Split Computing With Scalable Feature Compression for Visual Analytics on the Edge

Zhongzheng Yuan, Samyak Rawlekar, Siddharth Garg, Elza Erkip, *Fellow, IEEE*, and Yao Wang, *Fellow, IEEE*

Abstract—**Running deep visual analytics models for real-time applications is challenging for mobile devices. Offloading the computation to edge server can mitigate computation bottleneck at the mobile device, but may decrease the analytics performance due to the necessity of compressing the image data. We consider a "split computing" system to offload a part of the deep learning model's computation and introduce a novel learned feature compression approach with lightweight computation. We demonstrate the effectiveness of the split computing pipeline in performing computation offloading for the problems of object detection and image classification. Compared to compressing the raw images at the mobile, and running the analytics model on the decompressed images at the server, the proposed feature-compression approach can achieve significantly higher analytics performance at the same bit rate, while reducing the complexity at the mobile. We further propose a scalable feature compression approach, which facilitates adaptation to network bandwidth dynamics, while having comparable performance to the non-scalable approach.**

Index Terms—**Computer vision, feature compression, object detection, split computing.**

I. Introduction

DEEP learning models have achieved tremendous success in various visual analytics tasks. There is an increasing interest in leveraging the power of large neural network models on high-resolution image and video data to solve challenging problems. Being able to efficiently run these computationally intensive neural networks at a high frame rate is crucial for applications such as autonomous driving, navigation assistance for blind and visually impaired people, and augmented reality. However, end-user devices that capture the raw visual data are often constrained in computation power and cannot run these models with sufficient speed. Running these models also consumes significant energy, reducing battery life.

Manuscript received 10 December 2023; revised 12 March 2024 and 7 May 2024; accepted 12 May 2024. Date of publication 28 May 2024; date of current version 16 October 2024. This work was supported in part by NSF under Grant 2003182, and in part by Intel. The work of Zhongzheng Yuan and Yao Wang was supported by NSF under Grant 1952180. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Shiqi Wang.

Zhongzheng Yuan, Siddharth Garg, Elza Erkip, and Yao Wang are with the Electrical and Computer Engineering Department, New York University Tandon School of Engineering, Brooklyn, NY 11201 USA (e-mail: zy740@nyu.edu; sg175@nyu.edu; ee531@nyu.edu; yw523@nyu.edu).

Samyak Rawlekar is with the Department of Electrical and Computer Engineering, University of Illinois Urbana Champaign, Urbana, Illinois 61801 USA (e-mail: samyakr2@illinois.edu).

Digital Object Identifier 10.1109/TMM.2024.3406165

A solution to this problem is to employ edge computing, which transmits the acquired image data and offloads the computation to a nearby edge server with larger computation power. Efficient computation offloading requires optimizing the trade-off between multiple objectives including compressed data rate, analytics performance, and computation speed. There are two main frameworks for performing computation offloading. One is to first compress the image on the mobile device, then perform decompression and inference on the decompressed image at the server [1], [2]. The other approach, commonly known as *Collaborative Intelligence*, proposes to split the task model between the mobile and the server, and compress and send the intermediate features at the point of split to the server [3], [4], [5].

In either framework, compression can be performed using conventional, non-learned image/video compression standards. Such approaches may have practical advantages, because they can leverage existing hardware and software for compression, but the impact of compression on the analytics task's performance cannot be controlled directly. That is, the non-learned compression module cannot be optimized jointly with the task to optimize rate-performance tradeoffs. In contrast, using a learnable compression module enables direct optimization of the rate-performance trade-off; specifically, only features that are useful for the analytics task need to be generated and compressed. The split computation approach can also reduce the computation at the server, which could be important for applications where the server has resource constraints.

For these reasons, our work adopts the split computing framework and introduces a novel learning-based approach for feature compression with lightweight computation on the mobile side. We train the feature compression and decompression modules together with the analytics model to optimize the analytics task performance under a rate constraint. We demonstrate the effectiveness of the split computing approach with learned feature compression on two common computer vision tasks: object detection and image classification. Compared to baseline methods that apply either standard image compression or learned image compression at the mobile and perform decompression and visual analytics at the edge, the proposed system achieves higher object detection and classification accuracy in the low to medium rate range, while requiring substantially lower compute time on the mobile device.

While tailoring the split computing model for each target bitrate is likely to achieve the highest analytical performance for that bitrate, it may not be practically feasible to store multiple models on the client device to adapt to different bandwidth conditions. Scalable compression, a well explored concept that has been adopted in video compression standards [6], [7], is one possible method to achieve variable rate encoding. In scalable compression, a layered bitstream is generated by the compressor. The base layer bitstream provides a basic level of analytics performance, and each additional enhancement bitstream provides an incremental performance improvement. Thus, the encoder can quickly adapt to changing bandwidth by generating and sending the maximum number of layers that the current bandwidth allows. We propose the *first* scalable learned feature compression model and a corresponding strategy to train this model jointly with the task model. The resulting single scalable compression model and corresponding task model can achieve competitive performance over the entire rate range, compared to the non-scalable approach, which uses separately optimized compression and task models for each target bit rate.

Finally, we explore the possibility of reconstructing the original image using learned features for analytics. The intermediate features in the split computating framework are generated by a model trained specifically for a target task but not for human visualization. In some applications, it may be important for a human operator to visually inspect the image to verify the detection or classification result. We show that it is possible to recover a degraded version of the image from the compressed feature, which is sufficient for human verification in most instances.

Our contributions are summarized as follows:

- We propose a novel lightweight feature compression scheme embedded within the split computing framework. To exploit the cross-channel and spatial redundancy among the multi-channel features, we perform channel and spatial dimensionality reduction and further decorrelate the resulting channels. We use the compression architecture of [8] to encode the reduced features as Gaussian variables with mean and scale predicted by a side hyperprior bitstream.
- We demonstrate the effectiveness of the proposed scheme for both object detection and image classification. Our approach achieves higher task accuracy under similar bit rates, and significantly faster inference speeds, compared to the compression-decompression-analytics baselines.
- Our work is the first to introduce bitrate scalability for task-aware compression. We propose a scalable compression model for generating a layered bitstream that enables variable bitrate and efficient adaptability to changing bandwidth constraints. Our proposed model is also the first model that can achieve variable bitrate while using only a single model without the need for model switching.
- We show that it is possible to train an image reconstruction model to recover a degraded version of the original image from the compressed features that are extracted for object detection.

Preliminary version of this work has been presented in [9], [10]. The two prior works considered the object detection task only, while here we demonstrate the success of the proposed feature compression approach for image classification as well.

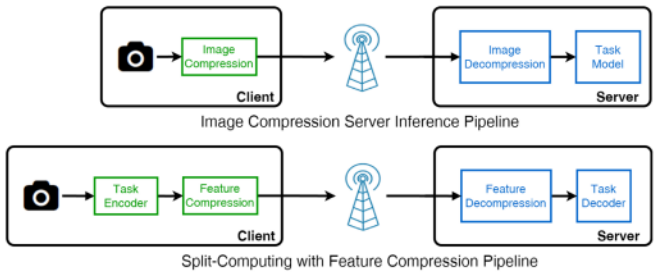


Fig. 1. Two pipelines for computation offloading. 1) Compress the image at the client side, then feed the decompressed image to the task model at the server. 2) Split the task model between the client and the server. Compress and transmit the intermediate features.

The scalable compression approach reported in [10] used multiple task decoder models to achieve good rate-task performance for different bitrate ranges. Here we show that a single task decoder is sufficient to achieve good performance over the entire rate range.

II. Related Works

The method of compressing imagery data for the purpose of computation offloading has been widely studied by several research groups and standardization efforts. The Motion Picture Expert Group (MPEG) has recently started a standardization effort known as “Video Coding for Machines” (VCM) [11], [12]. MPEG-VCM comprises of two main tracks: VCM and Feature Compression for VCM (FCVCM). VCM focuses on compressing the input image/video followed by decompression and inference on the server, while FCVCM focuses on compressing the intermediate feature of the task network. The FCVCM track fits into the broader framework of computational offloading known as Collaborative Intelligence, where the computation of an inference pipeline is split between the mobile device and the server. These two methods are illustrated in Fig. 1.

Here we review some representative works of learned compression modules for these two pipelines.

A. *Analytics-Aware Image Compression*

In the image compression approach for computation offloading, the input image is compressed by an image compression model and sent to the server. The server performs decompression and feeds the reconstructed image to the task model. This approach has the advantage that existing image/video compression standards can be utilized, as well as protocols for transporting the compressed bitstream. However, existing image/video compression standards were not optimized for analytics, but rather for image reconstruction. Several prior works have tried to address this problem by learning image compression models that better preserve analytics performance.

