Visual Analytics Through Edge Servers: Learned Feature Compression and Adaptive Video Coding

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Electrical Engineering)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Zhongzheng Yuan

May 2024

Visual Analytics Through Edge Servers: Learned Feature Compression and Adaptive Video Coding

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Electrical Engineering)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Zhongzheng Yuan

May 2024

Approved:

Ivan Selesnick

Department Chair Signature

May 10, 2024

Date

University ID: N15925747

Net ID: zy740

Approved by the Guidance Committee:

Major: Electrical Engineering

Wang Yao 5/9/2024

Yao Wang
Professor
Electrical and Computer Engineering

Elza 5/9/2024

Elza Erkip
Professor of
Electrical and Computer Engineering

Siddharth

Siddharth Garg
Associate Professor of
Electrical and Computer Engineering

Microfilm or other copies of this dissertation are obtainable from

UMI Dissertation Publishing
ProQuest CSA
789 E. Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Vita

Zhongzheng (Jacky) Yuan was born in Guangzhou, China, in 1997. He received his B.S. and M.S. degrees in Electrical Engineering from New York University Tandon School of Engineering in 2019 and 2020, respectively. He started his Ph.D. study at New York University Tandon School of Engineering in Fall 2020. His research focus on image and video compression, computer vision and deep learning. During his Ph.D. years, he interned at Netflix, Amazon, Dolby Laboratories, and Tencent America, working on video compression related research.

Acknowledgements

First and foremost, I would like to express my gratitude to my advisor Prof. Yao Wang for her guidance and mentorship throughout my doctoral study and for encouraging me to pursue a Ph.D. degree while I was an undergraduate student. Through her mentorship, I have learned to become a better researcher and engineer. I would also like to thank the guidance committee members, Prof. Elza Erkip and Prof. Siddharth Garg for reviewing this work and providing invaluable advices and insights throughout the research.

I am fortunate to have worked with research collaborators Samyak Rawlekar, Tommy Azzino, Haoyang Pei, Yu Hao, Junchi Feng, Prof. John-Ross Rizzo, Prof. Marco Mezzavilla, and Prof. Sundeep Rangan, whose assistance and suggestions have improved my research. I would also like to thank all the NYU Video Lab members for giving me inspiration and new ideas from their research and the current research trends.

Last but not least, I would like to thank my parents for their care and endless support on this journey. I would also like to thank my friend Thomas for his encouragement and companionship.

Zhongzheng Yuan

May 2024

ABSTRACT

Visual Analytics Through Edge Servers: Learned Feature Compression and Adaptive Video Coding

by

Zhongzheng Yuan

Advisor: Prof. Yao Wang, Ph.D.

Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy (Electrical Engineering)

May 2024

Deep learning models for visual analytics require high computation capabilities to process high-resolution images in real-time. Offloading the computation to an 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. In this thesis, we explore using learned compression techniques to improve the performance of visual analytics tasks under bitrate and computation constraints. We first consider using learned autoencoder-based image compression model to compress the image and perform inference on the decompressed image. Using a rate-task loss in training the compression and inference pipeline end-to-end, we show that the learned end-to-end trained pipeline achieves improved task performance over compression models optimized for human visualization. Split-computing is another approach for computation offloading where the task model is split between the mobile device and the server. The intermediate features are compressed and transmitted between the two devices. We propose a learned feature compression method using hyperprior and feature dimensionality reduction to compress the intermediate features. 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 on the mobile, and running the analytics model on the decompressed images on the server, the proposed feature-compression approach achieves significantly higher analytics performance at the same bit rate, while reducing the complexity on 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. Finally, we explore using standard video compression codecs in real-time video object detection applications. A high-resolution video dataset of pedestrian street scenes was collected to investigate the effect of changing spatial and amplitude resolution on object detection performance. Based on this result, a resolution adaptive video compression approach was proposed to maximize the task performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Deep learning models have been successful in achieving state-of-the-art performance in various visual analytics tasks. The ability to 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, the 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 and therefore reduces the battery life of the device.

One potential solution to this problem is to employ edge computing, which transmits the image data captured in the device to a nearby edge server with high computation ability. A crucial component in this system is the compression of the image data before it is transmitted. Efficient computation offloading requires optimizing multiple objectives including compressed data rate, analytics performance, and overall inference speed. Our research is focused on the problem of data compression for offloading the computation of visual analytics. We aim to develop a method of compression that maximizes the rate-accuracy trade-off, while under the constraint of fast inference speed of the overall system.

1.2 Organization of the Thesis

This thesis is organized as follows:

In Chapter 2, we provide a brief review of learned image compression models and our work on fine-tuning image compression models for analytics.

In Chapter 3, we propose a split computing model with a learned feature compressor to compress the intermediate features. We compare the performance of the split computing approach to the image compression approach.

