Tensor Compression and Reconstruction in Split DNN for Real-time Object Detection at the Edge

Tensor Compression and Reconstruction in Split DNN for Real-time Object Detection at the Edge

Yenchia Yu
Politecnico di Torino
Turin, Italy

Marco Levorato
University of California, Irvine
California, USA

Carla Fabiana Chiasserini
Politecnico di Torino, Italy
Chalmers University of Technology, Sweden

***Abstract*—Computer vision applications for UAVs often rely on deep neural networks (DNNs) to increase prediction accuracy. As such DNNs crave for computational resources that can be hardly matched by those available at the UAVs, an emerging solution is to split the DNN into a low-complexity head model run at the UAV and a heavier tail run at the edge. This approach, however, comes at the cost of transmitting a large tensor data, hence of high bandwidth consumption, on the UAV-edge radio link. We tackle this problem by proposing the Compressed Tensor-based DNN split (CoTeD) framework, which executes tensor compression at the UAV and reconstruction at the edge, while conveniently trading off tensor compression with quality of the computer vision task output. When compared with the no-split case, CoTeD reduces the UAV computational burden by 50% w.r.t. performing inference at the UAV only, and the amount of transmitted data by over one order of magnitude w.r.t. running inference at the edge only. When compared to compressive sensing, JPEG-100, and the whole DNN run at the edge, CoTeD decreases the overall latency by (resp.) 95%, 75%, and 80%.**

***Index Terms*—Edge computing, UAVs, Bandwidth utilization**

I. Introduction

Computer vision applications play a fundamental role in unmanned aerial vehicles (UAVs), as they enable object detection and tracking in real-time, thus providing support for autonomous UAV navigation and monitoring of the UAV surroundings. Object detection, in particular, emerges as one of the fundamental, but also most challenging, computing vision tasks [1]. Indeed, the following issues may heavily impact the execution and the output of object detection: (1) UAV image quality varies severely depending on the taken scene, the weather conditions, and the sensor capability; (2) the size and scale of the objects captured in the images may also significantly vary; (3) the deep neural networks (DNNs) that are required to ensure a high-quality output in spite of the issues above are often complex; (4) ensuring timely inference through such DNNs is difficult and costly.

An approach that effectively addresses the last two issues, while preserving data privacy, is DNN split, which implies partitioning the DNN into segments, deploying them on different computing nodes, and connecting these segments according to the original processing pipeline with an appropriate communication protocol [2], [3], [4]. However, the implementation of DNN splitting requires careful consideration of such factors as the partitioning strategy, the radio bandwidth consumption, and the processing latency at the different computing nodes.

This work has been supported by Leonardo and by the EU under the Italian NRRP of NextGenerationEU RESTART program (PE00000001).

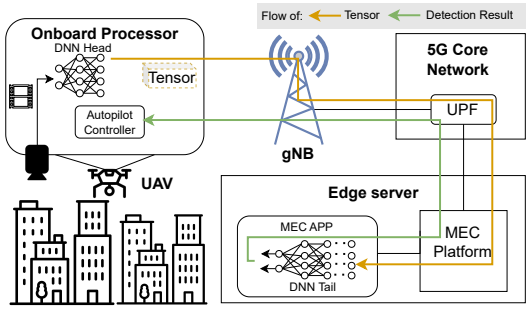

Fig. 1: Application and system scenario: UAV-edge cooperative video object detection for UAV flight control.

In this work, we face the above challenges focusing on the network scenario depicted in Fig. 1, where the DNN is split into a small, low-complexity head segment executed at the UAV and a larger, more complex tail segment running at an edge server. Then, to overcome the issue of transferring the large data tensor produced by the head model to the tail model running at the edge, we propose CoTeD, a low-complexity framework for efficient tensor compression and transmission. The benefit of CoTeD can be summarized as follows:

- It can be easily adapted so to different pre-trained DNNs without any model modification;
- It can automatically and dynamically adjust the tensor compression ratio to the available radio bandwidth, thus trading off radio resource consumption with application quality;
- When compared to alternative tensor compression techniques like compressive sensing and JPEG, or edge-only DNN deployment, it exhibits high efficiency and decreases the overall latency by over 95%, 75%, and 80% (resp.).

In the rest of the paper, we discuss some relevant related work and highlight the novelty of our study in Sec. II. We then detail our reference scenario in Sec. III and introduce CoTeD in Sec. IV. Further, we show some experimental results on CoTeD operational efficiency and performance in Sec. V. Finally, we draw our conclusions in Sec. VI.