A task-aware JPEG compression model was proposed in [13]. The work proposes to use a convolutional network to predict the quantization table for DCT coefficients for JPEG compression. Images compressed using the predicted quantization table achieved better task performance than when the standard table was used. In [1], [2], the learned image compression model with hyperprior [8] was used to compress images, and then the decompressed images were fed to a detection or segmentation model. By jointly training the compression and task models, some performance loss due to compression is recovered. These prior works require relatively large convolutional networks to perform image compression on the mobile device, which still consumes substantial computation time and battery.

B. Analytics-Aware Feature Compression

Collaborative intelligence (CI) is another approach for offloading computation from the mobile device to the server. In contrast to the image compression approach, where inference of the task model is performed fully on the server side, the mobile and the server each share a part of the computation of the task model in collaborative intelligence [3]. CI for deep neural networks was first proposed by [4], which demonstrated that latency and energy consumption can be reduced by splitting a deep neural network between a mobile device and an edge server.

Compression of the intermediate feature is necessary to reduce the bitrate for transmission. There has been no widely adopted standard for compressing the features of a model. Several works have proposed using existing image/video compression standards or learned compression models for feature compression.

Intermediate feature compression using standard image/video codec was studied in [11], [14] with HEVC, and in [15] with JPEG and additional dimensionality reduction. Using standard codecs for feature compression in general did not achieve good performance, as the codecs were designed for compression of images and not features.

Feature compression with learned image compression model was proposed in [16]. The hyperprior compression model was used and the task model was end-to-end trained. However, the split point considered by [16] was at the second to the last layer of the original deep learning model, so that the mobile device still has to do a majority of the computation task. In [17], a learned feature dimension reduction and entropy coding approach was proposed. The model achieved good rate-task performance, but switching between different feature compression modules was still necessary to compress at different bitrates. In addition, because only the feature compression modules were trained, its performance was lower compared to end-to-end trained models. Another feature compression approach for CI was proposed in [18], where a learned entropy compression model was used to compress the features and the input image was downsampled to reduce spatial redundancy.

The standardization of feature compression for analytics has also started recently, with the FC-VCM standardization effort calling for proposals of feature compression solutions. In [19], the intermediate features were stacked in a matrix and PCA was performed to transform the matrix into coefficients. The coefficients are compressed using the VVC standard video codec. A similar strategy was used in [20], except that a trained neural network was used instead of the PCA to transform the intermediate features. The methods both use standard video codecs to compress the intermediate bitstream, which are not end-to-end trainable and may be suboptimal to end-to-end trained methods.

Our approach to feature compression is most similar to that of [17] and [18]. Compared to [17], we also employ a feature dimension reduction module, and in addition, we trained our model end-to-end and used the hyperprior model for better rate reduction. Compared to [18], our model do not downsample the input image, which may lead to decreased performance for the detection of smaller objects. Both works do not have a variable rate model that does not require model switching. Our scalable compression model, which we describe below, enables variable rate compression with single model.

C. Scalable Compression

Scalable image compression (also known as layered compression or coding) encodes an image into a base layer z_1 and additional enhancement layers z_2, z_3,..., z_M for a total of M layers. The layers are embedded in that layer m is useful and contribute to improved quality only if all previous layers up to $m-1$ are available [21].

The sender can adaptively generate and send a set of layers given the current communication channel throughput. When the decoder receives the base layer bitstream only, it can decode the image with a basic reconstruction quality. With each additional enhancement bitstream received, the decoder can decode the image with successively higher quality. When the same image or video is sent to multiple receivers with different network throughputs such as in a live broadcast, scalable compression facilitates the sharing of the lower layers among multiple receivers, reducing the total traffic load in the network.

Another benefit of layered coding is that lower layers can be protected using stronger error correction or delivered over a more reliable channel, so that lower layers can be successfully delivered with a high probability, leading to more graceful performance degradation when the network connectivity is poor [6]. Compared to non-scalable compression, which needs to receive the entire bitstream in order to decode an image at a fixed bitrate, scalable compression offers more robustness to the communication channel variability.

In the case of feature compression for analytics, scalability can be considered as generating a bitstream with multiple layers, such that each additional layer leads to improvement in the analytics performance. In practical applications, the mobile device is often deployed in areas with weak and unstable internet connections. In such scenarios, scalable compression can be particularly useful. In the case of a sudden drop in the network throughput (e.g., switching from 5 G to 4 G wireless network), the base layer can be transmitted to ensure a basic performance in analytics. When the network condition improves, additional layers can be generated and sent.

There have been several works on learned scalable compression of images for human visualization. In [22], scalability is

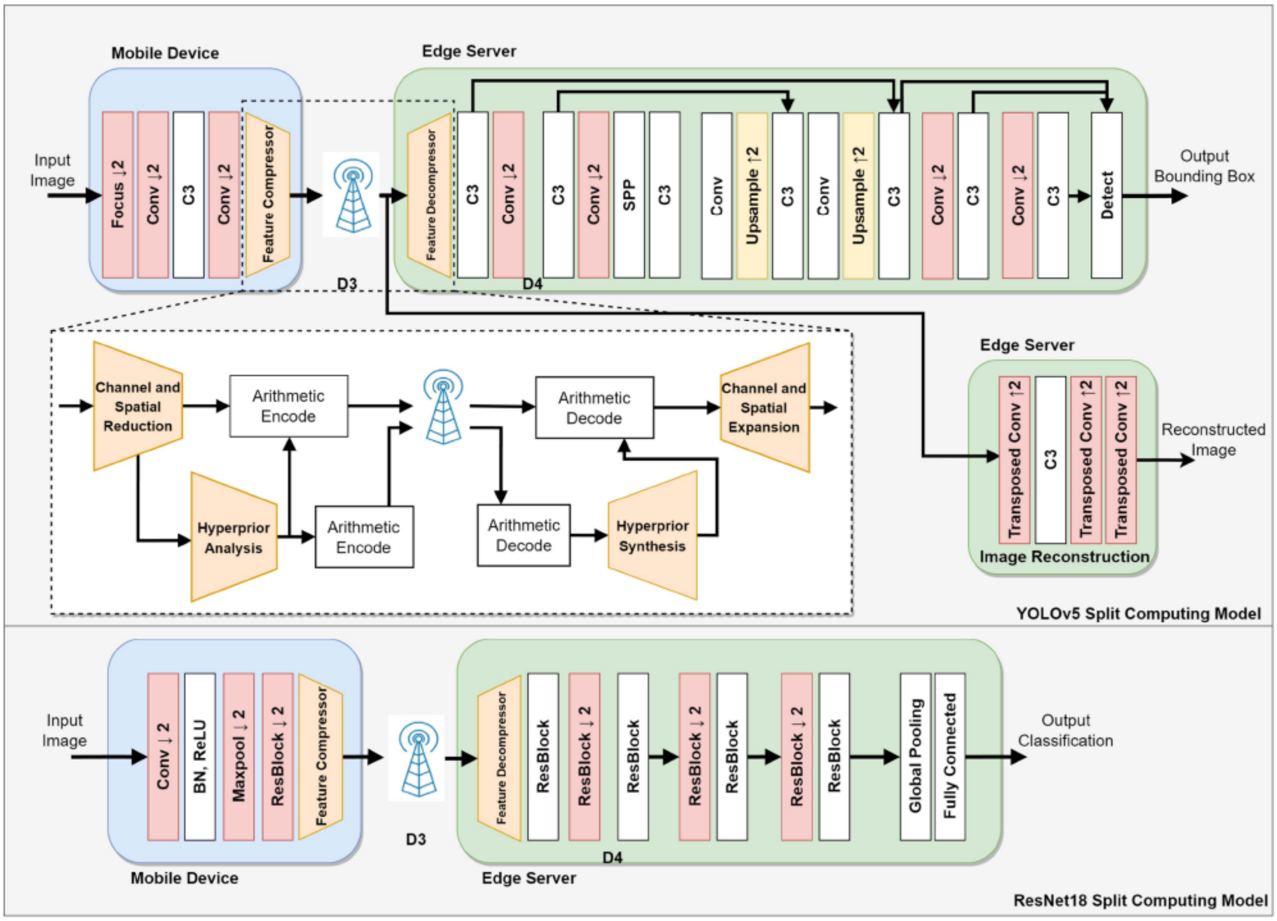


Fig. 2. Overview of the proposed system. The task model is split into two parts that run on the mobile device and the edge server, respectively. A feature compression model is used to compress the intermediate features at D3. Another split point (D4) considered in our experiments is also indicated. The notation ↓2 and ↑2 refers to down-sample by a factor of 2 and up-sample by a factor of 2 respectively. (a) For object detection using the YOLOv5 architecture. When splitting at D4, the first skip connection of the YOLO model was removed. The C3 layer which originally receives this skip connection was modified with less input channels. (b) For image classification using the ResNet 18 architecture.

achieved by using multiple encoder models to successively compress the residual of the reconstructed image and send the residual latent information in layers. Another model proposed by [23] encodes the input image to layered latent features and uses lower layer latents to predict and enhance the higher layer bitstreams. More recently, a fine-grained scalable model is proposed by [24]. The model generates a base and an enhancement feature tensor. The base feature is sent as a whole, while the enhancement feature tensor is split along the channel dimension, and each channel is sent one by one for each enhancement layer.