In Chapter 4, we propose a method of scalable feature compression as an extension to the split computing model. The scalable model offers variable bitrate compression and better adaptation to dynamic network environments than the non-scalable model.

In Chapter 5, we investigate the use of a standard video codec to compress a video stream in an edge computing system for blind and visually impaired navigation assistance. We propose a spatial resolution adaptive compression strategy to improve object detection performance.

Chapter 2

Image Compression for Analytics

2.1 Learned Image Compression

Image compression is a fundamental and widely studied research problem in image processing and computer vision. The goal of image compression is to reduce the bitrate necessary to store and transmit image, while minimizing the distortion caused by compressing the image. Most image compression models follow the transform-based image compression framework. It consists of three basic modules: transform function for converting the input image to a more energy compact transform domain, quantizer for discretizing the transform coefficients, and entropy coding to losslesssly compress the quantized coefficients. The popular image compression standard JPEG is one example of this approach which uses the discrete cosine transform (DCT), a transform that was found to have good energy compactness and decorrelation to the image [50].

More recently, learned image compression approaches have been proposed and achieved remarkable success in improving rate-distortion performance. Deep learning based models benefit from the strong modeling capacity of deep neural networks and end-to-end optimized pipeline to achieve better rate-distortion results than traditional image compression codecs. The variational autoencoder model proposed by Ballé et. al. [2] is an influential work that proposed end-to-end training and the use of convolutional neural network as the encoder and decoder model. The encoder model is a fully convolutional network that transforms the input image into a latent tensor $\boldsymbol{z}$, with reduced spatial dimension and increased channel dimension. The latent tensor is then quantized and entropy coded with a piecewise linear probability model. At the decoder, the decoded quantized latent is transformed back to the image domain using a decoder model with transposed convolution.

This model is later extended by Ballé et. al. to use a hyperprior model to estimate the mean and variance of the Gaussian distribution used for encoding the quantized features [3]. A hyperprior encoder is trained to extract a set of features $\boldsymbol{z}_{hyp}$ from the main features $\boldsymbol{z}$. The quantized $\hat{\boldsymbol{z}}_{hyp}$ is also entropy encoded and sent as side information to the decoder. The hyperprior decoder decodes $\boldsymbol{z}_{hyp}$ to obtain the mean and variance parameters for entropy encoding and decoding.

The encoder and decoder networks are end-to-end optimized using a rate-distortion loss:

$$\begin{aligned} L &= R + \lambda \cdot D \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_x}[-\log_2 p_{\hat{\boldsymbol{z}}|\hat{\boldsymbol{z}}_{hyp}}(\hat{\boldsymbol{z}}|\hat{\boldsymbol{z}}_{hyp}) - \log_2 p_{\hat{\boldsymbol{z}}_{hyp}}(\hat{\boldsymbol{z}}_{hyp})] + \lambda \cdot \mathbb{E}_{\boldsymbol{x}\sim p_x}[d(\boldsymbol{x}, \tilde{\boldsymbol{x}})], \end{aligned} \tag{2.1}$$

where λ is a Lagrangian multiplier that controls the rate-distortion trade-off, $\hat{\boldsymbol{z}}$ is the quantized latent, $\hat{\boldsymbol{z}}_{hyp}$ is the quantized hyperprior latent, and $d(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ is the distortion between the original image $\boldsymbol{x}$ and the decoded image $\hat{\boldsymbol{x}}$. For image compressors trained for human visualization, the distortion function is commonly chosen to be MSE or SSIM to preserve the perceptual quality of the compressed images.

