FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile

Tianxiang Tan and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
Email: {txt51, gxc27}@psu.edu

arXiv:2001.04049v1 [cs.NI] 13 Jan 2020

***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 two problems: *Max-Accuracy* where the goal is to maximize the accuracy under some time constraints, and *Max-Utility* where the goal is to maximize the utility which is a weighted function of processing time and accuracy. 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.**

I. 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.

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 only has 200MB 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 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.

Based on the accuracy and the processing time requirement of the mobile application, we study two problems: *Max-Accuracy* where the goal is to maximize the accuracy under some time constraints, and *Max-Utility* where the goal is to maximize the utility which is a weighted function of accuracy and processing time. To solve these two 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 and propose a heuristic based solution.
- We formulate the Max-Utility problem and propose an approximation based solution.
- We implement FastVA on smartphones and compare it with other techniques through extensive evaluations.

The rest of the paper is organized as follows. Section II presents related work. Section III 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 IV. In Section V, we study the *Max-Utility Problem* and give a heuristic based solution. Section VI presents the evaluation results and Section VII concludes the paper.

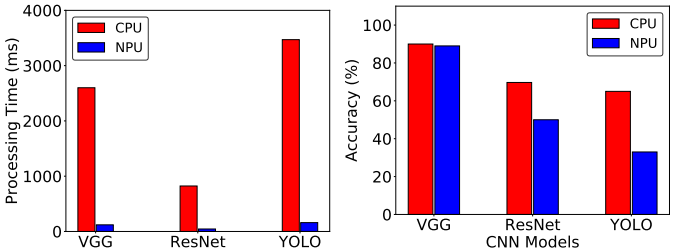
II. Related Work

Over past years, there have been significant advances on object recognition with CNN models. For example, GoogleNet [7] and ResNet [8] 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 [9], 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.* [10] optimize the convolutional operations by reducing the redundant parameters in the neural network. FastDeepIoT [11] compresses the CNN models by optimizing the neural network configuration based on the non-linear relationship between the model architecture and the processing time. 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 [12] and many other works [13]–[15] 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 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, Cappuccino [16] optimizes computation by exploiting imprecise computation on the mobile system-on-chip (SoC). Oskouei *et al.* [17] developed an Android library called CNNdroid for running CNN models on mobile GPU. DeepMon [18] leverages GPU for continuous vision analysis on mobile devices. DeepX [19] 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.

III. Preliminary



(a) Processing Time Comparison (b) Accuracy Comparison

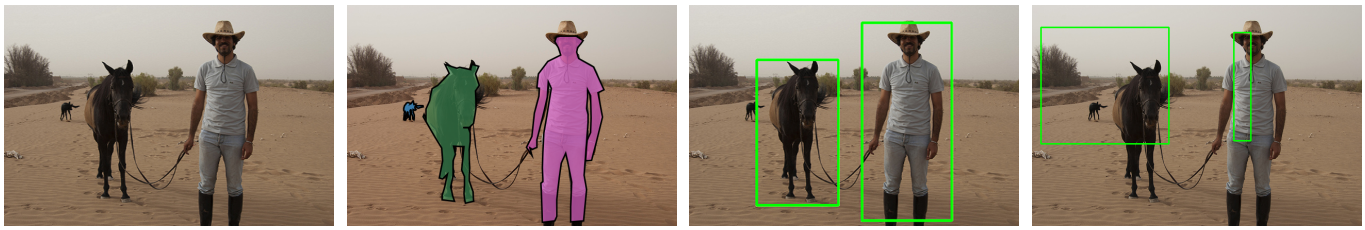
Fig. 1: Performance Comparison of NPU and CPU.

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 between accuracy and processing time. More importantly, the decision depends on how various CNN models perform on NPUs in terms of accuracy and processing time. In this section, we first show some results on how various CNN models perform on NPUs and local CPUs to understand the characteristics of NPU, and then give an overview of the proposed FastVA framework.