Some prior works have considered scalable compression for analytics, such as [25], and [26]. However, the scalability proposed by these works refers to the ability of the server model to perform additional analytics tasks as it receives each additional bitstream, while our proposed scalable model increases the accuracy of a single task with additional layers.

III. Feature Compression for Split Computing

A. Proposed Model

We create a split computing model by splitting the task model into two parts, with the first part running on the mobile device and the second part running on the server. This paper calls the mobile part of the model *the task encoder* $\mathcal{F}$, and the server part of the model *the task decoder* $\mathcal{G}$. The intermediate features at the point of split are compressed and transmitted from the mobile to the server.

Choosing the point of split is an important consideration as it affects the amount of computation needed to be done by the mobile device. In general, a split point that is deeper into the task model results in sparser features that are easier to compress. However, splitting deeper is counter to the goal of computational offloading, as a larger percentage of the task model needs to execute on the mobile.

We observe that in typical learned image compression models [8], [27], the model architecture involves the use of strided-convolutions or downsample layers to reduce the spatial dimension of the input. In the learned image compression model proposed by [8], [27], the input image undergoes four $2\times$ down-sample convolution layers before being entropy encoded. Interestingly, several task models [28], [29], including the ones we use, also contain strided-convolutions or $2\times$ downsampling. Motivated by these similarities, we place the split point in our task models after the fourth down-sampling layer (D4), mirroring the architecture of [8], [27]. We also consider splitting after the third down-sampling layer (D3) for less computation on the

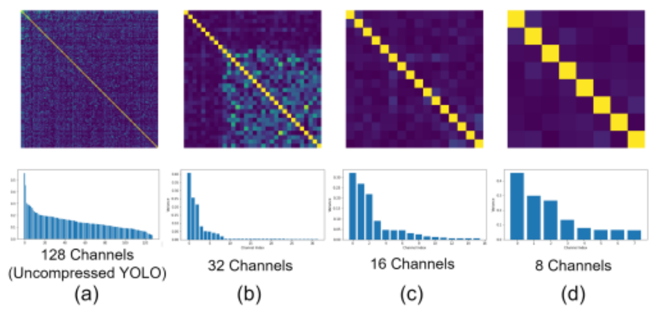


Fig. 3. Inter-channel covariance matrix (top row) and the variance of each channel (bottom row) for the intermediate features from the YOLOv5 model at the D3 split point, for the original feature channels and after channel reduction to different number of channels.

mobile side. In typical task models, these points of split are early in the task model, so the share of computation allocated for the mobile device is small. This methodology can be applied to a range of different computer vision models. We demonstrate this by showing its effectiveness in the YOLOv5 model [28] for object detection, and the ResNet18 model for image classification [29]. An overview of the proposed split computing model for object detection and classification is shown in Fig. 2.

B. Feature Dimensionality Reduction

It is common for convolutional network models to increase the number of channels at each downsample layer. While having a large number of channels is beneficial for task performance, there is significant redundancy among the channels, which can be seen in the inter-channel covariance matrix, as shown in Fig. 3(a). Coding these channels directly and independently does not exploit the redundancy between channels. We propose to first reduce the channel dimension of the features before performing entropy coding. We use a 1×1 convolution layer to reduce the number of channels from N to N_R. In addition to reducing the number of channels, this layer also serves to decorrelate the resulting channels. In the receiver, we use a reverse 1×1 convolution layer to increase the channel number back to N. As shown in Fig. 3(b), the first 10 channels capture most of the total variance of the original $N = 128$ channels.

Similarly, we reduce the spatial dimension of the features by down-sampling. At the encoder side, we use a convolution layer with a kernel size of 5×5 and a stride of 2 to down-sample the features by a factor of 2. At the decoder side, 5×5 transposed convolution is used to upsample the features back to the original spatial dimension. The reduction and expansion layers are placed before the nonlinear activation of the output layer at the point of split. Similar to compression models [8], [27] where there is no activation function at the last layer of the encoder, we do not use nonlinear activation in the feature compression layers because the entropy coder expects a Gaussian distribution, while nonlinear activation in the model typically produces a single-sided distribution. We perform spatial reduction in addition to channel reduction for the split point at D3 so that the resulting spatial dimension is the same as in D4, while only channel reduction is performed at the D4 split point.

Reducing the channel and spatial dimensions has the added benefit that it reduces the computation time for arithmetic coding. We observed that larger tensor size leads to longer arithmetic encoding time, as shown in our results Section VI-F. Reducing the dimension of the feature tensor helps increase the overall inference speed.

C. Rate-Task Loss

In the learned image compression framework proposed by [27], the image compression model is trained through minimizing a rate-distortion loss,

$$\begin{aligned} L &= L_R + \lambda \cdot L_D \\ L_R &= \mathbb{E}_{x\sim p_x}[-\log_2 p(\hat{y}(x;\theta))] \\ L_D &= \mathbb{E}_{x\sim p_x}[d(x, \hat{x}(\hat{y};\theta))], \end{aligned} \tag{1}$$

where λ is a hyper-parameter that controls the rate-distortion trade-off, $\hat{y}$ is the quantized feature vector, and $d(x, \hat{x})$ is the distortion between the original image x and the decoded image $\hat{x}$. Models with different compression bitrates were trained by setting the λ hyper-parameter to different values.

In our framework, y is the feature vector to be compressed at the point of split, and we further perform channel and spatial reduction to obtain the reduced feature vector z. To optimize for task performance, we replace the distortion loss by the task loss, L_{task}. For example, we use the cross-entropy loss for the classification task, and the combination of object presence loss, object class loss, and box coordinate loss for the object detection task [30]. We use the combined Rate-Task loss to train the model end-to-end:

$$\begin{aligned} L &= L_R + \lambda \cdot L_{task} \\ L_R &= \mathbb{E}_{x\sim p_x}[-\log_2 p(\hat{z}(y(x;\theta);\phi))] \end{aligned} \tag{2}$$

where θ indicates the task model parameters, and ϕ the feature compression model parameters. Instead of directly entropy coding the quantized z, $\hat{z}$, we follow the idea of hyperprior in [8] to more efficiently perform entropy coding. We generate a hyperprior feature vector z_h from z using a hyperprior analysis model. z_h is quantized to $\hat{z}_h$ and entropy encoded to be included in the bitstream as side information. $\hat{z}_h$ is decoded by a hyperprior synthesis model to predict the probability distribution of z for entropy coding of $\hat{z}$. The rate loss term would include the entropy for both $\hat{z}$ and $\hat{z}_h$.

D. Reduction/Expansion Layers Pre-Training

Instead of training the entire model (including the task model and the feature compression model) with the Rate-Task loss from scratch, we found it to be beneficial to pre-train the channel and spatial reduction/expansion layers in the feature compression model, while keeping the task model to be the same as the pre-trained task model without rate constraint. If these layers are randomly initialized, inserting these layers into the pretrained task model severely reduces the overall performance. Therefore, we pre-train these layers with the MSE loss between the input to the reduction layer and the output of the expansion layer. This