2.2 Finetuned Image Compression for Visual Analytics

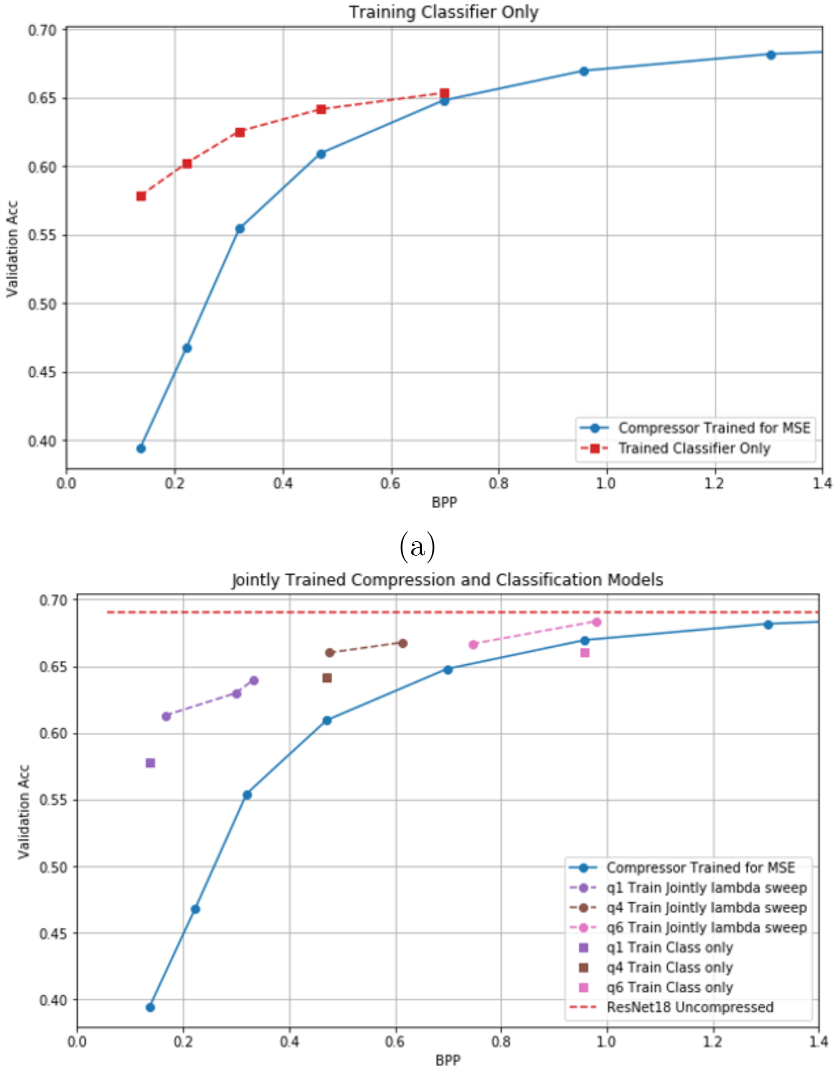
Using a compression model trained for visual quality to compress images for visual analytics will lead to a decrease in task accuracy. As shown in Figure 2.1, the accuracy of the classificaiton model decreased significantly when the image was compressed by a compressor trained with MSE as the distortion function. This result suggests that the compressor trained for MSE distortion was not optimized to keep information in the image that are important for the analytics model.

We proposed to finetune both the image compression model and the analytics model to improve the accuracy under compression. This was achieved by training the compressor-task model pipeline with a rate-distortion loss that takes into account the task performance. As an example, for a classification task, we replaced the distortion loss with cross-entropy (CE) loss between the classification results based on compressed images and the ground-truth labels y.

$$L = CrossEntropy(g(f(\boldsymbol{x})), y) + \lambda \cdot \mathbb{E}_{\boldsymbol{x} \sim p_x}[-\log_2 p_{\hat{\boldsymbol{z}}}(\hat{\boldsymbol{z}})], \tag{2.2}$$

where $\boldsymbol{x}$ is the raw image, $\hat{z}$ is the quantized latent, f and g are the compressor and the classifier, and λ is a scalar that controls the RD trade-off.

We performed our experiment with image classification as the task, and the ImageNet dataset for training and evaluation [13]. The joint-autoregressive and



(a)

(b)

Figure 2.1: Rate-Accuracy curves of compressor-classifier model: (a) training the classifier only, (b) jointly training both the compressor and the classifier

hierarchical priors network [35] was used as the compression model. The ResNet18 classification models was used as the classifier, and the models was initialized with pre-trained weights from `torchvision` in the Pytorch library [40].

We trained the classifier and compressor models in two steps to optimize both models for compressed images. First, we fixed the weights of the compressor model and trained the classification model only. The compressor was initialized using pre-trained model in the `CompressAI` library [4], trained using Rate-MSE loss to different bit-rates (identified by the quality number $q = 1, 2, ..., 8$). The λ parameter in the loss was set to 0 to minimize only the cross-entropy term. This step trained the classifier to learn the characteristics of compressed images and adapt to the compression artifacts in the images. The accuracy of the trained classifiers are shown in Figure 2.1a. For all rate-points, training the classifier model led to an increase in the classification accuracy over the initial model. The increase is especially significant for the low bitrate models.

In the second step, we jointly-trained both the compressor and classifier networks. Initializing the classifier models from the trained models in the previous train step, we continued the training by optimizing both compressor and classifier for the combined rate-CE loss in Eq. 2.2. For each compressor-classifier pair at different q, different values for the λ hyper-parameter were used to train models that are near each initial rate point q. As shown in Figure 2.1b, the jointly-trained models led to further increase in performance compared to training the classifier only.