A. 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 [20] which is used for face recognition. In the experiment, we use the face images from the LFW dataset [21].
- The ResNet-50 model [8] which is used for object recognition. The evaluation was based on 4000 object images randomly chosen from the VOC dataset [22], and the results were based on the Top-1 accuracy.
- The YOLO Small model [23] which is designed for detecting objects in an image. The evaluation was based



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

Fig. 2: Detection result with YOLO Small

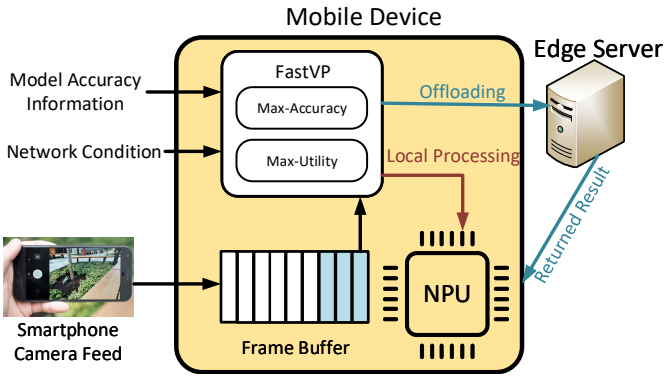


Fig. 3: FastVA overview

on 100 images randomly chosen from the COCO dataset [24], 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 Samll. This is because different CNN models have different ways to process these images.

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 pth 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 qth 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.

B. 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 at the cost of accuracy, and a larger model can increase the accuracy at the cost of longer processing time. If several CNN models are available, FastVA will choose the proper one for processing under different constraints. Based on the network condition and the accuracy of the CNN models, the schedule decision will be determined by the proposed Max-Accuracy or Max-Utility. To provide real time video analytics, FastVA ensures that the processing of each video frame is completed within a time constraint.

Due to the limited computational and bandwidth resources, the scheduling needs to be designed carefully in order to maximize the accuracy or utility. In the following sections, we formulate and solve the Max-Accuracy problem and the Max-Utility problem.

IV. 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.

A. 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 jth model on NPU
T_j^o	Processing time using the jth model on the server
$a(j, r)$	The accuracy of the jth model with input images in resolution r
T_c	Communication delay between mobile device and server
B	upload bandwidth (data rate)
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 I: Notation.

Assume the incoming frame rate is f, the time interval between two consecutive video frames is $\gamma = \frac{1}{f}$. For the ith 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 jth 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 denote the upload bandwidth and let T_c denote the communication delay between the edge server and the mobile device. Then, it takes $\frac{S(I_i,r)}{B} + T_j^o + T_c$ to transmit the ith 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 I. The Max-Accuracy problem can be formulated as an integer programming in the following way.

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

$$\text{s.t.} \quad \sum_{k=i'}^{i}\sum_{j} T_j^{npu} X_k^j + i'\gamma \le T + i\gamma,\ \forall i, i', i' \le i \tag{2}$$

$$D(i', i) + T_c \le i\gamma + T,\ \forall i, i', i' \le i \tag{3}$$

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

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

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

Where $D(i', i) = i'\gamma + \sum_j(\sum_r \sum_{k=i'}^{i} \frac{S(k,r)Y_k^r}{B} + T_j^o X_i^j)$ is the offloading time for the frames that arrive between $I_{i'}$ 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 jth model. If $X_i^j = 1$, the frame I_i is run by the jth 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.

B. 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 $t + T_j^o + T_c \le T$ **then**
      Add $(0, j, r)$ to S'
      $A' \leftarrow A' + a(j, r)$
      break
  $n_l \leftarrow \lfloor \frac{S(I_i,r)}{B\gamma} \rfloor$
  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.

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 $\frac{S(I_0,r)}{B} + T_c + T_j^o \leq T$ 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, $n_l = \lfloor \frac{S(I_0,r)}{B\gamma} \rfloor$ 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.

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:

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

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 jth model with resolution r. The running time of our algorithm is $O(n_r * n_c * n)$

V. 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.

A. 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.

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

$$\text{s.t.} \quad \sum_{k=i'}^{i} \sum_j T_j^{npu} X_k^j + i'\gamma \leq i\gamma + T, \forall i, i', i' \leq i \tag{10}$$

$$D(i', i) + T_c \leq i\gamma + T, \forall i, i', i' \leq i \tag{11}$$

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

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

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

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.

B. Max-Utility Algorithm

Since the problem is NP-hard, it costs too much time to find the optimal solution. Therefore, 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 $\frac{S(I_0,r)}{B}$, which means the frames are offloaded at the frame rate $\frac{B}{S(I_0,r)}$. The schedule decision for I_0 is made by solving $\max_{r,j} \frac{B}{S(I_0,r)} + \alpha \times a(j,r)$. Since the result from the server should be received within the time limitation, the constraint $T \geq \frac{S(I_0,r)}{B} + T_c + T_j^o$ should be satisfied. $n_l = \lfloor \frac{S(I_0,r)}{B} \rfloor$ 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.