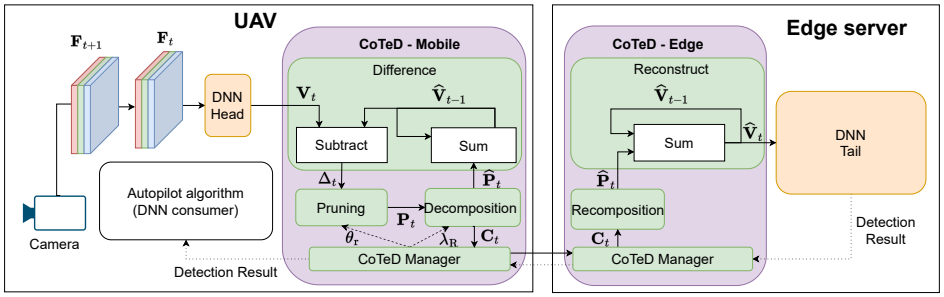

Fig. 2: Scheme of the CoTeD framework: DNN head and tensor difference, pruning, and decomposition at the UAV; tensor recomposition and reconstruction and DNN tail at the edge server.

II. Related Work

Edge-assisted split DNN inference is a pivotal technology that enables the deployment of complex DNNs on computation-limited devices like UAVs. A survey of the main split DNN applications is presented in [5], which also highlights as major challenges the use of early exit branches in DNN models and the need to reduce bandwidth consumption due to tensor transfer between DNN head and tail. For tensor compression, a popular solution is to introduce "bottleneck" layers at the end of the DNN head [6], [7], which is considered to be highly effective but it requires DNN model modification and retraining. Another approach leverages picture/video encoding-based algorithms, e.g., JPEG [8], to encode the DNN head output tensor directly, which however requires a careful balance between encoding quality and bandwidth consumption. Other solutions like [9] propose to quantize and compress the features that are output by the split DNN head, which requires analyzing the value distribution of each activation tensor and, thus, may lead to high processing delay.

Novelty. Unlike prior art, CoTeD provides an effective tensor transmission solution for split DNN by dynamically adjusting the compression ratio and without requiring any modification to pre-trained models. In doing so, CoTeD makes the deployment of split DNN efficient and sustainable, from both the communication and computation point of view.

III. Reference Application and System Scenario

We consider the scenario depicted in Fig. 1, including an edge server, a 5G base station (gNB) and Core Network (CN), and a UAV acting as 5G User Equipment (UE). The edge platform uses the 5G-CN's User Plane Function (UPF) as its virtualized data plane, thereby ensuring minimal latency and communication distance between the edge applications and the 5G UE. The UAV is operated by an onboard autopilot application, which includes a DNN for video object detection on the frames captured by the UAV's camera.

Due to the computational limitations of the UAV's onboard processor, a split DNN architecture is deployed [10], composed of head and tail segments. Having limited complexity, the head model is executed at the UAV, while the higher-complexity tail model is run at the edge. The rationale behind the choice of splitting the DNN is that, on one hand, processing video frames at the UAV contributes to privacy preservation; on the other hand, delegating the heavier application processing to the edge both suits the resource limitations of the UAV and ensures fast processing times. It is indeed worth noting that object detection needs to be output within a strict deadline (namely, 10 s–100 ms) imposed by the autopilot application.

DNN splitting, however, necessitates the transfer through the 5G radio interface of the tensor generated by the head model, which is natively large in size, easily reaching tens of Mbits. Thus, **the tensor transmission requires large bandwidth and, given the application deadline, a 5G-connection at least one order of magnitude faster than the one available today**. To address this issue, we introduce CoTeD (Compressed Tensor-based DNN split), which operates as described below.

IV. CoTeD: Compressed Tensor-based DNN Split

We consider a video object detection process that is collaboratively executed by a UAV and an edge server. As illustrated in Fig. 2, an onboard camera captures a sequence of video frames, denoted by $[\mathbf{F}_t, \mathbf{F}_{t+1}, \ldots]$, which serve as input to the DNN head model hosted on the UAV. Upon processing frame $\mathbf{F}_t$, the head model generates an activation tensor, $\mathbf{V}_t \in \mathbb{R}^{I \times J \times K}$, that must be transmitted to the edge server and processed by the DNN tail model.