Chapter 3

Feature Compression and Split Computing

3.1 Introduction

In Chapter 2, we have shown that jointly training the learned image compression model and the task model is effective in achieving high rate-task performance. However, this framework has a few drawbacks when used in practical applications. The learned image encoder is a large model that consists of convolution layers with large number of channels. Such model is computationally expensive for the client device which typically has no GPU processor for convolutional network acceleration. Furthermore, when the compressed features are sent to the server, the image decoder has to fully decode the image back to the original image dimension before feeding it to the task network for processing, adding an additional amount of computation that reduces the overall inference speed.

split computing, also known as 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 [1]. CI for deep neural networks was first proposed by [22], 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 features 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 [8, 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 rather than task model features.

Feature compression with learned entropy compression model was proposed in [44]. The hyperprior model was used to predict feature probability model and the compression and task model was end-to-end trained. However, the split point considered by [44] 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 [12], 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 [52], 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 Feature Coding for Video Coding for Machines (FCVCM) standardization effort calling for proposals of feature compression solutions. In [23], the intermediate features were stacked in a matrix and Principal Component Analysis (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 [54], except that a trained neural network was used instead of the PCA to transform the intermediate features. The two 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.

3.2 Split Computing for Visual Analytics Models

We propose a split computing model with learnable feature compressor and decompressor. The task model is split into two parts, with the first part running on the mobile device and the second part running on the server. We call 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. An example of the split computing network is shown in Figure 3.1, where the YOLOv5 model is modified for split computing.

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 located deeper into the task model outputs 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 [2, 3], 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 [2, 3], the input image undergoes four 2× down-sample convolution layers before being entropy encoded. Interestingly, several task models [18, 48], including the ones we use, also contain strided convolutions or 2× 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 [2, 3]. We also consider splitting after the third down-sampling layer (D3) for less computation on the 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 [48] for object detection, and the ResNet18 model for image classification [18].

3.3 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,

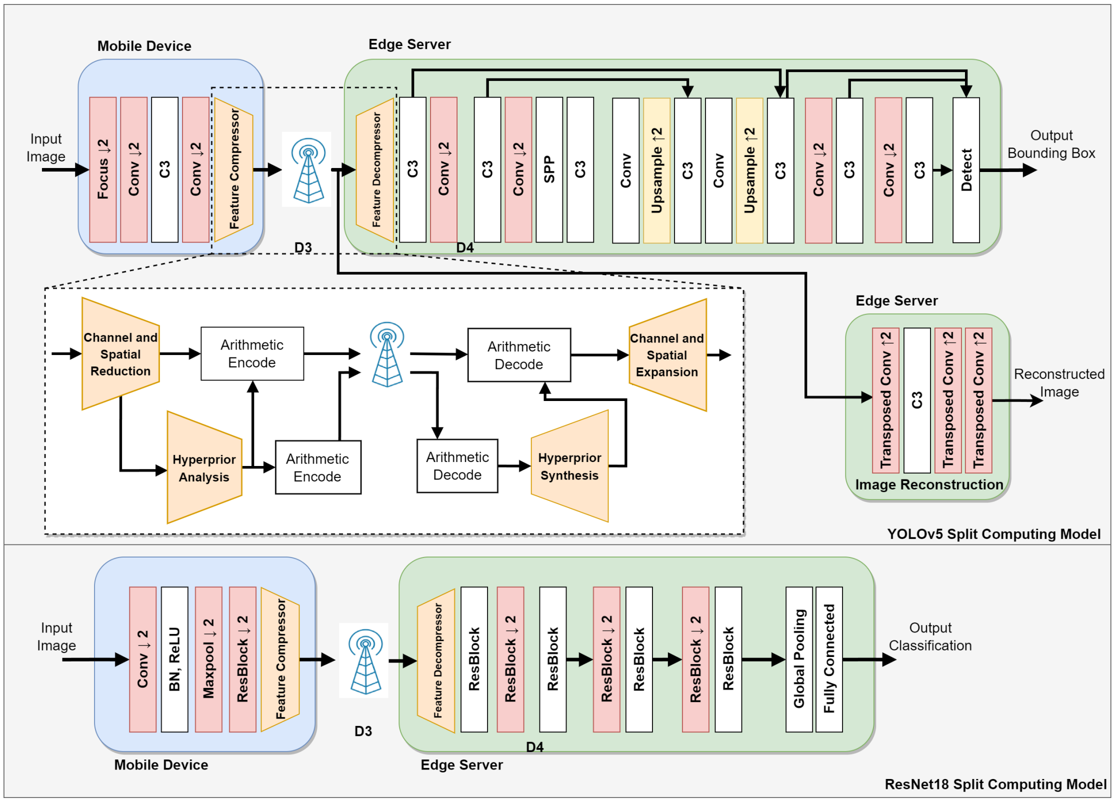


Figure 3.1: Overview of the proposed system. The task model is split into two parts that run on the mobile device and the edge server, respectively. A feature compression model is used to compress the intermediate features at D3. Another split point (D4) considered in our experiments is also indicated. 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.

which can be seen in the inter-channel covariance matrix, as shown in Fig. 3.2 (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. 3.2 (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. 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 4.7. Reducing the dimension of the feature tensor helps increase the overall inference speed.

