ResMap: Exploiting Sparse Residual Feature Map for Accelerating Cross-Edge Video Analytics

Ning Chen[#], Shuai Zhang[#], Sheng Zhang[*], Yuting Yan, Yu Chen, and Sanglu Lu
State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
Email: {ningc, zhangshuai.cs, yuting.yan, DZ1933005}@smail.nju.edu.cn, {sheng, sanglu}@nju.edu.cn

***Abstract*— Deploying deep convolutional neural network (CNN) to perform video analytics at edge poses a substantial system challenge, as running CNN inference incurs a prohibitive cost in computational resources. Model partitioning, as a promising approach, splits CNNs and distributes them to multiple edge devices in closer proximity to each other for serial inferences, however, it causes considerable cross-edge delay for transmitting intermediate feature maps. To overcome this challenge, we present ResMap, a new edge video analytics framework that significantly improves the cross-edge transmission and flexibly partitions the CNNs. Briefly, by exploiting the *sparsity* of the intermediate raw or residual feature map, ResMap effectively removes the redundant transmission, thereby decreasing the cross-edge transmission delay. In addition, ResMap incorporates an *Online Data-Aware Scheduler* to regularly update the CNN partitioning scheme so as to adapt to the time-varying edge runtime and video content. We have implemented ResMap fully based on COTS hardware, and the experimental results show that ResMap reduces the intermediate feature map volume by 14.93-46.12% and improves the average processing time by 17.43-30.6% compared to other alternative designs.**

***Index Terms*—edge video analytics, cross-edge CNN inference, residual feature map, model partitioning**

I. Introduction

Video analytics is becoming the modern solution to amounts of critical scenarios. From augmented reality application [1], [2] and mobile game, to autonomous vehicles, cognitive assistance and surveillance systems, mobile edge devices capture and analyze frames from video streams to derive helpful information. The emerging mobile edge computing [3]–[5] enables the edge infrastructures to be provisioned with considerable computing resources, which can thus potentially support *edge video analytics* using convolutional neural networks (CNNs).

Considering the prohibitive overhead in computation and memory of running CNN inference purely on single device, the *model partition* [12]–[19], which splits the model and distributes the slices to multiple edge device in closer proximity to each other, makes full use of the resources of involved devices to coherently complete the inference. Therefore, the multi-edge collaborative method is widely applied to solve the edge video analytics tasks. Fig. 1 illustrates such a system in a multi-edge environment. An k-layer CNN model is partitioned into three slices, which are scheduled to three edge devices physically close. For an input frame, it sequentially runs the concrete CNN inferences at each edge device, transmits its intermediate outcomes (i.e., feature map) to the next device, and outputs the analytics results in the final device.

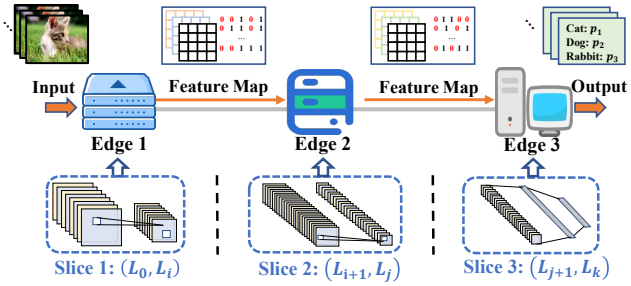


Fig. 1. Multi-edge collaborative video analytics system.

It is, however, non-trivial to optimally deploy such multi-edge collaboration video analytics system, since it often requires the cross-edge management of many factors, such as the provisioning of edge resources, data transference among devices, and the execution status of frames that are being processed at each device. In fact, it faces two critical challenges:

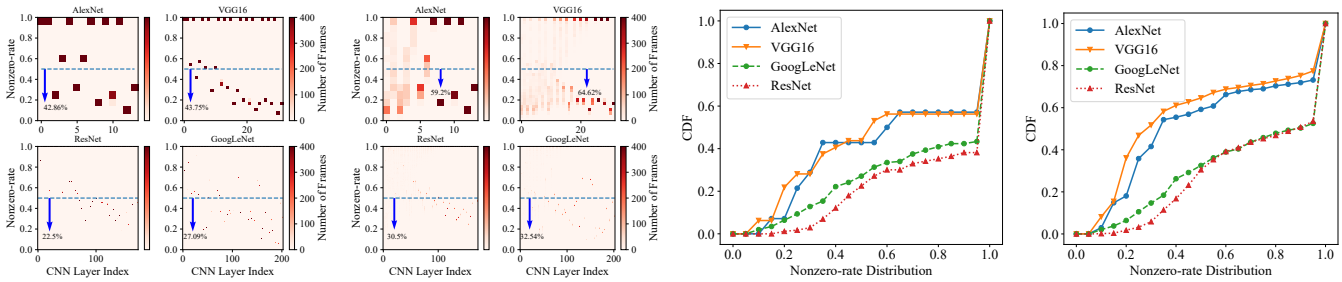
First of all, the intermediate feature maps for successive inference are commonly considerably large, and transferring them to the next device probably incurs huge transmission delay. In general, CNN layers are often arranged as a *DAG* (directed acyclic graph), and one device has to delivers its output to the next one for subsequent inference, thus causing an extra transmission delay. This challenge escalates when the model is complex, the number of involved edge devices is large, or the inter-edge network connections are unreliable.

Second, both real-world edge platforms and video contents are intrinsically dynamic and uncertain, and fixing the model partitioning scheme possibly derives diminishing video analytics performance. In fact, for the newly arrival video, the model partitioning scheme needs to be tuned before knowing the dynamic status regarding the edge environment, and such status include time-varying volumes of intermediate feature map at each layer, as well as the completion times of frames that are being processed. One may use the historical values to predict the inputs and control the scheme based on such estimations; however, it can hardly ensure the quality of this predictor, compared to the offline optimum in hindsight.

Existing researches fall insufficient for addressing the aforementioned challenges. Some [12]–[14] have considered the effective model partitioning based on the devices' computing capability and inter-device bandwidth, but they do not optimize the feature map transmission. Others [15]–[17] use adaptive

[#] These authors contribute to this work equally and are co-first authors.

[*] The corresponding author is Sheng Zhang (sheng@nju.edu.cn). This work was supported in part by NSFC (61872175, 61832008), the Fundamental Research Funds for the Central Universities (2022300297), and Collaborative Innovation Center of Novel Software Technology and Industrialization.



(a) Case for raw feature map (b) Case for residual feature map (c) CDF of raw feature map (d) CDF of residual feature map

Fig. 2. Sparsity of the raw and the residual feature map.

compression ratio or feature pruning techniques to reduce the intermediate data volume, which probably results in a decrease in analytics accuracy. Several researches [18], [19] present the iterative alternating optimization to derive the optimal partitioning scheme from the large solution space, which incurs a considerable profiling delay. The rest [20]–[23] studies the model partitioning in edge-cloud platform, but does not investigate the edge-only collaborative inference.

In this paper, the following two observations motivate us to overcome the above challenges: (1) the large intermediate feature map of each layer, however, is commonly sparse (see Section II); (2) The time-varying sparsity of each feature map can be efficiently predicted (see Section IV-B), which facilitates online model partitioning. Therefore, we carefully design three functional modules to optimize both the cross-edge feature map transmission and model partitioning:

Feature map sparse encoder. The intermediate feature map, however, contains substantial useless or redundant values (i.e., 0). Hence for each path of a feature map, we use the *nonzero-rate* to measure the *sparsity*, and once the *nonzero-rate* meets the given threshold, we adopt *sparse encoding* to compress it. In addition, drawing from the fact that successive frames show significant similarity, we obtain the potentially sparser *residual feature map by subtracting the feature map of the last frame from that of current frame*. At every transferring, if the one that has a lower *nonzero-rate* and satisfies the threshold from these two feature maps, we use *sparse encoding* to compress it. Otherwise, we transmit the raw feature map.