To reduce bandwidth consumption in the tensor transfer from the UAV to the edge, we introduce the Compressed Tensor-based DNN split (CoTeD) framework. CoTeD is composed of two main modules: CoTeD-Mobile and CoTeD-Edge, deployed on the UAV and the edge server (resp.). Each module encompasses a framework manager responsible for configuring the framework settings, facilitating inter-module communication, and interacting with the split-DNN and the DNN consumers. Prior to transmission from the UAV to the edge server, $\mathbf{V}_t$ is processed by a sequence of low-complexity operators in CoTeD-Mobile: (i) Difference, yielding $\mathbf{\Delta}_t$, (ii) Pruning, generating $\mathbf{P}_t$, and (iii) Decomposition, producing $\mathbf{C}_t$. The outcome of the operators' sequence, $\mathbf{C}_t$, is a list of tensor decomposition factors, i.e., a highly compressed representation of $\mathbf{V}_t$ that can be efficiently transmitted to CoTeD-Edge.

Upon reception, CoTeD-Edge processes $\mathbf{C}_t$ using the following operators: (i) Recomposition, yielding $\widehat{\mathbf{P}}_t$, and (2) Reconstruction, $\widehat{\mathbf{V}}_t$. The output of the Reconstruction operator is an estimated activation tensor, which is then input

to the DNN tail model running at the edge server to produce the object detection outcome for frame $\mathbf{F}_t$. The result is relayed back to the UAV via the CoTeD framework, and ultimately consumed by the UAV's autopilot application.

A. Tensor difference and reconstruction

We first detail the tensor Difference and Reconstruction operators, which are pivotal to the activation tensor compression process. Upon receiving the activation tensor, $\mathbf{V}_t$, from the DNN head, the Difference operator in CoTeD-Mobile (1) subtracts $\widehat{\mathbf{V}}_{t-1}$ from $\mathbf{V}_t$, generating the diff-tensor $\mathbf{\Delta}_t$, and (2) sums the pruned tensor $\mathbf{P}_t$ with $\widehat{\mathbf{V}}_{t-1}$, and stores the summation result, $\widehat{\mathbf{V}}_t$, in the internal memory for the subtract operation in the next time instance. Specifically, $\widehat{\mathbf{V}}_{t-1}$ represents the reconstructed pruned activation tensor in the previous time instance (set to zeros at the start time).

Notice that our proposed architecture makes CoTeD able to perform the differential operation on the activation tensors without the need to cache the original video frame streams. Also, leveraging $\widehat{\mathbf{V}}_{t-1}$ as reference for the Difference operation reduces the pruning and decomposition loss effect on the overall detection quality, thus increasing CoTeD's level of robustness. Indeed, whenever some values in the activation tensor $\mathbf{V}_t$ have a small variation rate, they can significantly affect the final detection result, using the previous activation tensor ($\mathbf{V}_{t-1}$) as a reference would make such values become small in the diff-tensor and eventually be pruned by the pruning operator. Instead, in CoTeD, the small value variations in the activation tensor accumulate over time and, hence, they will eventually be transferred to the DNN tail model.

In CoTeD-Edge, the Reconstruction operator sums the reconstructed pruned tensor $\widehat{\mathbf{P}}_t$ with $\widehat{\mathbf{V}}_{t-1}$ and generates the reconstructed activation tensor $\widehat{\mathbf{V}}_t$. Specifically, $\widehat{\mathbf{V}}_{t-1}$ represents the reconstructed activation tensor in CoTeD-Edge in the previous time instance, which is initialized to a zero tensor. $\widehat{\mathbf{V}}_t$ is then processed by the DNN tail model at the edge server.

B. Pruning

According to our experimental analysis on real-world DNNs (detailed in Sec. V), we noticed that when the consecutive frames input to the DNN head model have high similarity to each other, the corresponding output activation tensors will have high similarity as well. As a result, the diff-tensor $\mathbf{\Delta}_t$ is mainly composed of small normalized values, which are considered to have negligible impact on the further processing in the DNN tail model, but introduce significant difficulties in the tensor compression and transmission. To solve this issue, CoTeD processes $\mathbf{\Delta}_t$ with a pruning operator [11], [12], which generates the pruned tensor $\mathbf{P}_t \in \mathbb{R}^{I \times J \times K}$ by setting to zero the elements of $\mathbf{\Delta}_t$ with normalized value smaller than a threshold μ. To do so, Pruning (i) creates a binary mask $\boldsymbol{\mathcal{M}}$ of values larger than μ, and (ii) calculates the element-wise product between mask $\boldsymbol{\mathcal{M}}$ and diff-tensor $\mathbf{\Delta}_t$.

