNet to process some of the frames within the time slot.

By comparing Figure 13(b) and Figure 13(a), we can see that the Local method has the same energy consumption (note that the scale on y-axis is different). This is because the Local method does not offload any frame to the server and then its performance is not affected by the network bandwidth. This is different for the other three methods. When the wireless bandwidth becomes higher, it costs less energy to offload high resolution frames than processing them locally on NPU. As a result, DeepDecision, Offload



(a) $\theta_a = 0.4$ (b) $\theta_a = 0.6$

Fig. 14: The impact of network bandwidth



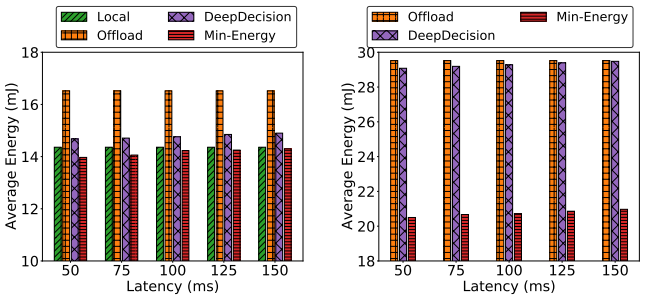
(a) $\theta_a = 0.4$ (b) $\theta_a = 0.6$

Fig. 15: The effects of frame rates

and Min-Energy performs much better when the network bandwidth increases, as shown in Figure 13(b). Different from the Offload method which offloads all frames to the server, in Min-Energy, the frames with large data size are processed locally on NPU to save energy and the frames with small data size are offloaded in high resolution to satisfy the accuracy constraint. As a result, Min-Energy has the best performance.

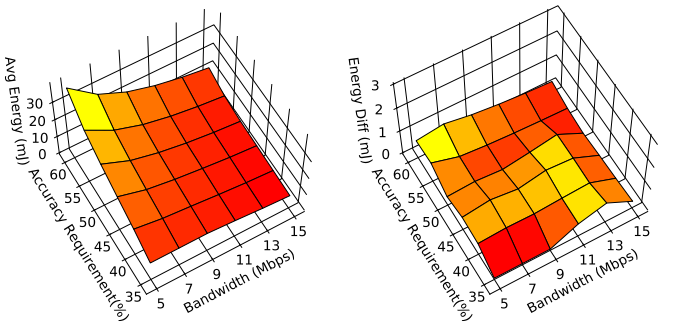
In Figure 14, we compare the performance of Min-Energy with the Local, Offload, and DeepDecision method under different network conditions. In the evaluation, the frame rate and the network latency are set to be 30 fps and 100 ms, and the accuracy threshold θ_a is set to be 0.4 and 0.6. Since the Local method does not offload any frames, its performance remains the same under different network condition in Figure 14(a). As shown in Table 2, the accuracy of the local models is below 0.6. As a result, the Local method could not satisfy the accuracy constraint ($\theta_a = 0.6$) and it is not shown in Figure 14(b). When the bandwidth increases, the energy consumption of Offload, DeepDecision and Min-Energy drops. Compared to Figure 14(a), the energy consumption of Min-Energy drops more significantly in Figure 14(b). The reason is as follows. When θ_a is small, the local CNN models is used to save energy and the accuracy constraint is satisfied. Min-Energy reduces energy consumption by processing most frames locally. When θ_a is large, Min-Energy has to spend more energy to satisfy the accuracy constraint by offloading most frames to the server. As a result, the network bandwidth has a larger impact on the performance of Min-Energy in Figure 14(b).

Figure 15 shows the impacts of frame rate for different methods. In the evaluation, the bandwidth and the network



(a) $\theta_a = 0.4$ (b) $\theta_a = 0.6$

Fig. 16: The effects of network latency



(a) Performance of the Optimal (b) Difference between Min-Energy and Optimal

Fig. 17: Comparison between optimal and Min-Energy

latency are set to be 8 Mbps and 100 ms, and the accuracy threshold θ_a is set to be 0.4 and 0.6. As can be seen from the figures, the energy consumption of all methods drop slightly when the frame rate increases. The reason is as follows. In the experiment, the frames can only be offloaded in five different resolutions and there are four different CNN models used to process the frames. The average accuracy of the processed frames is a little bit higher than accuracy threshold θ_a and a small amount of energy is wasted to increase the average accuracy above θ_a. When the frame rate increases, more frames are processed within the same amount of time. The average accuracy is closer to θ_a, and less energy is used.