Intermediate data predictor. The volume of each feature map outputted by each CNN layer, which is of vital importance for making efficient model partitioning scheme for the newly arrival video, is however not available until the concrete inferences have done. In this case, *we turn to profile the correlations between the known volume of raw or residual frame and the unknown volumes of all the subsequent layers*. We experimentally show that for each type of layer, there exists a functional relation (e.g., *logistic* for convolutional layer and *linear* for ReLU and MaxPool layer). Thus, we directly estimate the volume of each layer from the volume of raw or residual frame and ensure the quality of this predictor.

Online data-aware scheduler (DAS). Based on the intermediate data volume estimation of each layer, the online *DAS* is proposed to advance completion time for the newly arrival video by tuning the model partitioning scheme. To avoid the buffer invalidation due to frequent scheme updates, *DAS* makes decisions based on video chunk granularity, that is, the frames in the same chunk share the same execution plan. In this case, until the previous video chunk has been executed, the next chunk is submitted to the pipeline system. *DAS* first simulates the execution of the previous chunk and gather its completion times at each stage, which are also the ready time for the next chunk. Then, it runs the *Data-Aware DP* algorithm to derive the scheme with an estimated earliest completion time, and takes this as the latest partition strategy.

To this end, we present ResMap, a new edge video analytics system that significantly enhances the intermediate feature map transmission and model partitioning. Briefly, for the newly arrival video chunk, ResMap first uses the *Intermediate Data Predictor* to estimate the feature map volume produced by each layer. Based on this estimation, ResMap incorporates an online *DAS* to remake the model partitioning scheme. After that, ResMap distributes each model slice to the involved edge devices for concrete inference. Once one device has done the inference, ResMap uses the *Sparse Encoder* to compress the outputted feature map and transmits it to the next device for running next model slice.

We have implemented ResMap and deployed it over three 4B Raspberry Pis. In addition, we have built 4 typical models including AlexNet, VGG16, GoogLeNet, and ResNet in it. The experimental results measured in VIRAT dataset [24] show that ResMap reduces the intermediate feature map volume by 14.93-46.12% and improves the average processing time by 17.43-30.6% compared to other alternative designs. We summarize our major contributions of this work as follows:

- Reduction for cross-edge data transmission. By exploiting the sparsity of the raw or residual feature map, ResMap largely reduces the transmission data volume.
- Optimization for model partitioning. By accurately predicting the future intermediate volume of each layer through online *DAS*, ResMap significantly improves the efficiency of model partitioning.
- Improvement for resource utilization. Pipeline execution enables to make full use of the edge devices resources such as GPU, memory, and inter-device bandwidth.

II. Motivations and Insights

This section exhibits our motivations to propose ResMap. As stated before, the cross-edge collaborative CNN inference accelerates the video analytics, however, generating large volume of intermediate feature map for subsequent inferences.

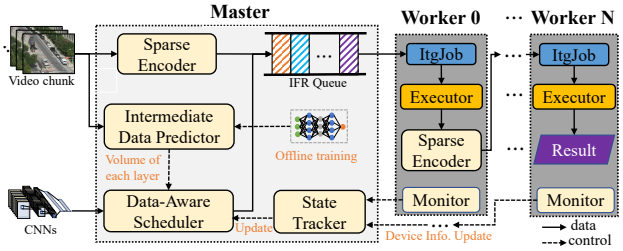


Fig. 3. The architecture of ResMap.

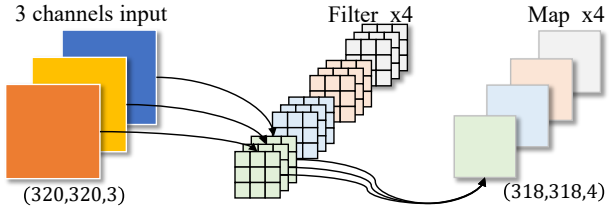


Fig. 4. Feature maps in CNN inference.

Directly delivering such raw feature map probably incurs excessive delay. Hence we desire to improve the feature map transmission by exploiting its structural information. Preliminarily, we use the road video with a total of 400 frames from VIRAT dataset [24] to acquire the intermediate feature maps of four typical CNNs including AlexNet, VGG16, GoogLeNet, and ResNet that consist of 14, 32, 173, 203 layers respectively.

Case for the raw feature maps. In general, several two-dimensional matrixes make up a feature map. In many cases, a feature map contains considerable duplicate values (e.g., 0 in this paper). Hence an intuitive idea is to only transmit the non-duplicate value, but fill back the feature map with the duplicate value afterwards. Without loss of generality, we use *nonzero-rate* to measure the sparsity for a given feature map. We show the *nonzero-rate* of each frame at each layer in the heat map Fig. 2(a), and a region with deeper color indicates that more samples fall on it. It is clear that 42.86% feature maps have a *nonzero-rate* lower than 0.5 for AlexNet, and 43.75%, 22.5%, and 27.09% for VGG16, ResNet, and GoogLeNet respectively. What's more, we observe that for a fixed layer, its feature maps show almost the same *nonzero-rate*. For instance, feature maps generated by layer 2 (or 5, 8, 10, 12, 13) in AlexNet have highly similar *nonzero-rates*. Other CNNs present the similar rules. We also depict the *nonzero-rate* distribution in Fig. 2(c). Despite many sparse samples, most of the feature maps (over 50% for AlexNet and VGG16, and 70% for ResNet and GoogLeNet) present high *nonzero-rates*. For these samples, we consider another case to further lower their *nonzero-rates*.

Case for the residual feature maps. It's common that the successive frames captured by stationary camera present the similar contents. For instance in a road surveillance video, it shows few scene-changes in the background, and especially at night there are almost no cars on the road. The adjacent frames share abundant pixel values, thus probably producing the similar feature maps. Specifically, *we acquire the residual feature map by subtracting the feature map of the last frame from that of the current frame.* Then we present the resulting *nonzero-rate* in Fig. 2(b). Fortunately, 59.2% feature maps have a *nonzero-rate* lower than 0.5 for AlexNet, and 64.62%, 30.5%, and 32.54% for VGG16, ResNet, and GoogLeNet respectively. Compared to the raw feature map, the *nonzero rate* distribution of residual feature maps is much more decentralized. It is clear that in Fig. 2(d) the residual feature map further improves the sparsity of intermediate data.

To sum up, the intermediate raw or residual feature maps, however, are commonly sparse, thus exploiting such sparsity is promising to reduce the transmission volume. Specifically, when an edge device runs the concrete inference for the specified model partitions and outputs the feature map, it first evaluates the sparsity of this raw or its residual feature map, then if the one from these two feature maps has a lower *nonzero-rate* and meets the threshold, it uses *sparse encoding* to compress it and transmits it to the next device. Otherwise, it delivers the raw feature map. In addition, once the next device receives the residual feature map, it extracts the buffered result of last frame and generates the raw feature map for inference.

III. ResMap Overview

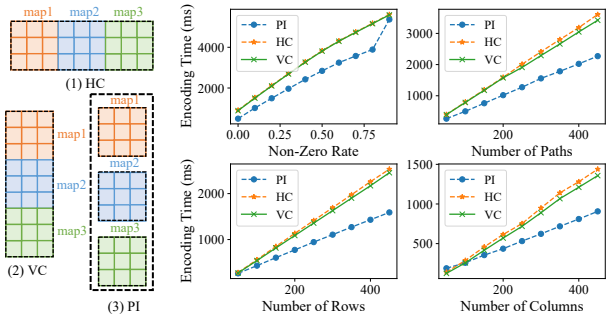
The goal of ResMap is to accelerate the cross-edge collaborative video analytics on multiple resource-constrained edge devices by efficient CNNs partitioning and intermediate feature map reduction. Typical CNNs including AlexNet, VGG, GoogLeNet, and ResNet have been built in ResMap, users are capable of defining their own CNNs with the APIs provided by ResMap. Fig. 3 presents the architecture of ResMap. There are two roles for the involved devices, i.e., *Master* and *Worker*. Briefly, the *Master* is responsible for model partitioning, while each *Worker* executes the concrete inference tasks and transmits the encoded feature map to the next *Worker*.