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 kth 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 jth 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 $O(n^2 * n_c)$.

Algorithm 2: Max-Utility Algorithm

Data: Video frames in the buffer
Result: Scheduling decisions

1 The frame schedule list $S \leftarrow \{\}$
2 $u \leftarrow 0$
3 **for** *j from 1 to n_s* **do**
4 **for** *each possible resolution r* **do**
5   $u' \leftarrow \frac{B}{S(I_0,r)} + \alpha \times a(j,r)$
6   **if** $\frac{S(I_0,r)}{B} + T_j^o + T_c \leq T$ *and* $u < u'$ **then**
7    $p \leftarrow (0, j, r), u \leftarrow u'$
8 Add p to S
9 $n_l \leftarrow \lfloor \frac{S(I_0,r)}{B} \rfloor$, $U(0) \leftarrow \{(T^{idle}, 0, 0)\}$
10 **for** $i \leftarrow 1$ *to* n_l **do**
11 **for** *each* $(t, u, m) \in U(i-1)$ **do**
12   Add (t, u, m) to $U(i)$
13   **for** *each local model j* **do**
14    $t' \leftarrow \max(t, i\gamma) + T_j^{npu}$
15   **if** $t' \leq T + i\gamma$ *and* $i\gamma < t$ **then**
16     $A \leftarrow \frac{m}{m+1}(u - \frac{m}{n_l \gamma}) + \alpha \frac{a(j, r_{max})}{m+1}$
17   Add $(t', A + \frac{m+1}{n_l \gamma}, m+1)$ to $U(i)$
18 Remove the dominated pairs from U(i)
19 $(t', u', m') \leftarrow \arg\max_{(t,u,m) \in U(n_l)} u$
20 **for** *i from $n_l - 1$ to 0* **do**
21 **for** *each pair (t, u, m) in $U(i)$* **do**
22   **for** *each local model j* **do**
23    $A \leftarrow \frac{m}{m'}(u - \frac{m}{n_l \gamma}) + \alpha \frac{a(j, r_{max})}{m'}$
24   **if** $t + T_j^{npu} = t'$ *and* $A + \frac{m'}{n_l \gamma} = u'$ **then**
25   Add $(i+1, j, r_{max})$ to S
26     $t' \leftarrow t, u' \leftarrow u, m' \leftarrow m$
27   **break**
28 **return** S

VI. Performance Evaluations

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

A. 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 [25] 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 [26]. 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. 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.

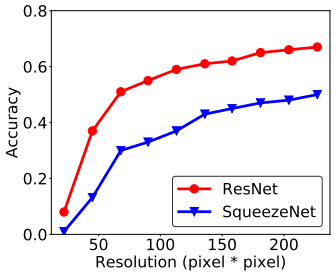


Fig. 4: Accuracy vs. Resolution.

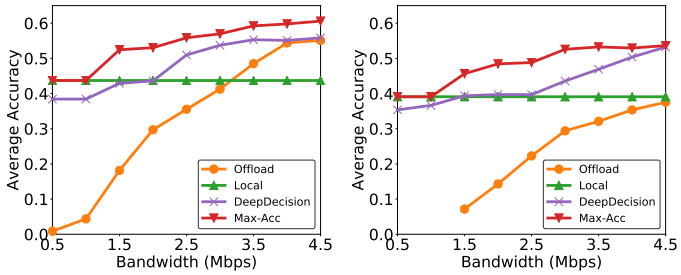
In the experiment, object classifications are performed on mobile devices, which are common computer vision tasks for many mobile applications. In our experiment, two different object recognition CNN models are used, ResNet-50 [8] and SqueezeNet [27], which are well known and are widely used. Moreover, 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 [28], 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).

B. 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 II, 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 communication delay between the server and mobile device.



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

Fig. 5: The performance of different methods under different network conditions.

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 4 shows the tradeoff between accuracy and resolution. We note that the accuracy does not scale linearly with the resolution. The data in Table II and Figure 4 are used for making scheduling decisions in FastVA.