3.4 End-to-end Split Computing Model Training

In our framework, y is the feature generated at the split point of the YOLO network, which is further reduced to z by the channel reduction layer. We use θ to denote YOLO model parameters, and use ϕ to indicate the channel reduction

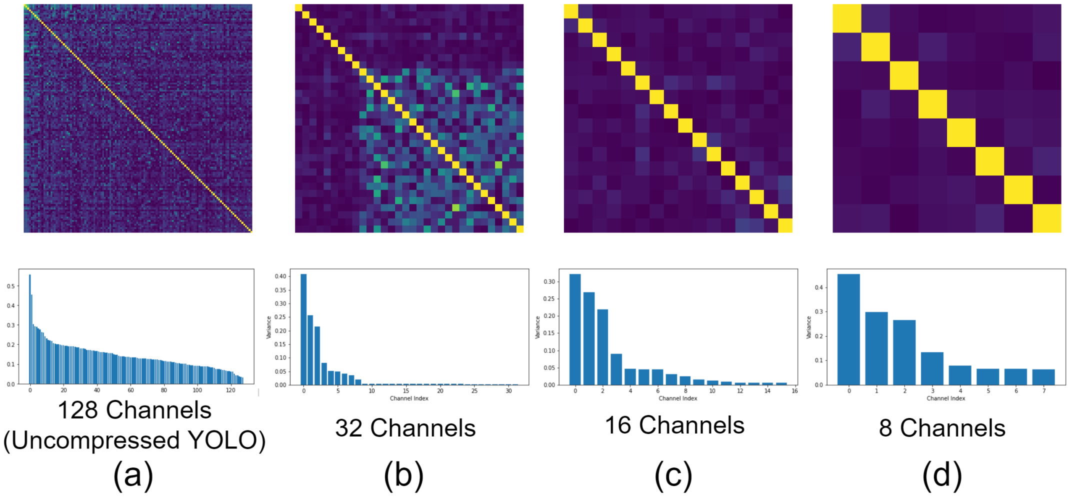


Figure 3.2: Inter-channel covariance matrix (top row) and the variance of each channel (bottom row) for the intermediate features from the YOLOv5 model at the D3 split point, for the original feature channels and after channel reduction to different number of channels.

and expansion layer parameters. Thus, we write $z(y(x;\theta);\phi)$ to indicate these dependencies. We modify the rate-distortion loss in Eq. (2.1) to perform end-to-end training of the entire system including the YOLO detection model (parameterized by θ) and the feature compressor and decompressor (parameterized by ϕ) inserted at the split point, for detection-aware compression. Instead, we replace the distortion loss by a detection loss L_{det} that directly measures the detection accuracy of the model's output:

$$\begin{aligned} L &= L_R + \lambda \cdot L_{det} \\ L_R &= \mathbb{E}_{x\sim p_x}[-\log_2 p(\hat{z}(y(x;\theta);\phi))] \\ L_{det} &= L_{obj} + L_{class} + L_{box}. \end{aligned} \tag{3.1}$$

L_{det} is the loss used for training the uncompressed YOLO model, and it consists of the object detection loss L_{obj}, object class loss L_{class}, and bounding box loss L_{box}, which will depend on $\hat{y}(\hat{z};\phi)$ and θ. A combination of rate and detection loss allows us to perform end-to-end training of the entire model including both the compression and detection components.

Instead of directly performing entropy coding on the quantized version of the reduced feature $\hat{z}$, we follow the hyperprior idea proposed in [3] to generate the hyperprior feature that helps the entropy coding of $\hat{z}$. As shown in Fig 3.1, the hyperprior encoder generates quantized hyperprior feature $\hat{z}_h$, from $\hat{z}$. The hyperprior decoder predicts the mean and variance of each element in $\hat{z}$, in the Gaussian model used for entropy coding of $\hat{z}$. The additional rate for $\hat{z}_h$ is included in the rate loss L_R.