Master. On one hand, as the global coordinator, it involves a *State Tracker* to continuously gather the runtime system status. On the other hand, for the newly arrival video chunk, the *Intermediate Data Predictor* estimates its feature map volume at each CNN layer. Based on the above information, the *Data-Aware Scheduler*, which is pre-trained in an offline manner, makes effective partitioning scheme for the given model, and generates an *IFR* file for each frame. Each initial *IFR* consists of the layers index and the initial input for the first *Worker*.

Worker. Once it receives an *IFR*, the *Merger* decodes the input, and regenerates the complete feature map. Then, the *Executor* runs concrete inference for the layers specified in the *IFR* and produces a new feature map. The *Sparse Encoder* compresses this intermediate feature map based on its sparsity, updates the *IFR* file and transfers it to the next *Worker*. The successive *Workers* repeat these operations one by one, and the final *Worker* directly outputs the analytics results. Each *Monitor* persistently tracks the *IFR* status, collects the runtime information, and feeds them back to the *Master*.

IV. ResMap Design

ResMap significantly reduces the intermediate feature map volume by exploiting the *Sparse Encoder*, and periodically tunes an effective model partitioning scheme that's completely adapted to the future knowledge, including the estimated data



(a) Methods (b) Encoding efficiency of three methods.

Fig. 5. Evaluations of three methods based on CSR.

volume of each layer from *Transmission Predictor*, and the completion time of each runtime frame from *State Tracker*. This section details the design of *Sparse Encoder, Intermediate Data Predictor, and Data-Aware Scheduler.* Finally, we illustrate the workflow of cross-edge collaboration inference based on the partitioning scheme.

A. Feature Map Sparse Encoder

For each frame to be analyzed, it goes through multiple convolution and pooling operations, and produces a feature map once it finishes a layer inference. Note that, given a concrete model that specifies the number of filters and their sizes for each layer, the shape of output feature map is also determined. For instance in Fig. 4, feeding a frame with shape 320x320x3 to the convolution layer with 4 3x3x3 filters, it outputs 4 maps with shape 318x381, which form the full feature map. Hence, for any input, we can easily acquire the feature map size of each layer. The widely used *sparse encoding* methods for two-dimension (2D) matrix consists of *CSR* (Compressed Sparse Row) and *CSC* (Compressed Sparse Column) that compress the matrix by row and column respectively. The width of captured videos is commonly larger than the height, thus using the *CSR* approach probably benefits more. Considering a feature map consists of several 2D maps, we propose three methods to encode the multi-path feature map as shown in Fig. 5(a):

- Horizontal or Vertical Combination (*HC* or *VC*). We merge all the 2D maps horizontally or vertically, and run *CSR* encoding to compress the aggregated 2D matrix.
- Path Independently (*PI*). We perform *CSR* encoding to compress each 2D map and put the results together.

Fig. 5(b) illustrates the efficiencies of these three methods. It is clear that no matter how we tune the number of paths, rows and columns, *PI* outperforms other schemes in the encoding time. Hence we employ *PI* method to encode the feature map.

B. Intermediate Data Predictor

Problem and goal. Unlike single-device inference, multi-edge collaborative inference experiences a cross-edge feature map transmission stage, and commonly the data volume is substantially huge. Scheduling the layers with huge outputs to the involved devices probably incurs a significant transmission time, thereby largely increasing the average completion time. The *Sparse Encoder* is proposed to reduce the feature map

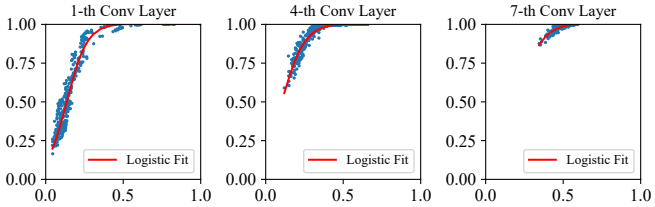


Fig. 6. Relation between the first layer and Convolution layer in AlexNet.

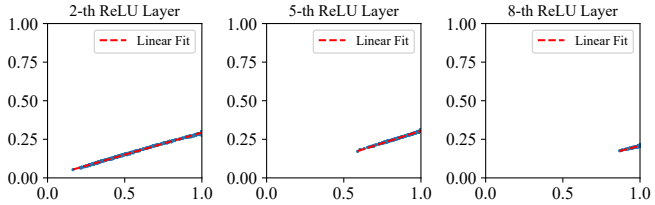


Fig. 7. Relation between the first layer and ReLU layer in AlexNet.

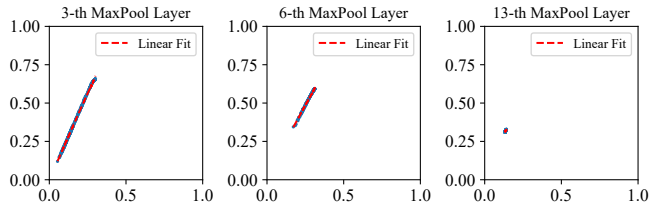


Fig. 8. Relation between the first layer and MaxPool layer in AlexNet. The x-axis shows the *nonzero-rate* of output residual feature map $O_{f,0} - O_{f-1,0}$ of frame f and y-axis indicates the *nonzero-rate* of $O_{f,l} - O_{f-1,l}$.

volume to some extends, but the volume is still considerable and varies with the sparsity measured by *nonzero-rate*. What's worse, for the newly arrival video chunk, we can hardly get their *nonzero-rates* of each layer in advance until they have executed the corresponding layers, and thus the intermediate feature map volumes of each layer are also unavailable.

Method. In this case, we exploit a predictive approach to estimate the *nonzero-rate* of each layer. Specifically, we first present the *sparsity* estimation for each layer, and then derive the intermediate data volume prediction.

1) Feature map sparsity prediction: To accurately estimate the feature map volume, we first predict its sparsity. A CNNs comprises multiple layers, which are organized as a DAG (Directed Acyclic Graph), and a frame is processed at each layer in the topological order of this DAG. We denote the output feature map of frame f at layer l by $O_{f,l}$. To simplify the description, $O_{f,0}$ indicates the raw input frame. Based on Section II, we observe that both the raw feature map $O_{f,l}$ and the residual feature map $O_{f,l} - O_{f-1,l}$ are probably sparse.

Case for raw feature map. As Fig. 2(a) illustrates, for a CNN layer, the resulting *nonzero-rates* for whatever inputs are highly concentrated in a certain interval. Take AlexNet as an example, its *nonzero-rates* in several layers (e.g., 2, 3 and 5) are distributed narrowly around a specific value. Thus for each layer of a given CNN, we directly use the average *nonzero-rate* of the total samples as the sparsity estimation.

Case for residual feature map. As shown in Fig. 2(b), for a specific layer, its *nonzero-rates* with different inputs are distributed asunder. Hence, we make the sparsity estimation for each frame. We detail the methodology as follows.