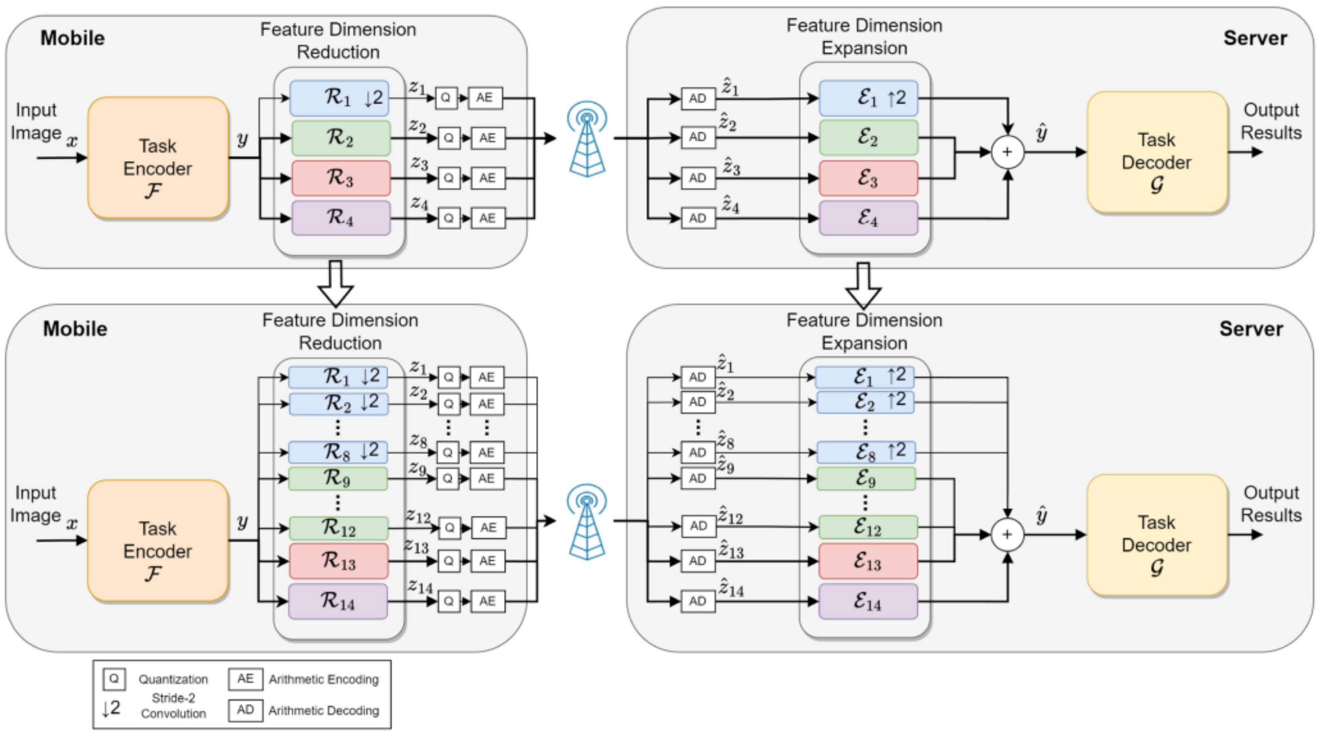


Fig. 4. Overview of the proposed scalable compression model. The model creates scalable layers by generating and entropy coding different groups of reduced feature channels from the task encoder. In this example, we first train a model with four layers with each layer having 8 channels. We then split the base layer into 8 layers of 1 channel each and the second layer into 4 layers of 2 channels each. This model with $M=14$ layers is then further trained to cover a wider rate range when the number of received layers l varies from 1 to 14.

step helps to initialize the reduction and expansion layers to better reconstruct the feature y. Further refinement of the entire model serves to reduce the correlation between the features in z (through the rate loss) and adapt the task model layers to work with the reconstructed features y from quantized z.

IV. Scalable Feature Compression

In the work just presented, we assume a different split computing model (including a task model and a compression model) is trained for each target rate by selecting a different lambda. We extend such non-scalable model to enable rate-performance scalability for feature compression. We transform the original feature tensor with multiple feature channels into groups of channels, each with a small number of channels, to form different scalable layers.

We first perform dimensionality reduction to the intermediate feature at the split point to reduce the features into M groups of features $z_m, m = 1, 2, \ldots, M$, each with a small number of channels. Unlike the single dimensionality reduction layer in the non-scalable model, the scalable model uses M separate convolution layers $\mathcal{R}_m, m = 1, 2, \ldots, M$, to generate M groups of features $z_m = \mathcal{R}_m(y)$ with N_{R_m} channels. A separate hyperprior model is trained to estimate the mean and variance parameters for each z_m, which are individually quantized and entropy encoded using their respective hyperpriors.

At the server side, M separate dimension expansion layers $\mathcal{E}_m$ are used to expand all received dequantized features $\hat{z}_m$ back to N channels and the original spatial dimension of y. The expanded tensors are added together to produce the input to the task decoder. During inference, if only l scalable layers are received, the recovered feature $\hat{y}$ is the sum of all received and expanded tensors:

$$\hat{y} = \sum_{m=1}^{l} \mathcal{E}_m(\hat{z}_m), l \in \{1, 2, \ldots M\}. \tag{3}$$

This combined feature is then input to the task decoder model to produce the analytics result $t = \mathcal{G}(\hat{y})$. An overview of our scalable model is shown in Fig. 4.

A. *Multi-Round Refinement of Scalable Layers Using Rate-Task Loss*

With a pre-trained YOLO model, we first pre-train the reduction/expansion modules by minimizing the MSE loss between the original and reconstructed features with all scalable layers activated. We then refine the model end-to-end with the rate-task loss using a training strategy that updates all scalable layers iteratively. For each batch of training data, we input the batch into the model over M rounds. In round l, only layers 1 to l are activated, for l from 1 to M. The model is updated using the loss corresponding to having only layers up to l:

$$L_l = \sum_{m=1}^{l} L_{Rate}^{m} + \lambda_l \cdot L_{Task}^{l}, \tag{4}$$

TABLE I
LAMBDA VALUE AND TOTAL NUMBER OF FEATURES CHANNELS UP TO l FOR EACH SCALABLE LAYER l

Number of scalable layers l	1	2	3	4	5	6	7	8	9	10	11	12
Number of channels up to l	1	2	3	4	5	6	7	8	12	16	24	32
λ_l	0.125	0.25	0.375	0.5	0.625	0.75	0.875	1	2	3	6	12

where L^m_{Rate} is the rate loss for th m-th layer, L^l_{Task} is the task loss when using up to l layers. After going through M rounds, the compression and decompression modules corresponding to all layers as well as the shared task encoder and decoder will be updated. We experimented with different λ_l values for training and found that good performance can be achieved by setting λ_l depending on the number of channels up to that layer. We set λ_l according to Table I.

Note that with this strategy, the compression and expansion modules for the lower layers are updated more times than the higher layers. This is appropriate because the lower-layer affects task-rate performance over a larger rate range than the higher layers, due to the embedded nature of the layered bitstream. We have found that this training strategy yields better performance over the entire rate range than some alternative approaches, including progressive training, where we first train only the base layer compression modules and task modules, and then train the second compression layer, while fixing the base compression layer and the task modules, and so on. The progressive approach would optimize the task modules only for the lowest rate, yielding suboptimal performance over the entire rate range.

B. Split Layer Training

We found that directly training the scalable model for many rate points will lead to low rate-task performance. For example, if we initialize the task encoder and decoder from a pretrained YOLO model and directly train 14 scalable layers with the scalable loss, the rate-task performance will be much lower than the non-scalable models.

Therefore, we first trained a model with a small number of scalable layers spanning a large rate range. Specifically, we trained a model with 4 scalable layers each with a channel size of 8. This model achieved good accuracy for the high bitrate points, but the base layer accuracy was slightly lower.

In order to create more operational rate points, we perform layer splitting to generate more scalable layers for the low to mid bitrate range. From the 4 scalable layer model, we split the base layer with 8 channels to 8 layers of 1 channel each. We also split the first enhancement layer with 8 channels to 4 layers of 2 channels each, resulting in a total of 14 scalable layers. In performing splitting to the feature dimension reduction/expansion modules, we separated the weight tensors of the convolution layers along the channel dimension, so that new independently operating convolution layers were formed for the new layers.

More generally, starting with a trained model that generates M layers, the original base feature tensor that consists of N_{R_1} channels is subdivided into M' tensors with $N'_{R_1}, N'_{R_2}, \dots, N'_{R_{M'}}$ channels respectively. This has the effect of splitting the original base layer into M' scalable layers for the lower bitrate range, and the model after the split will consist of $M + M' - 1$ scalable layers. Subsequent enhancement layers can be similarly subdivided into additional scalable layers, depending on the number of scalable layers that are required for a particular use scenario.

After splitting the layers, a new set of hyperprior models for each of the layers is initialized, and each newly formed layer is coded independently. We then train the entire scalable model end-to-end using the same multi-round training strategy using the loss in (4), for both the newly splitted layers and the un-split layers.

V. IMAGE RECONSTRUCTION

As described in the Introduction, in certain applications, it is helpful if the server can reconstruct the images from the received features, for example, to provide a human observer a low-quality rendition of the remote scene who can in turn verify the object detection results. In order to reconstruct the image from compressed features (after the channel and spatial expansion), we train an image reconstruction model to invert the down-sampling process in obtaining the features.

The reconstruction model follows the same architecture as the task encoder part of the split computing model, with the number of input channels and the number of output channels interchanged at each layers. The stride-2 convolution layers are replaced with stride-2 transposed convolution layers. And the number of input and output channels of each layer in the encoder are reversed in the reconstruction model.

We train the reconstruction model by minimizing a distortion loss between the original input image and the reconstructed output image. We tested both MSE and MS-SSIM as the distortion loss in our experiments, but we found that using both losses led to similar reconstruction results. During training, only the weights of the image reconstruction model are updated, while the weights of the split computing model, including the task model and the feature compression model, are frozen. Because the split computing model is not modified, training the reconstruction model does not affect the rate-task performance.