C. The Performance of Max-Accuracy

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

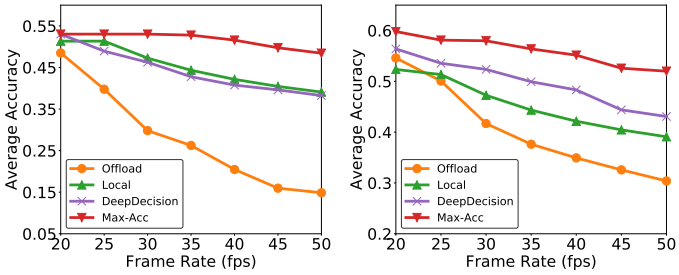
- **Offload:** In this method, all frames must be offloaded to the edge server for processing. Each frame will be resized to a resolution so that it can be offloaded before the next frame arrives, and the server chooses the most accurate model that can process the frames and return the result within 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, delay and the processing time requirement specified by the applications.

In Figure 5, we compare Max-Accuracy with the Local and Offload method under different network conditions. In the evaluation, we set the frame upload delay to be 100 ms. The Local method does not offload any frames, and thus its performance remains the same under different network

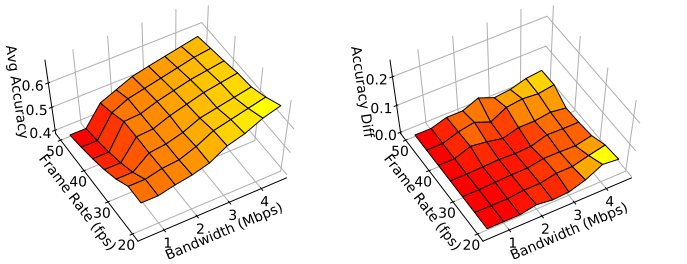
CNN model		Processing Time (ms)	Transmission Time (ms)	Top-1 Accuracy
ResNet	Local	52	0	0.52
	Server	69	39 - 242	0.67
SqueezeNet	Local	17	0	0.41
	Server	9	39 - 242	0.51

TABLE II: The performance of the CNN models.



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

Fig. 6: The performance of different methods under different frame rate requirements.



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

Fig. 7: Comparison between optimal and Max-Accuracy

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 5(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 4, 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 Offload become smaller since the mobile device can offload most of the frames in high resolution and achieve better accuracy.

In Figure 6, 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 7, we compare Max-Accuracy with the Optimal method under various frame rates and network conditions. As shown in Figure 7(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.

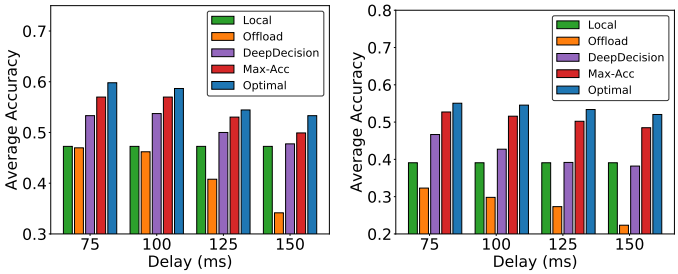
In figure 7(b), we plot the accuracy difference between Optimal and Max-Accuracy. The accuracy difference is computed using the accuracy of the Optimal method minus that of Max-Accuracy. As can be seen from the figure, the difference is almost 0 in most cases, which indicates that Max-Accuracy is close to Optimal.

In Figure 8, we evaluate the impact of frame upload delay 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 upload delay increases. Compared to DeepDecision, Optimal and Max-Accuracy, a significant accuracy drop can be observed in the Offload method, when the upload delay 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 upload delay. Although the accuracy is also dropped by processing the frames on NPU, the impact is not as significant as that in the Offload method.

D. The Performance of Max-Utility

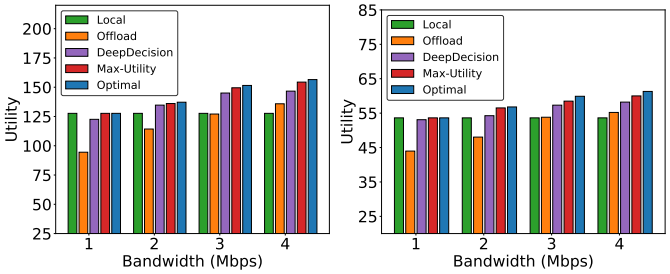
To evaluate the performance of Max-utility, we still compare it to Offload, Local, and Optimal. Since we focus on utility instead of accuracy in this subsection, these algorithms are also modified to maximize utility instead of accuracy.