In general, receiving the input $O_{f,0}$, we easily acquire $O_{f,0} - O_{f-1,0}$, while the residual outputs of the subsequent

Fig. 9. Methodology of *Intermediate Data Predictor*.

layers $(O_{f,1} - O_{f-1,1}, O_{f,2} - O_{f-1,2}, \cdots)$ are not generated until the corresponding inferences are finished. To support the online scheduling for the arrival frames, a promising method is to estimate the sparsity of each layer in advance purely based on the input $O_{f,0} - O_{f-1,0}$, and further obtain the approximate transmission volume. Take the AlexNet as an example, we show the relations between the subsequent layers and the first layer in terms of *nonzero-rate* in Figs. 6, 7, and 8. The x-axis shows the *nonzero-rate* of output residual feature map $O_{f,0} - O_{f-1,0}$ of frame f and y-axis indicates the *nonzero-rate* of $O_{f,l} - O_{f-1,l}$. For the Convolution layers, their *nonzero-rates* are Logistic relationship with the *nonzero-rate* of the raw frame or its residual frame. While there is an apparent positive linear correlation for the ReLU and MaxPool layers.

We exploit these facts to effectively predict the *nonzero-rate* for each layer. Specifically, for the Convolution layers, we use a multi-layer perceptron (MLP) with a hidden layer and take *Logistic* as the activation function, and for the ReLU and MaxPool layers, we directly use a linear function to fit the relation between the estimation and the input *nonzero* in the first layer. As stated before, we employ *CSR* method to encode the multi-path feature map using *PI* method, hence we predict the *nonzero-rate* for each path. Besides these basic layers, some complex CNNs probably comprise layers with multiple direct predecessors such as the *Inception* layer in GoogLeNet that splices the received feature maps. For these layers, we first estimate the *nonzero-rates* of their predecessors, and use the weighted sum of them to indicate their *nonzero-rates*.

2) Transmission volume estimation: Having obtained the sparsity information, we are capable of calculating the data volume of each layer. Suppose that for layer L, the $H_L \times R_L \times C_L$ feature map comprises H_L paths, and each path corresponds to a $R_L \times C_L$ 2D matrix. We denote by $\mathbf{z}_h \in Z_L$ the *nonzero-rate* of path h. In CSR encoding, three arrays, i.e., *indptr, indices, and data,* are established to store each 2D matrix. Specifically, *indptr* maintains the number of nonzero elements of each row, *indices* is to record the row index of each nonzero element, and *data* stores all the nonzero elements. The demanded lengths of these arrays are $R_L + 1$, $R_L C_L \mathbf{z}_h$ and $R_L C_L \mathbf{z}_h$ respectively. Hence the total number of elements D_L using CSR method is calculated as

$$D_L = \sum_{h=1}^{H_L} \left((2R_L C_L \mathbf{z}_h + R_L + 1)\, \mathbb{I}(\mathbf{z}_h < \delta) + R_L C_L \mathbb{I}(\mathbf{z}_h \geq \delta) \right), \quad (1)$$

where δ is the threshold to measure the sparsity, $\mathbb{I}(\cdot)$ is the indicator function and it returns 1 if the condition holds, otherwise it returns 0. To sum up, Fig. 9 shows the methodology of intermediate feature map volume prediction for a CNN with N layers. It first acquires the initial Z_0 of $O_{i+1,0} - O_{i,0}$, then derives all the *nonzero-rates* of successive layers, finally estimates the approximate volumes under *CSR* encoding.

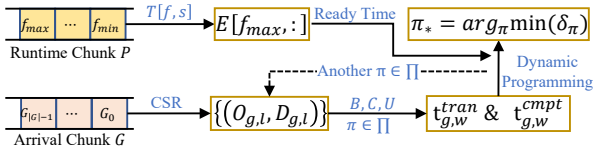


Fig. 10. Workflow of *Online Data-aware Scheduler (DAS)*.

TABLE I
SUMMARY OF NOTATIONS IN ALG. 1

Inputs	Description
G	The newly arrival video chunk
N, M	The number of layers and *Workers*
P	The frames that are being executed in the pipeline
$T[f,s] \in \mathcal{T}$	Completion time of completed stage s of frame $f \in P$
$O_{i,j} \in \mathcal{O}$	Raw feature map volume of frame i at layer j
$D_{i,j} \in \mathcal{D}$	Residual feature map volume of frame i at layer j
$b_w \in B$	Bandwidth between *Worker* w and *Worker* $w+1$
$u_i \in U$	Computation to run layer i
$\Omega_w \in \Omega$	The layers executed at *Worker* w
$c_i \in C$	Runtime GPU processing cycles of *Worker* i
Decisions	**Descriptions**
p_w	The layer index in front of the w-th partitioning

C. Online Data-aware Scheduler

As stated before, once a *Worker* receives the residual feature map $O_{f,l} - O_{f-1,l}$, it has to use the local buffered data (i.e., $O_{f-1,l}$) to generate the complete input $O_{f,l}$ for inference. In general, different partitioning scheme result in different buffered contents, hence frequently changing the scheme (e.g., every frame an update) probably invalidates the buffered data and seriously weakens the role of *Sparse Encoder*. In this case, the proposed online *Data-Aware Scheduler (DAS)* tunes the partitioning scheme at video chunk granularity. The goal of *DAS* is to derive the optimal partitioning scheme that largely advances the completion time for each arrival video chunk.

Alg. 1 illustrates the workflow of *DAS* and Tab. I summarizes the notations of inputs and decisions for Alg. 1, where $\mathcal{O}$ and $\mathcal{D}$ are derived from *Intermediate Data Predictor*, P, $\mathcal{T}$, B, U, Ω, and C are gathered by the *Monitor*, and fed back to *DAS*. In short, Alg. 1 goes through two steps, i.e., estimating the *ready time* (i.e., the time when the GPU and inter-device connection are released and ready for G) (lines 1-6) and profiling the optimal scheme with an earliest *completion time* (i.e., the time when the last layer is finished) (lines 7-23).

Ready time estimation. There are mainly two types of stages, i.e., cross-edge feature map transmission and on-device inference. To analyze a frame using M *Workers*, it experiences $M-1$ transmissions and M inferences, with a total of $2M-1$ stages. Considering the buffered content varies with the video chunk, the newly chunk G will not be executed until the previous chunk is totally finished. Hence, the ready time for chunk G is also the completion time of P. To estimate this ready time, we first figure out the head f_{min} and tail f_{max} frame in P, and the frames between these two frames have not been finished. Considering the sequential and non-preemptive processing in each stage, it is not until both the stage s of frame $f-1$ and stage $s-1$ of frame f are finished, the stage s of frame f starts to be executed. Based on this and the execution status T, we iteratively obtain the completion time of f_{max}, i.e., $T[f_{max},:]$, which is also the ready time for G.