Figure 16 shows the impact of network latency for different methods. In the evaluation, the bandwidth and the network latency are set to be 10 Mbps and 100 ms, and the accuracy threshold θ_a is set to be 0.4 and 0.6. Since the Local method does not offload any video frames to the server, its performance remains the same. The performance of Offload, DeepDecision and Min-Energy change slightly when the network latency increases. This is because the frames have to be offloaded in high resolution to satisfy the accuracy requirement and the energy consumption of offloading the high resolution frames remains the same under different network latency.

In Figure 17, we compare Min-Energy with the Optimal method under different accuracy thresholds and network conditions. As shown in Figure 17(a), the average energy

consumption of Optimal decreases when the network bandwidth increases. This is because offloading the frames to the server is more energy efficient than processing the frames on NPU when the network bandwidth is high. To satisfy higher accuracy constraint, the resolution of the offloaded frames must be increased and the frames must be processed by the models with higher accuracy and power consumption. As a result, the average energy consumption increases as the accuracy threshold increases.

In Figure 17(b), we show the average energy difference between Optimal and Min-Energy. The energy difference is computed using the average energy of Min-Energy minus that of Optimal. As can be seen from the figure, the difference is within 1 mJ in all cases, which indicates that Min-Energy is close to Optimal.

8 Conclusions

In this paper, we proposed a framework called FastVA, which supports deep learning video analytics through edge processing and Neural Processing Unit (NPU) in mobile. We are the first to study the benefits and limitations of using NPU to run CNN models to better understand the characteristics of NPU in mobile. Based on the accuracy and processing time requirement of the mobile application, we studied three problems: *Max-Accuracy* where the goal is to maximize the accuracy under some time constraints, *Max-Utility* where the goal is to maximize the utility which is a weighted function of processing time and accuracy, and *Min-Energy* where the goal is to minimize the energy consumption under the time and accuracy constraints. To solve these problems, we have to determine when to offload the computation and when to use NPU. The solution depends on the network condition, the special characteristics of NPU, and the optimization goal. We formulated them as integer programming problems and proposed heuristics based solutions. We have implemented FastVA on smartphones and demonstrated its effectiveness through extensive evaluations.

Acknowledgement

This work was supported in part by the National Science Foundation under grant number 2125208.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. *IEEE ICCV*, 2015.

[2] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. *ACM Sensys*, 2015.

[3] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E. Culler, and Randy H. Katz. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. *ACM Sensys*, 2018.

[4] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. *IEEE ICDCS*, 2017.

[5] Xukan Ran, Haoliang Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics. *IEEE INFOCOM*, 2018.

[6] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An Approximation-Based Execution Framework for Deep Stream Processing under Resource Constraints. *ACM Mobisys*, 2016.

[7] HiSilicon Kirin 970 Performance Overview. https://www.anandtech.com/show/12195/hisilicon-kirin-970-power-performance-overview/6.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *IEEE CVPR*, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *IEEE CVPR*, 2016.

[10] Sourav Bhattacharya and Nicholas D. Lane. Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables. *ACM Sensys*, 2016.

[11] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. *IEEE CVPR*, 2015.

[12] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. *ACM Sensys*, 2018.

[13] S. Han, H. Mao, and W. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *IEEE International Conference on Learning Representations (ICLR)*, 2016.

[14] D. Lin, S. Talathi, and S. Annapureddy. Fixed Point Quantization of Deep Convolutional Networks. *International Conference on Machine Learning*, 2016.

[15] Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep Learning with Low Precision by Half-Wave Gaussian Quantization. *IEEE CVPR*, 2017.

[16] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. *ACM Mobisys*, 2010.

[17] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic Execution between Mobile Device and Cloud. *ACM EuroSys*, 2011.

[18] M. Barbera, S. Kosta, A. Mei, and J. Stefa. To Offload or Not To Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing. *IEEE INFOCOM*, 2013.

[19] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura. COSMOS: Computation Offloading as a Service for Mobile Devices. *ACM Mobihoc*, 2014.

[20] Yeli Geng, Wenjie Hu, Yi Yang, Wei Gao, and Guohong Cao. Energy-efficient computation offloading in cellular networks. *IEEE ICNP*, 2015.