It is worth noting that Pruning not only increases the sparsity of tensor $\mathbf{P}_t$, but, more importantly, it reduces the rank of tensor $\mathbf{P}_t$, which has a critical impact on the following Decomposition procedure (Sec. IV-C) and is required to be adjustable. In particular, let us consider $\mathbf{P}_t$ to be composed of I slices, each denoted by $\mathbf{P}_t^i \in \mathbb{R}^{J \times K}$. We observe that the relationship between the pruning threshold μ and averaged tensor slice rank $Avg[rank(\mathbf{P}_t^i)]$ is nonlinear and strongly depends on the DNN head as well as the input video frames. To produce a $\mathbf{P}_t$ with a specific slice rank, Pruning operator implements a dynamic pruning algorithm as reported in Algorithm 1. The algorithm takes as input a target averaged tensor slice rank $\theta_r \in [0.4 \cdot K, 0.9 \cdot K]$ (the range is set so that over-pruning or under-pruning are avoided), and it uses a binary search method to find the proper pruning threshold μ. In so doing, the operator generates $\mathbf{P}_t$ with an average rank (computed over all $\mathbf{P}_t$'s slices) that fits the target. When the absolute error, η, of $\mathbf{P}_t$'s averaged slice rank or the search resolution drops below thresholds ϵ_η and ϵ_r (resp.), the search algorithm stops and returns the pruned tensor $\mathbf{P}_t$.

Algorithm 1 Dynamic pruning

Require: θ_r, $\mathbf{\Delta}_t$ **return** $\mathbf{P}_t$

$A \leftarrow 1$; $B \leftarrow 0$; $\eta \leftarrow 1$; $\mu \leftarrow (A-B)/2$
$K \leftarrow FullRank(\mathbf{P}_t)$
while $\eta > \epsilon_\eta$ && $(A-B) > \epsilon_r$ **do**
  Get pruned tensor $\mathbf{P}_t$
  $\rho \leftarrow Avg[rank(\mathbf{P}_t^i)]$
  $\eta \leftarrow |\theta_r - \rho| / K$
  if $\rho \leq \theta_r$ **then**
    $A \leftarrow (A-B)/2$
  else
    $B \leftarrow (A-B)/2$
  end if
  $\mu \leftarrow (A-B)/2$
end while

C. Tensor decomposition and reconstruction

The last operator in CoTeD-Mobile is the Decomposition, which leverages CANDECOMP/PARAFAC (CP) decomposition [13] to realize efficient tensor compression. The main goal of CP decomposition is to factorize a higher-order tensor as a sum of component rank-one tensors, simplifying the complex, multi-dimensional data structure into more manageable ones. For example, a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ can be expressed as sum of the outer product of three rank-one tensors, $\mathbf{a}_r \in \mathbb{R}^I, \mathbf{b}_r \in \mathbb{R}^J, \mathbf{c}_r \in \mathbb{R}^K$, i.e.,

$$\widehat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{1}$$

where R is the decomposition rank and symbol "$\circ$" denotes the vector outer product, according to which each element of $\boldsymbol{\mathcal{X}}$ is given by the product of the corresponding elements of the three rank-one tensors, i.e., $x_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$. Importantly, if R is equal to the rank of tensor $\widehat{\boldsymbol{\mathcal{X}}}, then\ \widehat{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}}$. The rank-one tensors can be obtained by solving the following problem:

$$\min_{\boldsymbol{\mathcal{X}}} \left\| \boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}} \right\|_F, \tag{2}$$

where $\|\boldsymbol{\mathcal{X}}\|_F = \sqrt{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}|\mathcal{X}_{ijk}|^2}$. The above optimization can be conveniently solved using the alternating least squares algorithm. In this work, we adopted TensorLy library (https://tensorly.org/stable/index.html) to implement the decomposition procedure, which can be accelerated using [14].