Optimal scheme profiling. To derive the optimal scheme

Algorithm 1 Data-Aware Dynamic Programming Algorithm

Input: G, $\mathcal{O}$, $\mathcal{D}$, P, T, Ω, B, C, U, N, M
Output: Partitioning scheme $(p_1, \cdots, p_{M-1})$

$f_{\min}, f_{\max} \leftarrow$ the head and tail frame in P
for frame index $f = f_{\min}$ to $f_{\max}$ **do**
  for f's first unfinished stage s to $2m-1$ **do**
    $E[f,s] \leftarrow \max\{E[f-1,s], E[f,s-1]\} + T[f,s]$
Denote the ready time of all stages by $\mathbf{r} \leftarrow E[f_{\max}, :]$
Define the optimal scheme and its delay as π^*, $\delta_{\pi^*} \leftarrow \infty$
for all $\pi \in \{(p_1, \cdots, p_{M-1}) | 0 < p_1 \leq \cdots p_{M-1} \leq N\}$ **do**
  Worker $0 \rightarrow M-1$ execute layers $(0, p_1) \rightarrow (p_{M-1}, N)$
  Define the virtual buffer $\Omega' \leftarrow \Omega$
  $\forall g \in G$, calculate its inference and transmission delay
  $\mathbf{t}^{cmpt}_{g,w} \leftarrow \left(\sum_{i=p_w+1}^{p_{w+1}} \mathbf{u}_i\right) / \mathbf{c}_w, \mathbf{t}^{tran}_{g,w}$
  for $g = G_0$ to $G_{|G|-1}$ **do**
    for worker $w = 0$ to $M-1$ **do**
      if $D_{g-1,p_w} \in \Omega'$ **then**
        $\mathbf{t}^{\text{tran}}_{g,w} \leftarrow (D_{g,p_w}) / \mathbf{b}_w$
      else
        $\mathbf{t}^{\text{tran}}_{g,w} \leftarrow (O_{g,p_w}) / \mathbf{b}_w$, $\Omega'_w \leftarrow \{p_w\}$
    $t_{g,s} \leftarrow \mathbf{t}^{\text{tran}}_{g,s/2}$ if $2 \mid s$ else $\mathbf{t}^{cmpt}_{g,s/2+1}$
  Define the estimated delay $\tau[g,s]$
  For G_0, $\tau[G_0, 0] \leftarrow \mathbf{r}_0 + \mathbf{t}_{G_0,0}$
  for $s = 1$ to $2M-1$ **do**
    $\tau[G_0, s] \leftarrow \max\{\mathbf{r}_s, \tau[G_0, s-1]\} + \mathbf{r}_{G_0,s}$
  for $g = G_1$ to $G_{|G|-1}$ **do**
    $\tau[g,0] \leftarrow \tau[g-1,0] + \mathbf{t}_{g,0}$
  The finish time of G : $\delta_\pi \leftarrow \tau[|G|-1, 2M-1]$
  if $\delta_\pi \leq \delta_{\pi^*}$ **then**
    $\pi_* \leftarrow \pi$
$T, \Omega \leftarrow T \bigcup \mathbf{t}, \left\{\left\{p_1^{\pi^*}\right\}, \ldots, \left\{p_{M-1}^{\pi^*}\right\}\right\}$
return π_*

$\pi^* \in \Pi = \{(p_1, p_2, \cdots, p_{M-1})\}$ with the earliest completion time δ_{π^*}, *DAS* runs two-stages profiling. Firstly, based on the raw data volume $\mathcal{O}$ and estimations $\mathcal{D}$, it directly calculates the transmission delay $t^{tran}_{g,w}$ and inference delay $t^{cmpt}_{g,w}$ for frame $g \in G$ in *Worker* w. Secondly, for a scheme π, based on the ready time and estimated delay for each stage, *DAS* uses the idea of *Dynamic Programming* to obtain the completion time of frame $g \in G$ at stage s, i.e., $\tau[g,s]$, and δ_π of this chunk G is the completion time the last stage $2M-1$ of frame $|G|-1$, i.e., $\tau[|G|-1, 2M-1]$. We traverse each scheme $\pi \in \Pi$ and calculate its δ_π. Finally, we derive the optimal π_*, which has an earliest δ_{π^*}. Fig. 10 shows the workflow of *Online Data-aware Scheduler*. When executing the arrival chunk, it persistently updates the T and the executed layers set Ω at each *Worker* for scheduling the next arrival video chunk.

D. Workflow of Cross-Edge Collaboration Inference

Since the online *DAS* has derived the optimal partitioning scheme π^* for the newly arrival video chunk with a length of K, *Worker* $0, w (1 < w < M)$ and $M-1$ execute the layers $(0, p_1^{\pi^*})$, $(p_w^{\pi^*}, p_w^{\pi^*}+1)$ and $(p_{M-1}^{\pi^*}, N)$ respectively. For each frame in this chunk, the *Master* generates an initial ***IFR*** (i.e., InFeRence) file, which indicates the layers to be executed at *Worker 0* and consists of the data of the raw or residual frame compressed by *Sparse Encoder*. Fig. 11 illustrates the workflow of cross-edge collaboration inference, where $Q_{i,j}$ indicates the encoded data for *Worker* j of frame i, and $F_{i,j+1}$ represents the output raw feature map at *Worker* j.

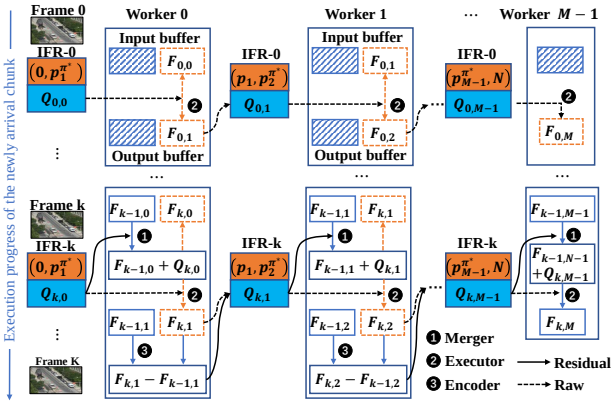


Fig. 11. Workflow of cross-edge collaboration inference.

TABLE II
LINES OF CODE IN RESMAP IMPLEMENTATION.

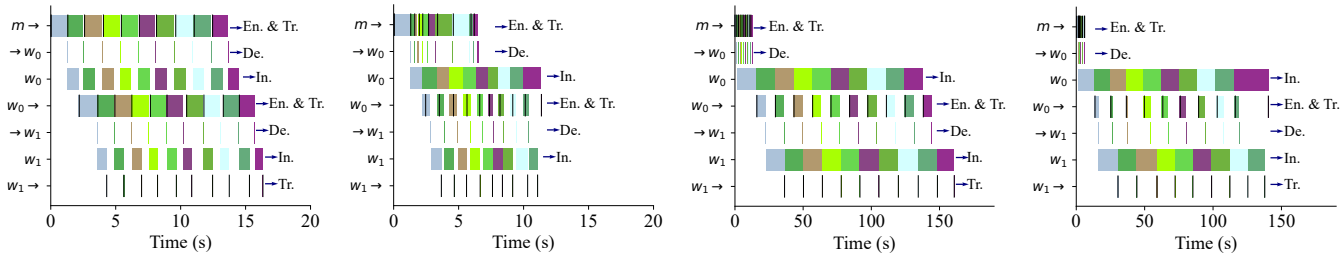
Roles	Component	Lines of Code (LoC)
ResMap Master	DAS, Predictor, Tracker	5,216 lines of Python
ResMap Worker	Executor, Merger, Encoder	2,257 lines of Python

For the header *IFR-0*, its data segment $Q_{0,0}$ is the encoded raw frame. When *Worker* 0 receives *IFR-0*, considering no input buffer (i.e., the raw feature map of last frame generated at the worker before), it directly takes $Q_{0,0}$ as the input to execute the concrete inference, and updates the input buffer with $Q_{0,0}$. Similarly, having no output buffer (i.e., the raw feature map of last frame generated at this worker), the *Sparse Encoder* directly updates $Q_{0,0}$ in *IFR-0* to $Q_{0,1}$ with the compressed $F_{0,1}$ and updates the output buffer with $F_{0,1}$. The subsequent *Workers* repeat these operations until *Worker* $M-1$ outputs the final result $F_{0,M}$.