VI. RESULTS

In this section, we discuss the results of experiments conducted on our proposed models. First, we present the results for the non-scalable split computing approach and demonstrate its effectiveness compared to the image compression approach, in terms of rate-task performance and computation time. We then describe the results for our proposed scalable compression model and show that it can achieve performance comparable to the non-scalable model. Furthermore, we draw comparison between our method and the use of PCA for obtaining transforms for scalable layers, demonstrating that learned transforms optimized for rate-task performance substantially outperform transforms obtained by PCA. Finally, we show visual results of image reconstruction using features optimized for object detection.

A. Non-Scalable Model for Object Detection

The Ultralytics YOLOv5 model [28] was used as the task model architecture for object detection. The smaller-sized

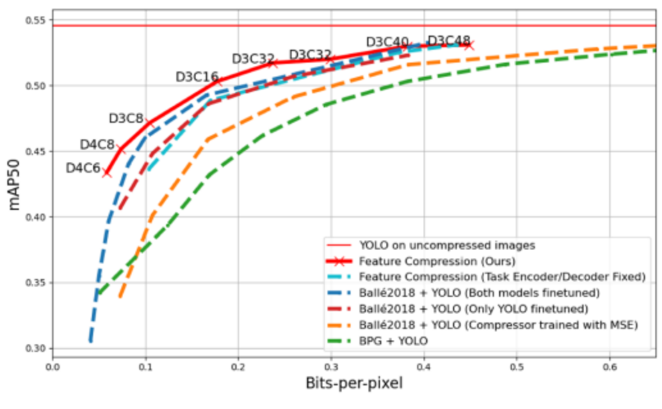
YOLOv5s model with 7.2 million parameters was chosen for faster training and inference speed. The model was initialized with weights provided by [28], which were trained with images from the entire COCO training set. We used the same strategy of data augmentation as in the pre-trained model. The original images were resized to 640 $\times$ 640 before being passed as input to the model.

We trained models at different bitrates by varying the number of compressed channels N_R, the point of split $D3$, $D4$, and the hyperparameter λ. From these models, we selected the models that achieve the best rate-task trade-off to draw a Pareto curve of task-accuracy vs. bitrate. We compared our models with three different baselines using the compression-decompression-analytics frame work. In the first baseline, we used the BPG for image compression, which is based on the intra-coding scheme of the H.265 video compression standard [31], followed by the pretrained task model without fine-tuning. The range of QP (Quantization Parameter) used for BPG is 29 - 45 for the Full COCO dataset, and 43 - 49 for the COCO-Traffic dataset. In the second baseline, the learned image compression model by [8] was used as the image compressor. The compression model was trained using MSE-loss for reconstruction and the task model was not fine-tuned for compressed images. In the third baseline, we fine-tuned both the image compression model and the task model end-to-end using the same rate-task loss from (2).

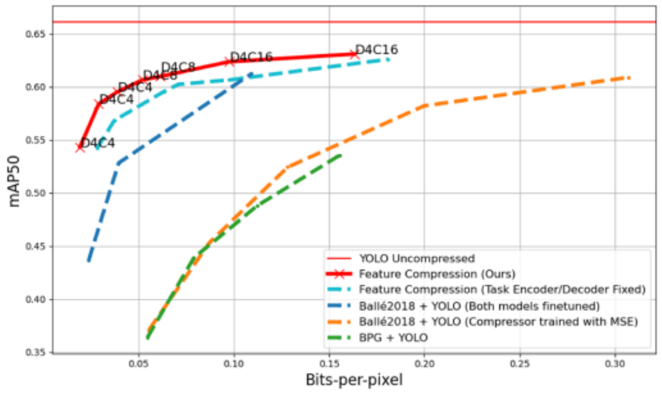
In some applications such as traffic monitoring or pedestrian navigation, it may be unnecessary to detect all 80 classes of objects in COCO. Therefore, we also performed an experiment on a smaller subset of the COCO dataset to demonstrate the potential for even better rate-accuracy performance under a limited number of object classes. From the COCO dataset, we picked 9 classes of objects, including Person, Car, Bus, Truck, Motorcycle, Traffic Light, Fire Hydrant, Stop Sign, and Parking Meter, that are relevant for traffic-related applications. We extracted images from the COCO dataset that includes at least one instance of the 9 classes into a dataset called the COCO-Traffic dataset [32]. We trained and evaluated our models on the COCO-Traffic dataset. In addition, we also evaluated the models trained on the COCO-Traffic dataset on the TJU-DHD dataset [33], which is a high-resolution object detection dataset for traffic scenes, to demonstrate our model's ability to detect on higher resolution images.

1) Rate vs. Detection Accuracy Performance: The performance for object detection on the entire COCO test set is shown in Fig. 5. The detection accuracy is measured in mAP50, or mean-Average-Precision using a 0.50 IOU threshold. Our feature compression model is shown to be outperforming the three baselines. In low to medium bitrates, our model achieved a higher mAP than all the baselines. At the high bitrates, our model achieves comparable mAP as the learned compression model fine-tuned using the Rate-Task loss.

For the experiment with the COCO-traffic dataset, we only evaluated models at the D4 split point and focused on the low bitrate region. By focusing on a more specific set of objects, the features can be compressed to very low bitrates (by using very few channels) while still maintaining high mAP (see Fig. 5(b)).



(a) Full COCO Dataset (80 object classes)



(b) COCO-Traffic Dataset (9 object classes)

Fig. 5. Detection performance under various bitrates for the full COCO dataset and the COCO-Traffic dataset. Points on the curves for the proposed method are labeled with their respective compression configurations. For example, D4C6 refers to compression at split point D4 and reduction to 6 channels.

The results are significant for application settings with low communication bandwidth and require compression into extremely low bitrates. Our split computing model enables highly accurate object detection while using low bandwidth transmission.

2) Fixed Task Encoder and Decoder Training: In some practical applications, it may be desirable to use a fixed pair of task encoder and decoder when integrating it in a split computing scenario. This enables the system to reuse a task encoder/decoder pre-trained from a large dataset without considering the compression artifacts and to switch only the compression/decompression modules when the network throughput changes. In such scenarios, only the feature compression layer should be trained by the rate-task loss while the rest of the analytics model remains fixed. We evaluate the performance loss due to this practical constraint.

We use a pre-trained YOLO model [28] as the basis for the task encoder and decoder and insert the non-scalable feature compression modules. The compression modules are then trained with the rate-task loss, while the task encoder and decoder are fixed. This approach is similar to that of [17], in which variable rate compression is achieved by switching the compression layers while the task encoder and decoder share the same weights

TABLE II
BREAKDOWN OF RUNTIME (MILLISECONDS) PER IMAGE (640 × 640 PIXELS) FOR THE PROPOSED SPLIT COMPUTING YOLO MODEL AND BASELINES

		Ballé2018 + YOLO (both models fine-tuned) (0.050 bpp, 0.404 bpp)	BPG + YOLO (0.0573 bpp)	BPG + YOLO (0.382 bpp)		D4C6 (0.0585 bpp)	D3C40 (0.382 bpp)	YOLO on Mobile
Mobile Device (CPU)	Image Compression	733.70	132.33	200.02	YOLO Pre-split Feature Compression	254.46 10.12	188.15 26.78	526.08
Edge Server (GPU)	Image Decompression	30.31	94.77	119.00	Feature Decompression	9.09	24.34	0
	YOLO	7.39	7.39	7.39	YOLO Post-split	5.3	6.6	
Total time on Mobile		733.70	132.33	200.02		264.66	214.92	526.08
Total time on Server		37.7	102.16	126.39		14.39	30.94	0
Total time		771.40	234.49	326.41		**279.05**	**245.86**	526.08
Detection mAP50		0.355, 0.533	0.342	0.503		**0.434**	**0.530**	0.546

D4c6 and D3c40 refer to models with split point at d4 and d3 and channel reduction to *nr* = 6 and *nr* = 40, respectively.