As mentioned above, upon applying CP decomposition on a tensor with R equal to the tensor rank, the tensor can be reconstructed exactly. However, computing the rank of a 3D tensor is NP-complete and NP-hard for higher-dimension tensors. To reduce the computation load introduced by the CP decomposition, the CoTeD-Mobile decomposition operator works with tensor slices, i.e., 2D tensors, whose rank can be easily obtained through low-complexity matrix operations. Specifically, when a pruned tensor $\mathbf{P}_t \in \mathbb{R}^{I\times J\times K}$ is input to Decomposition, it is processed slice by slice (a slice being denoted with $\mathbf{P}_t^i \in \mathbb{R}^{J\times K}$).

Notably, even though $\mathbf{P}_t$ is a pruned tensor with high sparsity, most tensor slices still have very high rank close to K; thus, the decomposition rank needs to be properly set for each slice separately. Further, to balance between compression ratio and tensor reconstruction quality, we introduce a scaling factor $\lambda_{\text{R}} \in [1/K, 1]$ to manipulate the decomposition rank for the I slices of the pruned tensor $\mathbf{R}=[R^1, R^2, ..., R^I]$, with R^i being the decomposition rank for slice $\mathbf{P}_t^i$, i.e.,

$$R^i = \begin{cases} 0, & \text{if } rank(P_t^i)=0 \\ \max(1, & \\ round[\lambda_R \cdot rank(P_t^i)]), & \text{else.} \end{cases} \tag{3}$$

After the CP decomposition, we get a list of I decomposition factors, denoted by $\mathbf{C}_t$, with each factor given by two pairs: (J, R^i) and (K, R^i). The compression ratio associated with tensor $\mathbf{P}_t$ can then be computed as,

$$\begin{aligned} r_c &= \frac{I \cdot J \cdot K}{\sum_{i=1}^{I}(J+K)\cdot R^i} \\ &\approx \frac{J \cdot K}{(J+K)\cdot \lambda_R \cdot Avg[rank(P_t^i)]}. \end{aligned} \tag{4}$$

Notice that, since $\mathbf{P}_t$ is generated by the Pruning operator, which prunes Δ_t to reach a target averaged slice rank θ_r using Alg. 1, we have $Avg[rank(P_t^i)] \approx \theta_r$. After obtaining $\mathbf{C}_t$, CoTeD Decomposition continue to perform two additional operations: *(i)* generates $\widehat{\mathbf{P}}_t$ from $\mathbf{C}_t$, which will be consumed by the Difference operator as described in Sec. IV-A, and *(ii)* transmits $\mathbf{C}_t$ to CoTeD-Edge via CoTeD Manager.

On the CoTeD-Edge, upon receiving $\mathbf{C}_t$, the Recomposition operator reconstructs the tensor slice by slice using (1) and generates the reconstructed pruned tensor $\widehat{\mathbf{P}}_t$. We remark that, since we leveraged the scaling factor λ_{R} to best tune the decomposition rank, the compression process is no longer lossless, i.e., $\widehat{\mathbf{P}}_t \neq \mathbf{P}_t$. However, our experimental results (Sec. V), show that $\widehat{\mathbf{P}}_t$ preserves the pattern of $\mathbf{P}_t$'s values, thus having a negligible impact on the object detection quality.

D. CoTeD manager

The CoTeD manager plays a pivotal role in the orchestration of the overall framework. When CoTeD is initiated, the basic configurations, including the size, σ (in bits), of the activation tensor $\mathbf{V}_t$ to be transferred, and the maximum bandwidth L_{max} over the radio link (in Mbps), are stored by the CoTeD manager for further use. Also, in CoTeD-Mobile, the manager monitors the real-time bandwidth availability, denoted as L_t, and adjusts the pruning and decomposition settings in real time to fulfill the DNN application's requirement.

Algorithm 2 CoTeD real-time framework setting update

Require: L_t, T_{tr}, σ, **return** θ_r, λ_R
1: $\lambda_R \leftarrow L_t / L_{\text{max}}$
2: **while** True **do**
3: $\quad\theta_r \leftarrow \frac{T_{\text{tr}}\cdot L_t}{\sigma \cdot \lambda_R} \cdot \frac{J\cdot K}{J+K}$
4: $\quad$**if** $\theta_r > 0.9 \cdot K$ **then**
5: $\quad\quad\lambda_R \leftarrow \lambda_R + 1/K$
6: $\quad$**else if** $\theta_r < 0.4 \cdot K$ **then**
7: $\quad\quad\lambda_R \leftarrow \lambda_R - 1/K$
8: $\quad$**else**
9: $\quad\quad$Break
10: $\quad$**end if**
11: $\quad$**if** $\lambda_R < 1/K$ **then**
12: $\quad\quad\lambda_R \leftarrow 1/K$; $\theta_r \leftarrow 0.9 \cdot K$; Break
13: $\quad$**else if** $\lambda_R > 1$ **then**
14: $\quad\quad\lambda_R \leftarrow 1$ $\theta_r \leftarrow 0.4 \cdot K$; Break
15: $\quad$**end if**
16: **end while**