For any *IFR-k*, $1 \leq k \leq K$, its data segment $Q_{k,0}$ consists of the compressed raw or residual frame. Specifically, if $Q_{k,0}$ is the residual data, the *Merger* in *Worker* 1 adds the buffered input $F_{k-1,0}$ to this residual data $Q_{k,0}$ to generate the complete input $F_{k-1,0} + Q_{k,0}$. Otherwise, the raw data $Q_{k,0}$ is also the input for inference. Then, the input buffer is updated with this new data, and the *Executor* runs concrete inference and produces a new feature map $F_{k,1}$. After that, the output buffer is also updated with $F_{k,1}$. Finally, the *Sparse Encoder* measures the *nonzero-rate* of this new feature map or its residual map, and selects the one with lower *nonzero-rate* to update the data segment in *IFR-k*. The successive *Workers* do the same procedures until *Worker* $M-1$ gets the analytics result $F_{k,M}$ for frame k. Note that, all the *IFRs* share the same inference plan (i.e., the layers executed at each *Worker*).

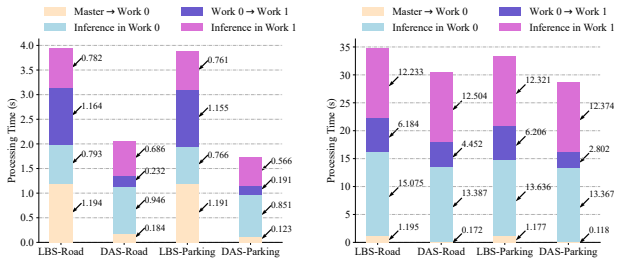
V. IMPLEMENTATION AND EVALUATION

ResMap focuses on accelerating the edge video analytics by the coherent execution of *Master* and multiple *Workers*,



(a) Pipeline of AlexNet using LBS (b) Pipeline of AlexNet using DAS (c) Pipeline of VGG16 using LBS (d) Pipeline of VGG16 using DAS

Fig. 12. The pipelines of AlexNet and VGG16 using *LBS* and *DAS* scheduler respectively.



(a) Time distribution of AlexNet (b) Time distribution of VGG16

Fig. 13. Processing times in each stage of AlexNet and VGG16.

which are deployed on edge devices. We have implemented both of them fully based on COTS hardware, and the deep learning framework adopted in ResMap is PyTorch, one of the most widely used framework. Table II show the lines of code (LoC). This section presents the effectiveness of ResMap in the inference acceleration and intermediate data reduction.

A. Experimental Setup

We use three 4B Raspberry Pi's that are provisioned with 2GB memory, one is used as the *Master*, and the other two devices are the *Worker* to execute the concrete inference. We use *TC* tool to set the maximum transmission rate between two devices to 4MB per second. For a comprehensive evaluation, we select four CNN models (i.e., AlexNet, VGG16, GoogLeNet and ResNet) with diverse complexities, which comprise 14, 32, 173, and 203 layers respectively. As for the testing data, we use two categories of videos including the road and parking videos from data set VIRAT [24]. These two types of videos have a static background and dynamic moving cars. Compared to the parking video, the road video presents more dynamics. Two key evaluation metrics are listed as follows:

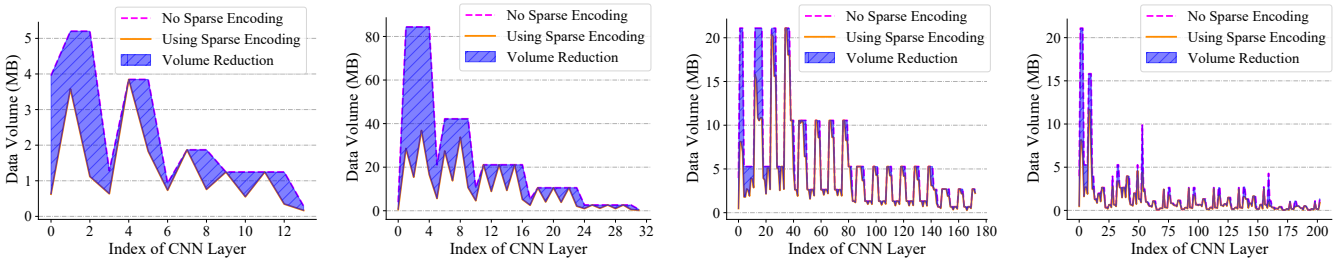
- *Feature map volume reduction* in each layer. As stated before, feature maps outputted by Convolutional, ReLU and MaxPool layers present diverse *nonzero-rates*. Thus, we explore the data reduction for each layer.
- *Average inference acceleration*. We also tune the maximum parallel pipelines and the length of video chunk to test the improvements.

As for the *baseline*, we compare the *Data-Aware Scheduler (DAS)* with the widely used *Load-Balanced Scheduler (LBS)* in many researches [12]–[14], [18], [19] , which distributes the total computations to the involved *Workers* based on the their computation resources and the inter-device bandwidth, and transmits the raw intermediate feature map to the next *Worker* for subsequent inferences. It is usual that *LBS* uses a fixed or slightly tuned scheme for the specific model.

B. Acceleration of Pipeline Execution

Since each stage (i.e., inference or transmission) consumes independent resources that do not affect each other: CNN inference utilizes GPUs or CPU resources of *Worker*, encoding and decoding use the hardware encoder and decoder respectively, and transmitting feature map consumes the bandwidth, ResMap enables the feature map transmission and CNN inference to be effectively pipelined and executed parallel. Gantt Fig. 12 illustrates the pipelines of *BLS* and *DAS* to run AlexNet and VGG16 for road video. We use the same color to record the executions of one frame from the *Master* m to the last *Worker* w_1. Both $'m \rightarrow'$ and $'w \rightarrow'$ indicate the stages of encoding and transmission, which are separated by a vertical line. $' \rightarrow w'$ represents the decoding in *Worker* w, and $'w'$ indicates the concrete CNN inference. For each frame to be analyzed, it is firstly encoded and transmitted by the *Master* m (i.e., $'m \rightarrow'$). Then, having received this frame, w_0 decodes it ($' \rightarrow w_0'$), runs CNN inference in it ($'w_0'$). When the inference is finished, it encodes and delivers the output feature map to w_1 ($'w_0 \rightarrow'$). w_1 repeats the above stages. Note that if one frame is completed in w_0, it will not be sent to w_1.

As Fig. 12(a) shows, considering the same computing powers of *Worker* w_0 and w_1, *LBS* schedules approximate computations to them, thus causing similar inference times. In addition, without using *Sparse Encoding*, *LBS* transmits the same data volume for each frame, thus leading a same transmission time. Though no *Sparse Encoding* makes the data encode and decode faster, transmission critically hinders the video analytics performance. On the contrary as Fig. 12(b) shows, the proposed *DAS* utilizes the *Sparsity* of feature map to flexibly partition the CNNs, thus significantly reducing the intermediate data volume and advancing the completion time. Though the *Sparse Encoding* brings extra overhead, it largely benefits the transmission. Due to the dynamics of video content over time, the volume of each encoded data is also different. Specifically, *DAS* directly schedules the entire CNN to w_0 for one-shot inference, which further advances the completion time. On the whole, *DAS* reduces the transmission delay by 68.41% and the average completion time by 30.6%. Similarly, we evaluate the effectiveness of ResMap to run VGG16, the results are illustrated in Fig. 12(c) and Fig. 12(d). *LBS* costs much more time to transmit the intermediate feature map compared to *DAS*. What's more,



(a) Data volume of AlexNet (b) Data volume of VGG16 (c) Data volume of ResNet (d) Data volume of GoogLeNet

Fig. 14. feature map reductions of AlexNet, VGG16, ResNet, GoogLeNet respectively using ResMap in road video.

LBS fixes the scheduling strategy to keep the approximate computation at each *Worker*, hence to improve the completion time, the transmission delay becomes the bottleneck. On the contrary, *DAS* tunes its partition scheme to make the finish time of each *Worker* as close as possible. The *maximum* finish time among all the *Workers* is the *minimum* completion time. To sum up, *DAS* enables to reduce the transmission delay by 53.58% and the average completion time by 17.43%.