TABLE III
BREAKDOWN OF RUNTIME (MILLISECONDS) PER IMAGE (360 × 360 PIXELS) FOR THE PROPOSED SPLIT COMPUTING RESNET18 AND BASELINES

		Ballé2018 + ResNet18 (both models fine-tuned) (0.051 bpp, 0.316 bpp)	BPG + ResNet18 (0.048 bpp)	BPG + ResNet18 (0.350 bpp)		D3C4 (0.046 bpp)	D4C64 (0.329 bpp)	ResNet18 on Mobile
Mobile Device (CPU)	Image Compression	181.94	74.31	104.66	ResNet18 Pre-split Feature Compression	64.27 7.79	83.14 16.54	130.15
Edge Server (GPU)	Image Decompression	15.71	36.18	41.63	Feature Decompression	7.62	10.10	0
	ResNet18	2.21	2.21	2.21	ResNet18 Post-split	1.78	1.21	
Total time on Mobile		181.94	74.31	104.66		72.06	99.68	130.15
Total time on Server		17.92	38.39	43.84		9.40	11.31	0
Total time		199.86	112.70	148.5		**81.46**	**110.99**	130.15
Classification accuracy		0.530, 0.656	0.182	0.578		**0.592**	**0.677**	0.705

for all bitrates. As expected, the rate-analytics performance with this approach is significantly lower than the end-to-end trained models (see Fig. 5). On one hand, this result demonstrates that end-to-end training of all modules can lead to significant performance gain. On the other hand, the performance with the fixed task model is still far better than than the image compression approach when both the task model and compression models are fine tuned, when the object detection task focuses on a small number of application-specific classes (see Fig. 5(b))

3) Runtime-Analysis: We ran our models and the baselines in a setting that may be feasible in a practical scenario. For computations ran on the mobile device, we used a 1.1 GHz CPU processor. For neural network computations on the server side, we used an Nvidia RTX-8000 GPU. Entropy coding and decoding, however, are not parallelized and are ran on the CPU for both the mobile and server.

A breakdown of the inference time for our feature compression model and the baselines is shown in Table II. Compared to running YOLO locally, our model achieved a 47% and 53% reduction in the total inference time for the low and high bitrate models, respectively. At high bitrates, our split computing approach has a clear advantage over both baselines in the total inference time. At low bitrates, although our model has a slightly longer runtime than BPG + YOLO, our detection performance is superior over both baselines. The baseline using learned image compression followed by YOLO turns out to be not viable at least for the setting considered here, since its total inference time is longer than running YOLO locally.

B. Non-Scalalable Image Classification

For our experiments on the classification task, we adopted the ResNet18 model implemented by the torchvision package [34], and we used weights pretrained using the ImageNet dataset as initialization for our training. To combat overfitting, we used the

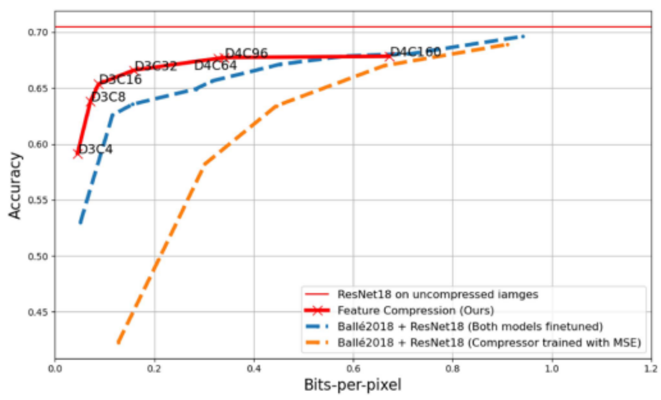


Fig. 6. Classification Accuracy under various bitrates for the ImageNet dataset.

AutoAugment augmentation strategy as described in [35] for training with ImageNet. The input original images were resized to 256 × 256 for faster training. For evaluation, an image size of 320 × 320 was used. The rate vs. classification accuracy curve is shown in Fig. 6. Our split computing classification model again achieved better accuracy against the baselines across different bitrates.

Using the same settings, we performed runtime analysis on the classification models. The results are shown in Table III. Our split computing model achieved a lower total inference time than both baselines. Compared to running ResNet18 locally, our model achieved a 37% and 15% reduction in total inference time for the low and high bitrate models, respectively.

The amount of time saving in this experiment is less than that of YOLO, because compared to YOLO, ResNet18 is more computationally expensive in the early parts of the model. If a more complex classification model is needed for higher accuracy, for example ResNet50, we expect that there would be more significant time saving with our split computing methodology.

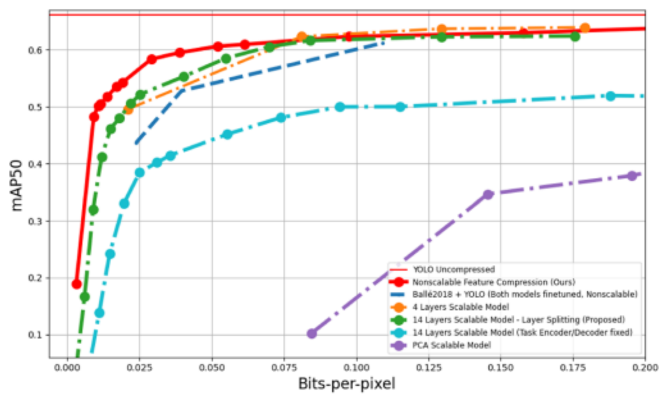


Fig. 7. Rate-Accuracy performance for the proposed scalable model compared to non-scalable compression baselines, for the COCO-Traffic dataset.

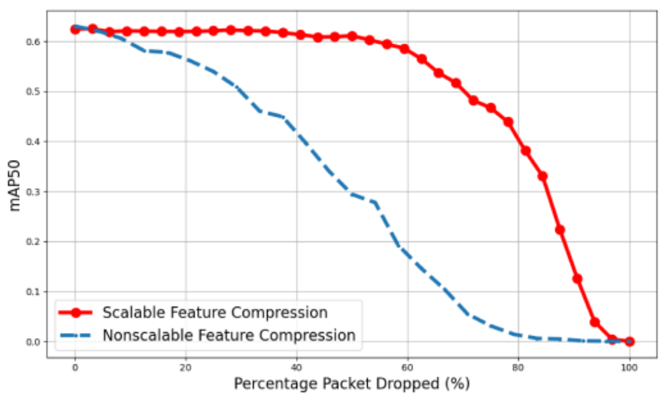


Fig. 9. Detection accuracy of the proposed scalable model compared to the non-scalable model under different packet drop rate.

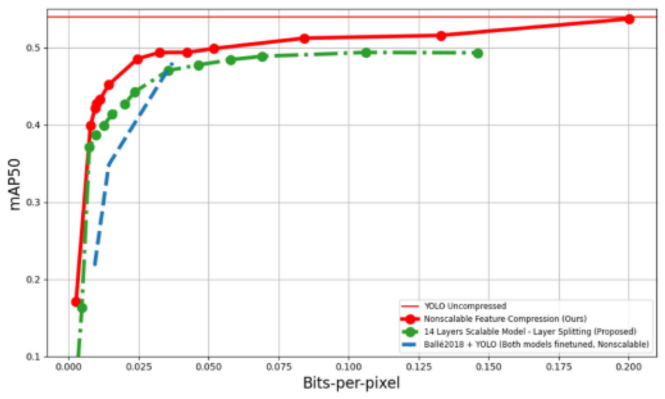


Fig. 8. Rate-Accuracy performance for the proposed scalable model compared to non-scalable compression baselines. Trained on the COCO-Traffic dataset, evaluated on the TJU-DHD dataset.

C. Scalable Compression for Object Detection

The rate vs. detection accuracy performance of the scalable model is shown in Fig. 7. We show results for two scalable models: one with 4 layers and one with 14 layers as described in Section IV-B. The 14-layer model was obtained from the 4-layer model through layer splitting and further refined through training all 14 layers. It achieved performance similar to the 4-layer model but includes more points at the lower bitrate range for more flexible adaptation to network bandwidth in low data rate communication scenarios. It is encouraging to see that the 14-layer scalable model achieved similar accuracy as the non-scalable model at the high rate range, and had relatively small accuracy drop at the lower rates.

The 4-layer model achieved high detection performance at the high bitrate range, even slightly surpassing the performance of the non-scalable models. We suspect that this is because we have not performed an exhaustive search of all possible model configurations including the reduced channel number, down-sampling of selected channels, lambda value, and split point for the non-scalable model. Had we found optimal configuration for each rate point, the non-scalable model should achieve equal or higher detection accuracy than the scalable model at every rate.

Similar to the experiment for the non-scalable model, we also trained a model where the task encoder and decoder are fixed with the pre-trained YOLO model weights. In this case, having a fixed task encoder and decoder lowered the detection performance more severely than the non-scalable model. This suggests that for the scalable model, it is even more important to train the entire model jointly to achieve good performance.

When evaluated on the higher-resolution (1624×1200) TJU-DHD dataset, our model achieved similar performance curves (Fig. 8). Even though the model was originally trained on the COCO-Traffic dataset with resolution 640×640, switching to higher-resolution data did not significantly affect the performance.

We would like to emphasize that for the baseline non-scalable methods, different points on the curve require a different set of task encoder, task decoder, compression and decompression modules. On the other hand, our scalable compression approach achieves all the rate points with a single pair of task encoder and decoder and a fixed set of compression/decompression layers. Higher rates are simply achieved when more compression/decompression layers are invoked. This makes our scalable approach much more practical for real-world applications.