Considering that the DNN application requires that a video frame is processed through the sequence of CoTeD operators (at UAV and edge) within a deadline T_D (measured in seconds), the activation tensor transmission time on the radio link should not exceed T_{tr}, given by: $T_{\text{tr}} = T_D - T_{\text{Head}} - T_{\text{Tail}} - T_{\text{CoTeD}}$, where T_{Head}, T_{Tail}, and T_{CoTeD} represent (resp.) the processing time in the DNN head and tail models and through the whole sequence of CoTeD operators – quite relatively stable values that can be measured in advance. To meet the transmission deadline T_{tr}, CoTeD-Manager has to properly configure the framework in order to reach a specific compression ratio on the activation tensor, which depends on the original size of the tensor, σ, and the real-time available bandwidth L_t. Thus, the CoTeD-Mobile manager computes the target tensor compression ratio as $r_c' = \frac{\sigma}{T_{\text{tr}}\cdot L_t}$. By using this expression in the approximated one provided above for r_c and considering that $Avg[rank(P_t^i)] \approx \theta_r$, we get:

$$\theta_r \cdot \lambda_R \approx \frac{T_{\text{tr}} \cdot L_t}{\sigma} \cdot \frac{J \cdot K}{J+K}, \tag{5}$$

where we recall that θ_r and λ_R are, respectively, the operation settings in Pruning and Decomposition, which are controlled by CoTeD-Mobile manager based on Alg. 2. To set λ_R, the algorithm first initiates λ_R to the relative network quality, i.e., L_t/L_{max}, and then iteratively adjusts its value within range $[1/K, 1]$ to find a value for θ_r (calculated using (5)) that is within in its functional range $[0.4{\cdot}K, 0.9{\cdot}K]$. If no values that meet their respective range can be found for

both parameters, the algorithm sets them to to their extreme values (as detailed in Alg. 2), resulting in a best-effort tensor compression.

V. Experimental Results

Testbed setup. We use a server with AMD EPYC 7601 CPU and NVIDIA Quadro GV100 GPU acting as the edge server and a mini-PC featuring AMD Ryzen 7 5700G CPU (no GPU) as the companion computer on the UAV. The server is connected with the mini-PC via a 100-Mbps Ethernet interface integrated with `tc`, a tool used to configure the Linux Kernel traffic scheduler, which can simulate the radio channel.

We selected a popular pre-trained real-time video object detection DNNs - YOLOv3-tiny [15] to test our CoTeD framework. It is designed as a smaller and more computationally efficient version of YOLOv3, allowing it to be deployed on resource-critical devices like UAVs. The pre-trained model is composed of 24 layers with 8.9M parameters in `float32` format. We then split the model at the 8-th layer, which outputs the activation tensor $\mathbf{V}_t$ of size $[128\times26\times26]$.

Since there is no ready-to-use dataset providing object bounding box labels for a sufficient length of continuous video frames, we create the test dataset by labeling the objects in the video frames using the extra-large YOLOv8 model from Ultralytics(https://docs.ultralytics.com/models/yolov8/), which is considered to have the cutting-edge accuracy performance. For simplicity, we selected a video with a single object in the frame (a car driving through mountains) and a duration of about 30 s, producing about 30 fps.

Framework operational analysis. We now present the analysis on the output tensor of YOLOv3's head model and the effect of CoTeD operations. Since the foundation of the CoTeD framework is the similarity between continuous video frames, we first characterize the similarity between consecutive video frames/activation tensors ($\mathbf{V}_t$) using the cosine similarity metric $\frac{\mathbf{V}_t \cdot \mathbf{V}_{t-1}}{\|\mathbf{V}_t\|\|\mathbf{V}_{t-1}\|}$ (Fig. 3(left)). One can observe that, when consecutive video frames $\mathbf{F}_t$ have a high similarity, the corresponding activation tensors ($\mathbf{V}_t$) generated by the head model exhibit a high similarity as well. In this case, the CoTeD-Difference operator effectively removes the common values in the tensors and retains only the changes in Δ_t. Then minor changes that still appear in the diff-tensor will be removed by pruning.