We also explore the resulting delay in each stage for single frame as Fig. 13 shows. To run AlexNet using road video, *LBS* takes almost the same times (i.e., 0.793 and 0.782 seconds) to perform CNN inference in *Worker 0* and *Worker 1*, while *DAS* costs 0.26 seconds more in *Worker 0* than that in *Worker 1*. In addition, *LBS* spends a total of 2.358 seconds on transmission, however, *DAS* costs only 0.416 seconds, thus reducing the transmission time by 82.35%. As a result, it improves the total overhead of each frame by 47.93%. Compared to the road video, the parking video presents less changes, thus causing sparser feature map, hence the transmission time of *DAS* in parking video (0.416) is lower than that in road video (0.314). Fig. 13(b) shows the results when running VGG16. For VGG16, its inference time accounts for the bulk of the total time (i.e., 27.308 to 34.687 seconds). For single frame, *DAS* makes little progresses in inference acceleration, but it largely reduces the transmission time from 7.379 to 4.624 seconds (i.e., a 37.3% improvement), and further reduces the total time by 12.03%. It shows the similar laws when we feed parking video. For a complex CNN, whose inference time takes up the most of total time, the proposed *DAS* benefits slightly the completion time despite significant data reduction.

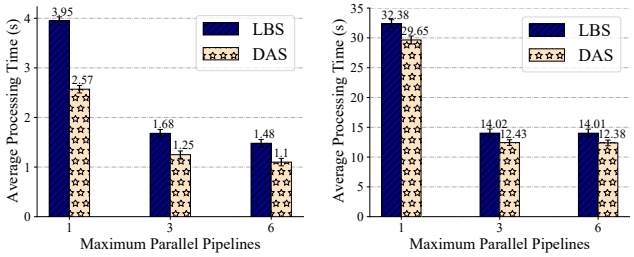
C. Feature Map Reduction

The goal of ResMap is to mitigate the heavy intermediate feature maps through *Sparse Encoding*. For each layer, ResMap employs *CSR method* to encoding the feature map if its *nonzero-rate* meets the given threshold. Fig. 14 shows the effectiveness in data reduction of ResMap. In Fig. 14(a), ResMap greatly reduces the data in layers 1, 2, 5, 8, 10 and 11, which are mainly the ReLU layers. While in convolutional layers 4, 7, 9, and 11, ResMap delivers the nearly entire feature map to the next *Worker*. As a result, it reduces the average data volume for a frame by 39.21% and the data for single layer by at most 87.85%. We then look at VGG16, its feature maps outputted by layers from 1 to 10 are clearly compressed, and in the following *ReLU* layers, ResMap is also working efficiently. In average, ResMap reduces the data by 46.12%. As for the ResNet with 173 layers, ResMap makes critical progress in layers from 0 to 30, and reduces the average data volume by 24.67%. Finally for the GoogLeNet, ResMap still obtains a 14.93% reduction. In fact, based on the reductions of each layer, ResMap enables to perform more efficient scheduling. Specifically, if the feature map of a layer is not sparse, ResMap desires to run the next layer in the same *Worker*, which does not make any cross-edge transmission. On the contrary, if ResMap detects a low sparse feature map, it probably encodes and transmits it to the next *Worker* for subsequent inferences.

D. Performance Measurements under Diverse Settings

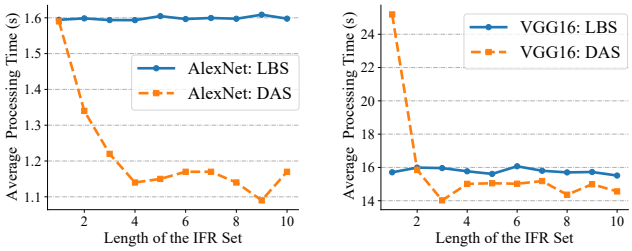
In a general pipeline system, to prevent the tasks including feature map transmission and CNN inference from piling up on one *Worker* and exhausting its constrained memory, we need to limit the number of parallel pipelines. Fig. 15 shows the achieved performances of *LBS* and *DAS* when setting the *maximum parallel pipelines (Mpp)* to 1, 2, and 3. *Mpp* equals m, which means that at most m frames are processed at the same time in all *Workers*. If setting *Mpp* to 1, one frame begins to be executed until the previous frame are finished, which can hardly utilize the advantage of *multi-edge* collaboration. However, using a large *Mpp* that exceeds the available resources in terms of computation and memory probably makes no sense to accelerate the average processing time. When running AlexNet, our proposed *DAS* costs 2.57, 1.25, and 1.1 seconds when *Mpp* equals 1, 3, and 6 respectively, which outperform the *LBS* by 1.38, 0.43, and 0.38 seconds respectively. We next measure the impacts for VGG16 in Fig. 15(b). Due to the expensive inference and transmission overhead, assigning a bigger *Mpp* does little to improve the completion time, but increase the computation and memory load of the involved *Workers*. For instance, *DAS* gets the similar average processing times (i.e., 14.02 and 14.01 seconds) when setting *Mpp* to 3 and 6 respectively. In fact, *Mpp* relies on both the complexity of CNNs and the number of *Workers*. We only employ two *Workers*, which largely limits its value range. In the future work, we focus on deploying more edge *Workers* to explore the optimal *Mpp*.

As stated before, ResMap makes model partitioning scheme at video chunk granularity and generates an executive file *IFR* for each frame in this chunk. Hence each *IFR* shares the same partition scheme, but carries different data. Increasing the length of the *IFR* set avoids continuous updates to the buffer in each *Worker*, but is probably not applicable to the videos with frequent scene-changes. Hence we flexibly choose the length



(a) Case for AlexNet (b) Case for VGG16

Fig. 15. Average processing times under different maximum parallel pipelines.



(a) Case for AlexNet (b) Case for VGG16

Fig. 16. Average processing times with different lengths of the IFR set.

of *IFR* set. Fig. 16 illustrates the resulting performances with diverse settings. In Fig. 16(a) for running AlexNet, *LBS* costs a close processing time in average no matter whatever length is given. While increasing it from 1 to 4, *DAS* largely improves the average processing time. However, it makes fewer progress if we choosing a bigger value such as 10, which is largely due to the variation of both the video content and available bandwidth. As for running VGG16 in Fig. 16(b), *DAS* is much more sensitive to the length of *IFR*. For *DAS*, until all the frames of last chunk have been finished, it begins to process the arrival video chunk. Hence, setting the length to 1 means that $Mpp = 1$, thus not taking the advantage of pipeline execution and incurring a poor processing time. It also makes no sense to choosing an excessive length for *VGG16*. As a result, we assign 4 and 3 to AlexNet and VGG16 respectively.

VI. Related Work

A. CNN Inference Acceleration.

To mitigate the insufficient computation and memory at the edges but accelerate the CNN inference, a variety of approaches have been studies, such as model compression [6]–[8], model early-exit [9]–[11], model partitioning [12]–[15], [20]–[23], data partitioning [25]–[29], and specific hardware or tools [30], [31]. Specifically, model compression is to prune the redundant layers by identifying the unimportant connections [6] or applying L1-norm channel pruning and Fisher pruning [8]. However, the compressed model still generates large intermediate data when given a huge input. Model early-exit proposes to execute partial DNNs to output the result [9]–[11]. For instance, BranchyNet [10] adds several exit branches to the origin CNNs. This method has to retrain the model, which is time-consuming. Model partitioning is to split the CNNs and distribute the slices to multiple devices. For instance, Jeong *et al.* [22] run these two operations in parallel. However, the edge-cloud collaborative method may incur lager amounts of intermediate data transmission. Data partitioning schedules the partitions to devices for parallel executions. MoDNN [25] presents two methods for fully-connected and convolutional layers. However, this method causes significant synchronization overhead. The Intel OpenVino [30] and Google TPU [31] do accelerate the inference, but not all devices are provisioned with these tools.