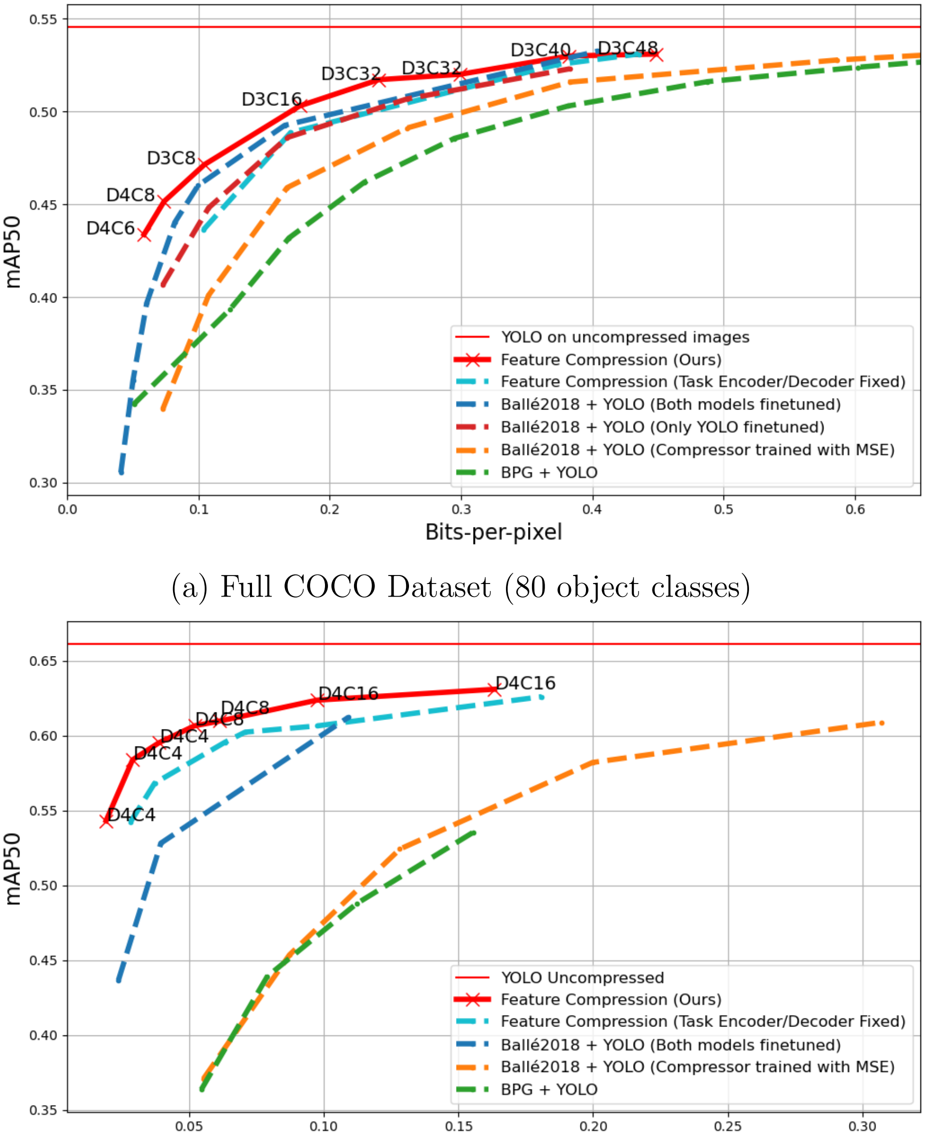
When inserting the dimension reduction layers into the model, the randomly initialized weights of the layers severely degrade the performance of the model. Through our experiments, we found it to be beneficial to first perform a pre-training step on the feature reduction and expansion layers, using the MSE loss between y and $\hat{y}$. This step does not invoke quantization (through adding random noise) on the reduced feature z, nor the hyperprior encoder and decoder and the rest of the network. Pre-training allows the feature reduction and expansion layers to learn to reconstruct the features as much as possible before training with the detection and compression objectives, which are more difficult to optimize than the MSE loss.

3.5 Experiments and Results

The Ultralytics YOLOv5 model [48] 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 [48], 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 640×640 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 framework. 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 [5], followed by the pretrained task model without fine-tuning. In the second baseline, the learned image compression model by [3] 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 Equation 3.1.

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 [55].



(a) Full COCO Dataset (80 object classes)

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

Figure 3.3: 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 rate vs. detection accuracy curves of our feature compression model and the benchmark methods are shown in Figure 3.3 (a). The detection accuracy is measured by the mean Average Precision under the Intersection over Union threshold of 50 (mAP50). We compared our approach against four baselines that compress and decompress the image before feeding into the detection network.

Our method performed better than the baselines of image compression followed by YOLO detection over the entire rate range considered, although the difference is small at the higher rates. Compared to jointly training the image compression model [3] and the YOLO model for rate-constrained object detection, our method is still better at the lower rate regime. We expect that with more exhaustive search of the hyperparameters in our feature compression layer, our method can be on-par with this benchmark over the higher rate range.

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. 3.3 (b)). The results are significant for settings with low communication bandwidth and require compression into extremely low bitrates. It is possible to deploy a split detector in these settings to have high detection performance while using low bandwidth transmission.