To assess the pruning effect on the diff-tensor and on the overall object detection quality, we perform an experiment by pruning the diff-tensor to generate $\mathbf{P}_t$, reassembling the pruned version activation tensor $\widetilde{\mathbf{V}}_t{=}\widetilde{\mathbf{V}}_{t-1}{+}\mathbf{P}_t$, and feeding $\widetilde{\mathbf{V}}_t$ to the DNN tail to get the object detection output. Fig. 3(center) shows the obtained sparsity and the averaged tensor rank $Avg[rank(P_t^i)]$ of $\mathbf{P}_t$, and the corresponding object detection sensitivity and accuracy as functions of the pruning threshold μ. Note that, for $\mu{=}0$, the pruning operator is deactivated, thus the corresponding object detection sensitivity and accuracy reflect the original performance of the YOLOv3-tiny model, and the diff-tensor Δ_t has full rank (i.e., 26). As μ increases, the tensor sparsity significantly grows, till $\mu{=}0.18$ (the upper limit in this test). At that point, the detection sensitivity of the split DNN significantly decreases, i.e., CoTeD is operating in over-pruning. Importantly, the detection accuracy of the split DNN remains constant for any value of μ, which shows that the Pruning operation in CoTeD only affects the detection sensitivity of the DNN.

After the Pruning operation, the pruned tensor is compressed for transmission by the Decomposition operator (which is lossy) and decompressed by the Recomposition operator. To investigate the effect of decomposition, we first fix the pruning threshold to 0.1, resulting in the pruned tensor having 80% sparsity and an averaged tensor rank of 22. After Decomposition and Recomposition, we input the reconstructed activation tensor $\widehat{\mathbf{V}}_t{=}\widehat{\mathbf{V}}_{t-1}{+}\widehat{\mathbf{P}}_t$ to the DNN tail. Fig. 3(right) depicts the Mean Square Error (MSE) of $\widehat{\mathbf{P}}_t$ and the compression ratio r_c of Decomposition operator, along with the object detection sensitivity and accuracy when $\widehat{\mathbf{V}}_t$ is input to the DNN tail, as functions of the decomposition rank scaling factor λ_R. As λ_R increases, the MSE of $\widehat{\mathbf{P}}_t$ w.r.t. λ_R decreases, but clearly the tensor compression ratio r_c decreases as well. Importantly, the effect of the reconstruction errors on the object detection sensitivity and accuracy of the DNN is negligible, demonstrating that CoTeD can highly compress the activation tensor for transmission over the radio channel with minimum impact on the overall detection quality.

Performance tests. Next, we look at the performance of CoTeD by first investigating the behavior of CoTeD Manager. We deploy YOLOv3-tiny model (split at the 8-th layer) on our testbed and set the transmission deadline of the activation tensor to $T_D{=}30$ ms. We also set the maximum bandwidth to $L_{\text{max}}{=}50$ Mbps and we let the available bandwidth L_t vary between 0.1 Mpbs and 50 Mbps.

Fig. 4(left) shows the target and actual values of the compression ratio, r_c' and r_c, and the corresponding tensor transmission time T_{tr} as functions of the real-time available bandwidth L_t. For $L_t{\leq}8$ Mbps, r_c' is larger than the native compression limit of the tensor (i.e., the limit of θ_r and λ_R in Sec. IV-D). Thus, CoTeD manager configures the corresponding operators so as to apply a best-effort compression and the system may be unable to honor the deadline T_D. On the contrary, for $L_t{>}8$ Mbps, the transmission deadline is always met; hence, CoTeD Manager reduces the target compression ratio so that, on the edge side, a higher quality of the reconstructed activation tensor $\widehat{\mathbf{V}}_t$ can be achieved.

Next, we measure the per-frame execution time (not including the communication delay) of the YOLOv3-tiny model when it is deployed: (1) completely on the UAV (on the mini PC in our testbed), (2) split at the 8-th layer, with the head model running at the UAV and the tail model at the edge server, and (3) completely on the edge server. Fig. 4(center) highlights that the split and edge deployments outperform the one at the UAV, respectively, with 5 ms and 10 ms average reduction in the model execution time for each video frame.