D. Performance Under Packet Drop

Because of the embedded structure of the scalable bitstream, the scalable bitstream is better protected against possible packet drop during transmission. We demonstrate this through an experiment where a certain percentage of packets are dropped, which reflects the scenario of packet drop due to congestion or bandwidth decrease in the network during transmission. We assume that each channel of the features is packetized as an individual packet for transmission. During inference, a certain percentage of the total packets are removed before passing to the task decoder.

We compare the performance of the scalable and the non-scalable model when different percentages of the packets were dropped. For the scalable bitstream, the highest layers will be dropped first while the channels closer to the base layer will be prioritized for delivery. For the non-scalable bitstream,

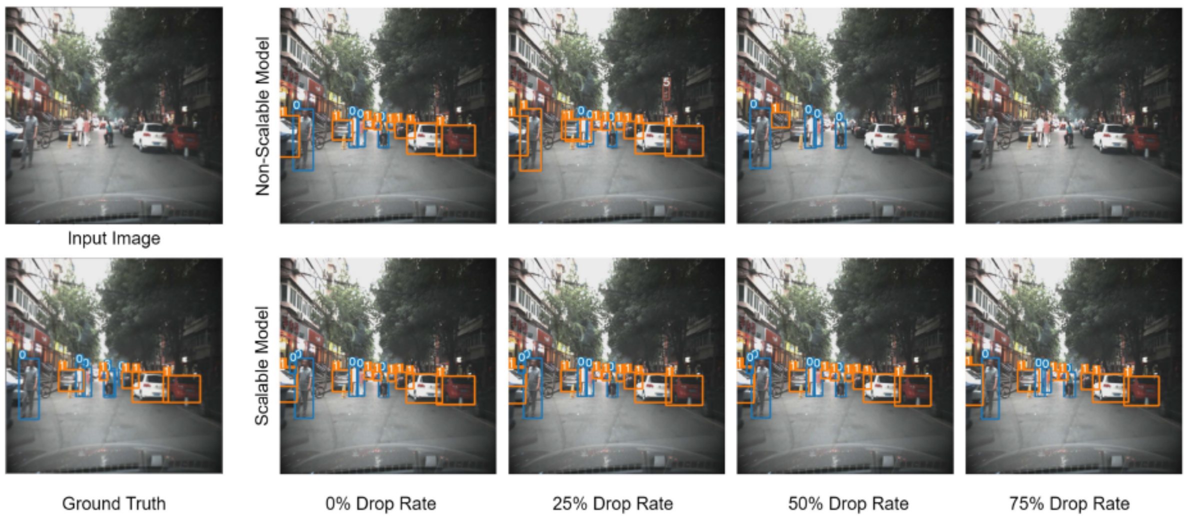


Fig. 10. Examples of detection results from the scalable and non-scalable model under different packet drop rates. Image samples taken from the TJU-DHD dataset.

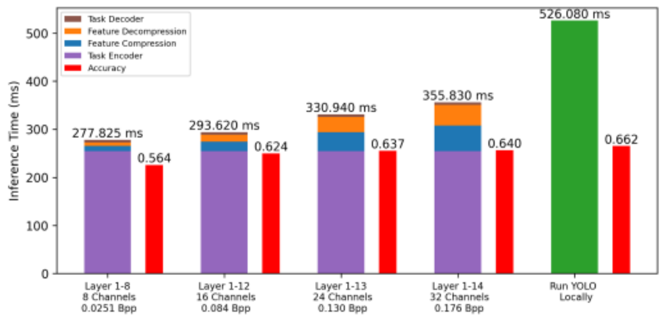


Fig. 11. Inference time of using different numbers of scalable layers.

because the channels have no particular order of importance, the channels are randomly dropped. Channels not received by the receiver will be filled with zeros before being passed to the task decoder model.

Fig. 9 shows the detection performance under different packet drop rates, evaluated on the COCO-Traffic dataset. The performance of the non-scalable model decreased rapidly as more packets were dropped. In comparison, the scalable model was able to maintain mAP higher than 0.6 with packet drop rate as high as 50%. This experiment demonstrates the ability of the scalable model to adapt to changing network bandwidth and maintain high detection performance when the channel bandwidth decreases. Examples of detection results at different packet drop rate are shown in Fig. 10. The scalable model continues to detect correctly while the non-scalable model fails to detect many objects when the packet drop rate increases to more than 50%. This demonstrates the benefit of the scalable model in scenarios such as vision-assisted navigation, where the mobile device is moving and the bandwidth may change rapidly as the environment changes.

E. Comparison With Principal Component Analysis

The proposed channel reduction module essentially performs a linear transform over the original intermediate features, in which the 1×1 convolution kernels are learned bases. To demonstrate the benefit of using learned bases, we experimented with using transform bases corresponding to Principal Component Analysis (PCA) (also known as Karhunen Loeve Transform). PCA provides transform bases that lead to uncorrelated channels and has the property that using a given number of coefficients with largest variances, the original features can be reconstructed with the lowest MSE.

With the pre-trained YOLO model without compression, we first generate intermediate feature channels for all images in the training set. We then perform PCA over all the sample vectors, each consisting of all the features corresponding to the same pixel, to obtain a set of transform bases and order them by the coefficient variances. The top 32 channels were used to construct a 4 layer scalable model, each layer having 8 channels. We then trained a hyperprior model for entropy coding of each layer while keeping the rest of the model fixed. As shown in Fig. 7, this PCA approach led to significantly lower rate-analytics performance. This is because PCA minimizes the MSE in reconstructing the intermediate features for a given number of reduced channels, but not the task loss. This result reinforces the importance of learning the transform bases using the rate-task loss for multiple scalable layers.

F. Complexity Scalability

In addition to bitrate scalability, the proposed model is also scalable in complexity. The scalable compression model compresses the features into a different number of channels depending on the target bitrate. The number of channels directly affects the runtime for the feature dimension reduction module and the arithmetic encoding and decoding.

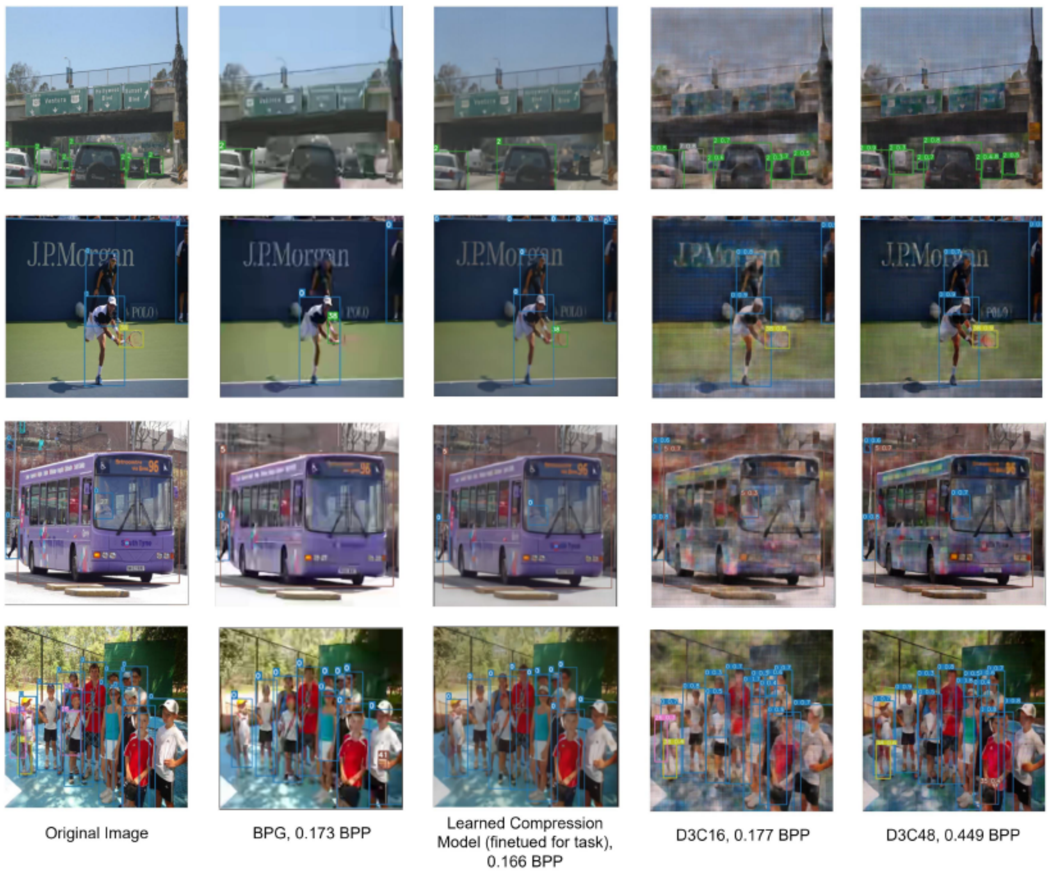


Fig. 12. Reconstructed images from compressed features from the D3C16 and D3C48 models, compared to decompressed images from BPG and the learned image compression model in [8].