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 YOLOv5s model 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 [12], in which variable rate compression is achieved by switching the compression layers while the task encoder and decoder share the same weights for all bitrates. As expected, the rate-analytics performance with this approach is significantly lower than the end-to-end trained models (see 3.3). 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 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. 3.3(b))

For our experiments on the classification task, we adopted the ResNet18 model implemented by the torchvision package [31], and we used weights pretrained using ImageNet as initialization for our training. To combat overfitting, we used the AutoAugment augmentation strategy as described in [11] for training with the ImageNet dataset. 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 Figure 3.4. Our split computing classification model again achieved better accuracy against the baselines across different bitrates.

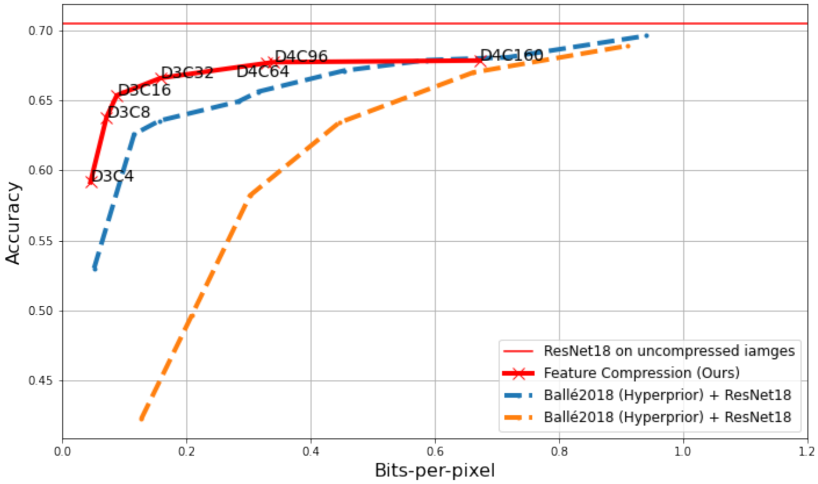


Figure 3.4: Classification Accuracy under various bitrates for the ImageNet dataset

3.6 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 1.1GHz 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 1.1GHz CPU for both the mobile and server.

For the object detection task, a breakdown of the inference time for our feature compression model and the baselines is shown in Table 3.1. 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

		[3] + YOLO (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

Table 3.1: Breakdown of runtime (milliseconds) per image (640 × 640 pixels) for the proposed split computing YOLO model and baselines. D4C6 and D3C40 refer to models with split point at D4 and D3 and channel reduction to $N_r = 6$ and $N_r = 40$, respectively. For each of the models D4C6 and D3C40, we compare the runtime with BPG compressed at a similar bitrate. The runtime for [3] + YOLO is similar regardless of the bitrate.

		[3] + ResNet18 (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

Table 3.2: Breakdown of runtime (milliseconds) per image (360 × 360 pixels) for the proposed split computing ResNet18 and baselines. For each of the D3C4 and D4C64 models, we compare the runtime with BPG compressed at a similar bitrate. The runtime for [3] + ResNet18 is similar regardless of the bitrate.

running YOLO locally.

Using the same settings, we performed runtime analysis on the classification models. The results are shown in Table 3.2. 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.