Fig. 4(right), instead, shows the overall delay (including

Fig. 3: Analysis on YOLOv3's head model output tensor and CoTeD operations: (left) Video frame/activation tensor similarity; (center) CoTeD-Pruning effects; (right) CoTeD-Decomposition effects. Black bars denote standard deviation.

Fig. 4: (left) Activation tensor compression managed by CoTeD-Manager; (center) Averaged per-frame processing time (with standard deviation plotted in black bars) under different deployment scenarios; (right) Overall delay introduced by different techniques.

the transmission delay from the UAV to the edge) and the corresponding data size to be transferred for each video frame, when the communication bandwidth on the radio link is fixed at 50 Mbps. The plot compares the performance achieved by the edge deployment, the CoTeD split. When the whole DNN is deployed at the edge, the large raw video frames have to be transferred on the radio link, which always gives the largest transmission delay. For the split deployment, our CoTeD framework greatly outperforms its alternatives, in which the tensor reconstruction at the edge for CS and the tensor transmission for JPEG-100 take the longest time. Also, notice that the delay introduced by CS and JPEG-100 are at seconds or sub-seconds level, i.e., much higher than the video frame deadline requirement. To summarize, CoTeD provides a controllable activation tensor compression and transmission, which allows the split DNN's head and tail to meet the video frame deadline while effectively reducing the amount of data transferred over the radio link.

VI. Conclusions

We dealt with the execution of DNN-based computer vision tasks in scenarios where resource-limited UAVs can leverage the help of edge servers. We exploited a split computing approach to run the low-complexity head DNN at the UAV and the computationally heavier DNN tail at the edge, and addressed the problem of efficiently transmitting over the radio link the tensor from the head to the tail model. Our solution, called CoTeD, consists of a sequence of computational-light and dynamically configurable differential, pruning, and decomposition operators at the UAV, and a similar chain of operations for tensor reconstruction at the edge. The advantages of CoTeD in terms of processing latency and bandwidth consumption are very evident, with a reduction in latency by over 95%, 75%, and 80% when compared to (resp.) compressive sensing, JPEG-100, and all-at-the-edge deployment.

References

[1] X. Wu, W. Li, D. Hong, R. Tao, and Q. Du, "Deep learning for unmanned aerial vehicle-based object detection and tracking: A survey," *IEEE Geosc. and Rem. Sens. Mag.*, vol. 10, no. 1, pp. 91–124, 2022.

[2] Y. Kang and et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comp. Arch. News*, 2017.

[3] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[4] F. Malandrino, C. F. Chiasserini, and G. di Giacomo, "Efficient distributed DNNs in the mobile-edge-cloud continuum," *IEEE/ACM ToN*, vol. 31, no. 4, 2023.

[5] Y. Matsubara, M. Levorato, and S. Banerjee, "Split computing for complex object detectors," *arXiv:1905.09712*, 2019.

[6] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh, "Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems," *IEEE Access*, 2020.

[7] P. Datta, N. Ahuja, V. S. Somayazulu, and O. Tickoo, "A low-complexity approach to rate-distortion optimized variable bit-rate compression for split dnn computing," in *IEEE ICPR*, 2022.

[8] J. Emmons and et al., "Cracking open the DNN black-box: Video analytics with DNNs across the camera-cloud boundary," in *ACM HotEdgeVideo*, 2019.
[9] R. A. Cohen, H. Choi, and I. V. Bajić, "Lightweight compression of intermediate neural network features for collaborative intelligence," *IEEE Open J. of Circ. and Syst.*, vol. 2, pp. 350–362, 2021.
[10] P. Zhang, H. Tian, H. Luo, X. Li, and G. Nie, "A hybrid fast inference approach with distributed neural networks for edge computing enabled UAV swarm," *Physical Comm.*, vol. 60, 2023.
[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Adv. in Neur. Inf. Proc. Syst.*, 2016.
[12] F. Malandrino, G. di Giacomo, A. Karamzade, M. Levorato, and C. F. Chiasserini, "Tuning DNN model compression to resource and data availability in cooperative training," *IEEE/ACM ToN*, 2023.
[13] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, 2009.
[14] X.-Y. Liu and et al., "High-performance tensor learning primitives using gpu tensor cores," *IEEE Trans. on Comp.*, vol. 72, no. 6, 2023.
[15] S. Ding and et al., "A novel YOLOv3-tiny network for unmanned airship obstacle detection," in *IEEE DDCLS*, 2019.