B. Collaborative Edge Video Analytics

Many cities and enterprises are deploying thousands of edge cameras and using video analytics to serve a variety of 24x7 applications [33], [34] such as traffic control and security monitoring. As the progress of Edge Intelligence (EI) or Edge AI [32], there emerges increasing efforts [20]–[23], [35]–[39] to accelerate the video analytics through collaborative edge devices and the cloud. DeepDecision [36] also ties together front-end devices and powerful cloud to complete the video analytics tasks. AutoML [37] proposes to configure the wireless network parameters to advance the video analytics at edge server. Han *et al.* [39] achieve efficient video analytics by adjusting the number of frames to be offloaded to the edge server. Most existing works consider an end-edge-cloud collaborative method. However, it incurs significant delay to transmit the large intermediate data. Our proposed ResMap delivers the cross-edge data using reliable inter-device connections, thus largely reducing the transmission overhead.

Despite several works that considered the multi-edge collaboration [12]–[19], they fall insufficient. Studies [12]–[14] make effective model partition based on the runtime information, but they do not optimize the feature map transmission. Works [15]–[17] use adaptive compression or pruning technique to reduce the data volume, however resulting in a decrease in analytics accuracy. The rest [18], [19] propose to iteratively derive the optimal scheme from the large solution space, however incurring a huge profiling delay. ResMap optimizes both the model partitioning and data transmission.

VII. Conclusion

In this paper, we present ResMap, a new video analytics framework that significantly accelerates the CNN inference by employing multi-edge collaborative execution. Rather than fixing the partition scheme, ResMap incorporates an online *Data-Aware Scheduler*, which designs a *DP (Dynamic Programming)*-based algorithm to flexibly partition the specific CNNs based on both the transmission data estimation of the arrival video chunk and the runtime information. In addition, ResMap proposes *Sparse Encoder* to reduce the large volume of intermediate feature map. For each frame to be analyzed, ResMap generates a customized *IFR* file to conduct its execution at each device. Coherently, ResMap employs *Master-Worker* mechanism to run each cross-edge inference task. We have implemented ResMap fully based on COTS hardware, and the experimental results show that ResMap reduces the intermediate feature map volume by 14.93-46.12% and improves the average processing time by 17.43-30.6%. In the future work, we aim to apply ResMap into more types of videos as well as complex CNNs to further test its generality.

REFERENCES

[1] Microsoft HoloLens. https://www.microsoft.com/en-us/hololens/.

[2] Magic Leap One. https://www.magicleap.com/.

[3] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on endedge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," in ACM Computing Surveys, vol. 52, no. 6, pp. 1–36, 2019.

[4] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," in Proceedings of the IEEE, vol. 107, no. 8, pp. 1655-1674, 2019.

[5] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar and A. Y. Zomaya, "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence," in IEEE Internet of Things Journal, vol. 7, no. 8, pp. 7457-7469, 2020.

[6] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in Proceedings of *NIPS*, pp. 1135–1143, 2015.

[7] E. J. Crowley, J. Turner, A. Storkey, andM. O'Boyle, "A closer look at structured pruning for neural network compression," in arXiv:1810.04622, 2019.

[8] X. Zhang, X. Zhou, M. Lin and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6848-6856, 2018.

[9] L. Li, K. Ota and M. Dong, "Deep Learning for Smart Industry: Efficient Manufacture Inspection System With Fog Computing," in IEEE Transactions on Industrial Informatics, vol. 14, no. 10, pp. 4665-4673, 2018.

[10] S. Teerapittayanon, B. McDanel and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in 2016 23rd International Conference on Pattern Recognition (ICPR), pp. 2464-2469, 2016.

[11] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?," in Cognitive Computation, vol. 12, no. 5, pp. 954–966, 2020.

[12] J. H. Ko, T. Na, M. F. Amir and S. Mukhopadhyay, "Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms," in 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1-6, 2018.

[13] M. Xu, F. Qian, and S. Pushp, "Enabling cooperative inference of deep learning on wearables and smartphones," in arXiv: 1712.03073, 2017.

[14] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT," in IEEE Internet of Things J., vol. 7, no. 10, pp. 9241–9254, 2020.

[15] D. Hu and B. Krishnamachari, "Fast and Accurate Streaming CNN Inference via Communication Compression on the Edge," in 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 157-163, 2020.

[16] J. Mao et al., "MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs," in 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 751-756, 2017.

[17] L. Zeng, X. Chen, Z. Zhou, L. Yang and J. Zhang, "CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices," in IEEE/ACM Transactions on Networking, vol. 29, no. 2, pp. 595-608, 2021.

[18] X. Tang, X. Chen, L. Zeng, S. Yu and L. Chen, "Joint Multiuser DNN Partitioning and Computational Resource Allocation for Collaborative Edge Intelligence," in IEEE Internet of Things Journal, vol. 8, no. 12, pp. 9511-9522, 2021.

[19] K.J. Hsu, K. Bhardwaj, A. Gavrilovska. "Couper: Dnn model slicing for visual analytics containers at the edge," in Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 179-194, 2019.

[20] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in ACM SIGARCH Computer Architecture News, vol. 45, no. 1, pp. 615–629, 2017.

[21] C. Hu, W. Bao, D. Wang and F. Liu, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge," in IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1423-1431, 2019.

[22] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "IONN: Incremental offloading of neural network computations from mobile devices to edge servers," in Proceedings of the ACM Symposium on Cloud Computing, pp. 401–411, 2018.

[23] S. Dey, J. Mondal and A. Mukherjee, "Offloaded Execution of Deep Learning Inference at Edge: Challenges and Insights," in 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 855-861, 2019.

[24] The VIRAT Video Dataset[EB/OL]. https://viratdata.org/.

[25] J. Mao, X. Chen, K. W. Nixon, C. Krieger and Y. Chen, "MoDNN: Local distributed mobile computing system for Deep Neural Network," in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1396-1401, 2017.

[26] Z. Zhao, K. M. Barijough and A. Gerstlauer, "DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2348-2359, 2018.

[27] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 195–208, 2019.

[28] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Collaborative execution of deep neural networks on Internet of Things devices," in arXiv: 1901.02537, 2019.

[29] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in Proceedings of Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 77–90, 2019.

[30] Intel distribution of OpenVINO toolkit. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/ openvino-toolkit.html

[31] Cloud TPU. [Online]. Available: https://cloud.google.com/tpu

[32] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen and M. Chen, "In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning," in IEEE Network, vol. 33, no. 5, pp. 156-165, 2019.

[33] Artificial Intelligence Surveillance Cameras Security. https://www.theverge.com/2018/1/23/16907238/artificial-intelligence-surveillance-cameras-security.

[34] New Search Engine Revolutionizes Video Surveillance. https://i-hls.com/archives/80734.

[35] L. Liu, H. Li, M. Gruteser. "Edge assisted real-time object detection for mobile augmented reality," in 25th annual international conference on mobile computing and networking, pp. 1-16, 2019.

[36] X. Ran, H. Chen, X. Zhu, Z. Liu and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, pp. 1421-1429, 2018.

[37] A. Galanopoulos, J. A. Ayala-Romero, D. J. Leith and G. Iosifidis, "AutoML for Video Analytics with Edge Computing," in IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pp. 1-10, 2021.

[38] T. Tan and G. Cao, "FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile," in IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pp. 1947-1956, 2020.

[39] M. Hanyao, Y. Jin, Z. Qian, S. Zhang and S. Lu, "Edge-assisted Online On-device Object Detection for Real-time Video Analytics," in IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pp. 1-10, 2021.