In Figure 9, we evaluate the impact of network bandwidth for different methods. In the comparison, the frame rate and the upload delay 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



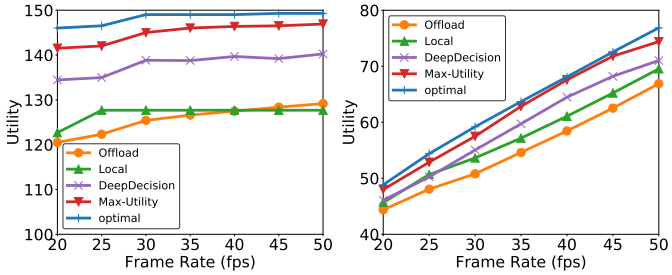
(a) Frame Rate 30 fps (b) Frame Rate 50 fps

Fig. 8: The performance of different methods under different frame upload delay.



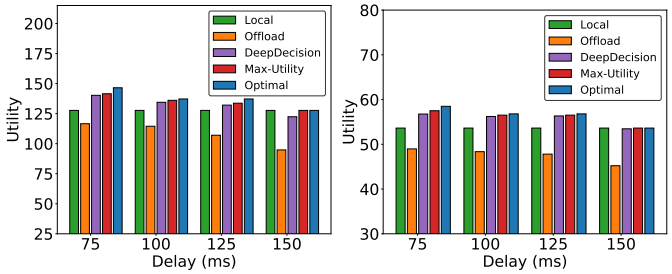
(a) $\alpha = 200$ (b) $\alpha = 50$

Fig. 9: The impact of network bandwidth.



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

Fig. 10: The effects of frame rates.



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

Fig. 11: The effects of upload delay

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 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 9(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 9(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 10 shows the impacts of frame rate for different methods. In the evaluation, we set the network bandwidth to be 2.5 Mbps and the upload frame delay to be 100 ms. As shown in the figure, Max-Utility outperforms Offload, Local and DeepDecision methods. When α is small, as shown Figure 10(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 10(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 11 shows the impact of upload delay 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 upload delay increases, less video frames can be offloaded to the server due to time constraints, and hence degrading the performance of the Offload, DeepDecision, Max-Utility, and Optimal algorithms.

VII. 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 two problems: *Max-Accuracy* where the goal is to maximize the accuracy under some time constraints, and *Max-Utility* where the goal is to maximize the utility which is a weighted function of processing time and accuracy. To solve these two 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.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *IEEE ICCV*, 2015.

[2] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," *ACM Sensys*, 2015.

[3] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, "MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency," *ACM Sensys*, 2018.

[4] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," *IEEE ICDCS*, 2017.

[5] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," *IEEE INFOCOM*, 2018.

[6] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," *ACM Mobisys*, 2016.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *IEEE CVPR*, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE CVPR*, 2016.

[9] S. Bhattacharya and N. D. Lane, "Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables," *ACM Sensys*, 2016.

[10] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," *IEEE CVPR*, 2015.

[11] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices," *ACM Sensys*, 2018.

[12] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," *ACM Mobisys*, 2010.

[13] Y. Geng, W. Hu, Y. Yang, W. Gao, and G. Cao, "Energy-efficient computation offloading in cellular networks," *IEEE ICNP*, 2015.

[14] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," *IEEE INFOCOM*, 2018.

[15] Y. Geng and G. Cao, "Peer-assisted computation offloading in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 7, pp. 4565–4578, 2018.

[16] M. Motamedi, D. Fong, and S. Ghiasi, "Cappuccino: efficient CNN inference software synthesis for mobile system-on-chips," *IEEE Embedded Systems Letters*, 2019.

[17] 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.

[18] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," *ACM Mobisys*, 2017.

[19] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," *IEEE IPSN*, 2016.

[20] A. Z. O. M. Parkhi, A. Vedaldi, "Deep Face Recognition," *British Machine Vision Conference*, 2015.

[21] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments," University of Massachusetts, Amherst, Tech. Rep., 2007.

[22] E. Mark, E. S. Ali, V. Luc, W. C. KI, W. John, and Z. Andrew, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, 2015.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *IEEE CVPR*, 2016.

[24] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *European conference on computer vision*, 2014.

[25] "HiAI." [Online]. Available: https://developer.huawei.com/consumer/en/devservice/doc/2020315

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *ACM International Conference on Multimedia*, 2014.

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

[28] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang, "Exploiting Feature and Class Relationships in Video Categorization with Regularized Deep Neural Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.