[21] Yeli Geng, Yi Yang, and Guohong Cao. Energy-efficient computation offloading for multicore-based mobile devices. *IEEE INFOCOM*, 2018.

[22] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai. Hermes: Latency Optimal Task Assignment for Resource-Constrained Mobile Computing. *IEEE Transactions on Mobile Computing*, 2017.

[23] Y. Geng and G. Cao. Peer-assisted computation offloading in wireless networks. *IEEE Transactions on Wireless Communications*, 17(7):4565–4578, 2018.

[24] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative Intelligence Between The Cloud And Mobile Edge. *ACM SIGARCH Computer Architecture News*, 2017.

[25] Zongqing Lu, Kevin Chan, and Thomas La Porta. A Computing Platform for Video Crowdprocessing Using Deep Learning. *IEEE INFOCOM*, 2018.

[26] T. Tan and G. Cao. Deep Learning Video Analytics on Edge Computing Devices. *IEEE SECON*, 2021.

[27] T. Tan and G. Cao. Efficient Execution of Deep Neural Networks on Mobile Devices with NPU . *IEEE IPSN*, 2021.

[28] L. Oskouei, S. Salar and H. Golestani and M. Hashemi and S. Ghiasi. CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android. *ACM international conference on Multimedia*, 2016.

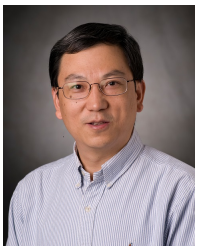
[29] Mohammad Motamedi, Daniel Fong, and Soheil Ghiasi. Cappuccino: efficient CNN inference software synthesis for mobile system-on-chips. *IEEE Embedded Systems Letters*, 2019.

[30] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile GPU-Based Deep Learning Framework for Continuous Vision Applications. *ACM Mobisys*, 2017.

[31] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx:

A software accelerator for low-power deep learning inference on mobile devices. *IEEE IPSN*, 2016.
[32] A. Zisserman O. M. Parkhi, A. Vedaldi. Deep Face Recognition. *British Machine Vision Conference*, 2015.
[33] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. Technical report, University of Massachusetts, Amherst, 2007.
[34] E. Mark, E. SM Ali, V. Luc, W. Christopher KI, W. John, and Z. Andrew. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 2015.
[35] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *IEEE CVPR*, 2016.
[36] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. *European conference on computer vision*, 2014.
[37] HiAI. https://developer.huawei.com/consumer/en/devservice/doc/2020315.
[38] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *ACM International Conference on Multimedia*, 2014.
[39] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with Festive. *ACM International Conference on Emerging Networking Experiments and Technologies*, 2012.
[40] X. Xing, J. Dang, S. Mishira, and x. Liu. A Highly Scalable Bandwidth Estimation of Commercial Hotspot Access Points. *IEEE INFOCOM*, 2011.
[41] Z.Ahmed Hamdy, R. Darijo, and S. Cormac J. ArbMter+: Adaptive Rate-Based Intelligent Http Streaming Algorithm for Mobile Networks. *IEEE Transactions on Mobile Computing*, 2018.
[42] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.
[43] Yu-Gang Jiang, Zuxuan Wu, Jun Wang, Xiangyang Xue, and Shih-Fu Chang. Exploiting Feature and Class Relationships in Video Categorization with Regularized Deep Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Tianxiang Tan received the BE degree from Sun Yat-sen University and the MS degree in computer science from University of Southern California. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, the Pennsylvania State University. His research interests include mobile cloud computing, edge computing and deep learning. He is a student member of the IEEE.



Guohong Cao received his B.S. degree in computer science from Xi'an Jiaotong University, and his Ph.D. in computer science from the Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently a Distinguished Professor. He has published more than 200 papers in the areas of wireless networks, mobile computing, machine learning, wireless security and privacy, and Internet of Things, which have been cited over 20000 times. He has served on the editorial board of IEEE Transactions on Mobile Computing, IEEE Transactions on Wireless Communications, and IEEE Transactions on Vehicular Technology, and has served on the organizing and technical program committees of many conferences, including the TPC Chair/Co-Chair of IEEE SRDS, MASS, and INFOCOM. He has received several best paper awards, the IEEE INFOCOM Test of Time award, and the NSF CAREER award. He is a Fellow of the AAAS and a Fellow of the IEEE.