Using the same setup that was used to measure the non-scalable model, we measured the inference time of the scalable model using different numbers of scalable layers. As shown in Fig. 11, a larger number of layers requires longer inference time, and the additional computation at the client leads to more battery energy consumption. In practical applications, the number of layers can be tuned based on the battery status of the client device and the end-to-end latency required for the application, in addition to the network throughput.

G. Image Reconstruction From Compressed Features

Fig. 12 shows reconstructed images from compressed features at a bitrate of 0.177 bpp and 0.449 bpp, respectively, along with images compressed by other models at a bitrate close to 0.177 bpp. From the images reconstructed from the task features, we can clearly recognize the detected objects.

At similar low bitrates, images compressed by BPG or the learned image compression model have more severe blurring and compression artifacts that can affect the object detection performance. In comparison, despite the blurring of the background and sometimes severe distortion in the color of the detected object (e.g. the purple bus) and irrelevant details (e.g. the text in the background and the bus), the shape and edges of the objects in our reconstructed image are more defined, which may have led to the better detection performance. For example in the top row, several cars are missed in the compressed images by BPG and the learned image coder, while they are successfully detected by the proposed scheme at similar low bit rates. In the second row, the tennis racket held by the person is completely blurred out with BPG compression, and as a result, is not detected by YOLO. With the learned image coder, the racket was detected as another object. In the reconstructed image from our models, the racket is more visible and correctly detected.

H. Limitations and Future Works

In this paper, we have not considered the latency due to the transmission of packets from mobile to server. We assume that this latency will be small (typically around 15ms [32]) because the server is close to the mobile device in edge computing. In addition, we designed our model around image-based computer vision models that do not consider temporal correlation between frames. Models with the ability to detect and track objects based on a series of frames reduce the need to re-detect per image. How to compress the features in models with temporal feature extraction is an important topic for further research. In practice, the total latency of the model may also decrease the accuracy of the prediction, because objects captured in the frame may have moved by the time results are sent back to the mobile device. A solution that addresses all of these problems is a promising direction for future research.

VII. Conclusion

This paper considered offloading deep-learning based visual analytics tasks by splitting the computation between the mobile device and the edge server. We introduced a lightweight trainable feature compression architecture, which includes feature channel and spatial reduction, and hyperprior-based entropy coding. With end-to-end training of the feature compression, decompression, and the task model using a rate-task loss, our approach can achieve higher task accuracy at low to medium rate range than baseline methods that perform image compression at the mobile device and image decompression and object detection or image classification on the server. Furthermore, our approach has significantly lower run-time at the mobile device (with CPU only) and consequently lower total inference time than the baseline methods.

We also proposed a scalable feature compression approach that can vary the number of bit stream layers and consequently the bit rate based on the sustainable throughput of the channel, desired in split computing through a volatile wireless link between a mobile and the server. Compared to the non-scalable model, our scalable model achieved comparable performance in rate-analytics trade-off with only a single task model. Although we only demonstrated the performance of the scalable compression approach for object detection, we expect similar trend for image classification.

References

[1] L. D. Chamain, F. Racapé, J. Bégaint, A. Pushparaja, and S. Feltman, "End-to-end optimized image compression for machines, a study," in *Proc. IEEE Data Compression Conf.*, 2021, pp. 163–172.

[2] N. Le, H. Zhang, F. Cricri, R. Ghaznavi-Youvalari, and E. Rahtu, "Image coding for machines: An end-to-end learned approach," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2021, pp. 1590–1594.

[3] I. V. Bajić, W. Lin, and Y. Tian, "Collaborative intelligence: Challenges and opportunities," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2021, pp. 8493–8497.

[4] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

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

[6] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.

[7] G. J. Sullivan et al., "Standardized extensions of high efficiency video coding (HEVC)," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 1001–1016, Dec. 2013.

[8] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rkcQFMZRb

[9] Z. Yuan, S. Rawlekar, S. Garg, E. Erkip, and Y. Wang, "Feature compression for rate constrained object detection on the edge," in *Proc. IEEE 5th Int. Conf. Multimedia Inf. Process. Retrieval*, 2022, pp. 1–6.

[10] Z. Yuan, S. Garg, E. Erkip, and Y. Wang, "Scalable feature compression for edge-assisted object detection over time-varying networks," in *Proc. Workshop Resource-Constrained Learn. Wireless Netw.*, 2023. [Online]. Available: https://openreview.net/forum?id=b9lQw75UugS

[11] L. Duan, J. Liu, W. Yang, T. Huang, and W. Gao, "Video coding for machines: A paradigm of collaborative compression and intelligent analytics," *IEEE Trans. Image Process.*, vol. 29, pp. 8680–8695, 2020.

[12] W. Gao et al., "Recent standard development activities on video coding for machines," 2021, *arXiv:2105.12653*.

[13] J. Choi and B. Han, "Task-aware quantization network for JPEG image compression," in *Proc. 16th Eur. Conf. Comput. Vis.*, 2020, pp. 309–324.

[14] Z. Chen et al., "Toward intelligent sensing: Intermediate deep feature compression," *IEEE Trans. Image Process.*, vol. 29, pp. 2230–2243, 2019.

[15] A. E. Eshratifar, A. Esmaili, and M. Pedram, "BottleNet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2019, pp. 1–6.

[16] S. Singh et al., "End-to-end learning of compressible features," in *Proc. IEEE Int. Conf. Image Process.*, 2020, pp. 3349–3353.

[17] 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 *Proc. IEEE 26th Int. Conf. Pattern Recognit.*, 2022, pp. 182–188.

[18] Z. Wang, F. Li, Y. Zhang, and Y. Zhang, "Low-rate feature compression for collaborative intelligence: Reducing redundancy in spatial and statistical levels," *IEEE Trans. Multimedia*, vol. 26, pp. 2756–2771, 2024.

[19] Y. Kim, S.-Y. Jeong, J. Lee, J. Lee, and M. Kim, "Pixel-unshuffled multi-level feature map compression for FCVCM," in *Proc. IEEE Int. Conf. Vis. Commun. Image Process.*, 2023, pp. 1–4.

[20] Y.-U. Yoon, G.-W. Han, J. Lee, S. Y. Jeong, and J.-G. Kim, "An advanced multi-scale feature compression using selective learning strategy for video coding for machines," in *Proc. IEEE Int. Conf. Vis. Commun. Image Process.*, 2023, pp. 1–5.

[21] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Process. Commun.*, vol. 1, Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[22] C. Jia, Z. Liu, Y. Wang, S. Ma, and W. Gao, "Layered image compression using scalable auto-encoder," in *Proc. IEEE Conf. Multimedia Inf. Process. Retrieval*, 2019, pp. 431–436.

[23] Y. Mei, L. Li, Z. Li, and F. Li, "Learning-based scalable image compression with latent-feature reuse and prediction," *IEEE Trans. Multimedia*, vol. 24, pp. 4143–4157, 2021.

[24] Y. Ma, Y. Zhai, and R. Wang, "DeepFGS: Fine-grained scalable coding for learned image compression," 2022, *arXiv:2201.01173*.

[25] N. Yan, D. Liu, H. Li, and F. Wu, "Semantically scalable image coding with compression of feature maps," in *Proc. IEEE Int. Conf. Image Process.*, 2020, pp. 3114–3118.

[26] H. Choi and I. V. Bajić, "Scalable image coding for humans and machines," *IEEE Trans. Image Process.*, vol. 31, pp. 2739–2754, 2022.

[27] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rJxdQ3jeg

[28] *Ultralytics*, "YOLOv5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[31] F. Bellard, "BGP image format," 2018. [Online]. Available: https://bellard.org/bpg/

[32] Z. Yuan et al., "Network-aware 5G edge computing for object detection: Augmenting wearables to 'see' more, farther and faster," *IEEE Access*, vol. 10, pp. 29612–29632, 2022.

[33] Y. Pang et al., "TJU-DHD: A diverse high-resolution dataset for object detection," *IEEE Trans. Image Process.*, vol. 30, pp. 207–219, 2020.

[34] *T. maintainers and contributors*, "TorchVision: PyTorch's computer vision library," Nov. 2016. [Online]. Available: https://github.com/pytorch/vision

[35] E. D. Cubuk et al., "AutoAugment: Learning augmentation strategies from data," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 113–123.