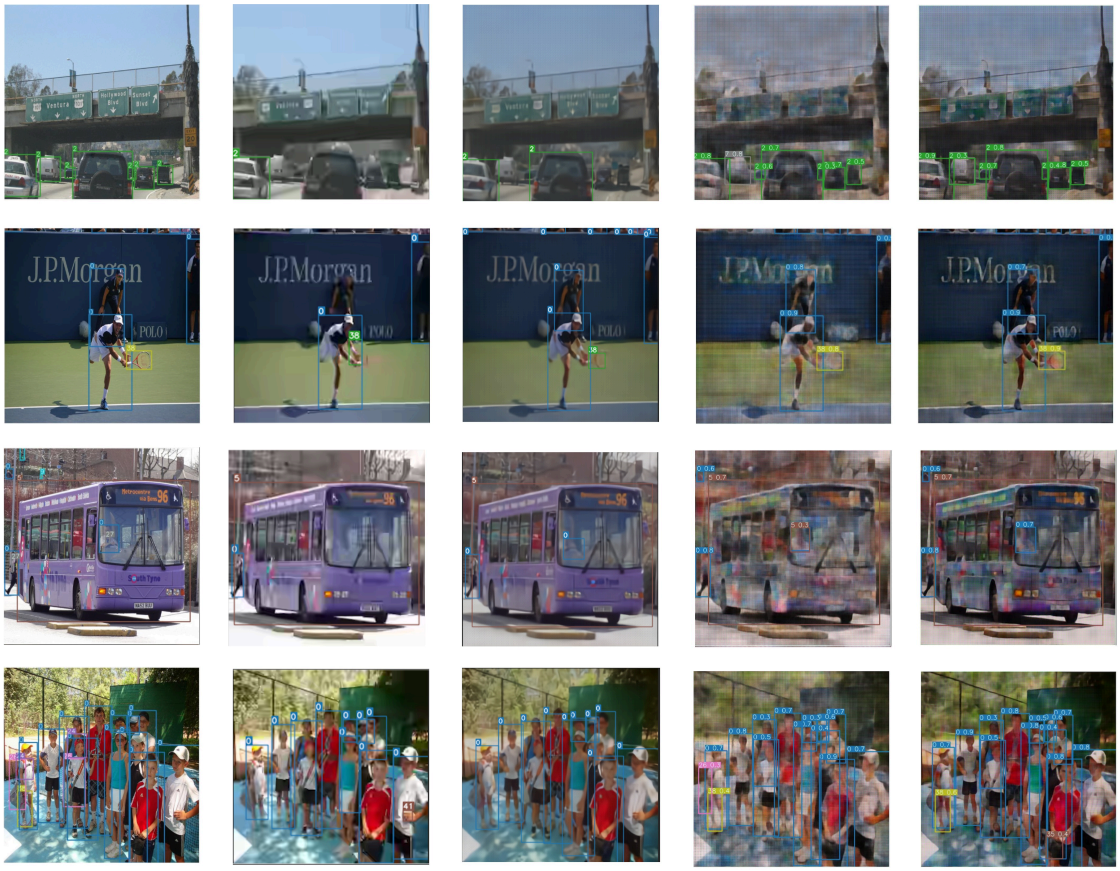
The amount of time saving in this experiment is less than that of YOLO because, compared to YOLO, ResNet18 is more computationally expensive in the early parts of the network. If a more complex classification model is needed for higher accuracy, for example ResNet50, we expect that there would be more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

3.7 Image Reconstruction From Compressed Features

In some applications, it may be necessary to not only have the task network output the detection results but to also have a human operator to verify these results. To demonstrate the possibility of achieving this functionality, we trained an image reconstruction network to reconstruct the image from the compressed features.

The reconstruction network follows the same architecture as the task encoder part of the split computing network, with the number of input and output channels reverted for each layers, shown in Figure 3.1. The stride-2 convolution layers are replaced with stride-2 transposed convolution layers. The loss used for training the reconstruction model is a distortion metric between the original input image and the reconstructed output image. We tested both MSE and MS-SSIM as the distortion metric in our experiments, but we found that both led to similar reconstruction results. Only the weights of the image reconstruction model are updated during training, while the weights of the split computing model, including the task network and the feature compression network, are frozen. Because the split computing model is not changed, training the reconstruction model does not affect the rate-task performance.

Figure 3.5 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 not detected in the compressed images by BPG and the learned image coder, while they are successfully detected by the proposed scheme at similar low bitrates. 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.



Original Image

BPG, 0.173 BPP

Learned Compression Model (finetued for task), 0.166 BPP

D3C16, 0.177 BPP

D3C48, 0.449 BPP

Figure 3.5: Reconstructed images from compressed features from the D3C16 and D3C48 models, compared to decompressed images from BPG and the learned image compression model in [3].

Chapter 4

Scalable Feature Compression

4.1 Introduction

In the previous chapters, we have explored image or feature compression trained for a single target bitrate. While training a different split computing model for each target bitrate is likely to achieve the highest analytical performance for each bitrate, it may not be practically feasible to store and operate 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 [43, 45], 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 bitrate.

4.2 Scalable Feature Compression for Analytics

Scalable image compression 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 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.

In a dynamic network with fast varying bandwidth, scalable compression has the benefit that only a single bitstream needs to be generated to adapt to varying network conditions. The sender can choose to send the number of layers that best suits the current estimated network condition. Whereas in non-scalable encoding, the encoder must encode the bitstream at a fixed bitrate based on the bandwidth estimate prior to encoding, which may be over or under the capacity of the network when the packet is actually sent.

In the case of feature compression for analytics, scalability refers to generate a bitstream with multiple layers, so 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 connection. In such scenarios, scalable compression can be particularly useful. In the case of a sudden drop in the network throughput (e.g., switching from 5G to 4G 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 [21], scalability is achieved by using multiple encoder networks to successively compress the residual of the reconstructed image and send the residual latent information in layers. Another model proposed by[34] 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 [29]. 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.

Several prior works have considered scalable compression for analytics, such as [53], and [9]. But the scalability proposed by these works refers to the ability of the server network to perform additional analytics tasks as it receives each additional bitstream, while our proposed scalable model aims at increasing the accuracy of a single task with each additional layer. To the best of our knowledge, this work is the first to propose a scalable feature compression model for analytics task.

4.3 Scalable Feature Compression Model

We extend the non-scalable feature compression model in Chapter 3 to enable scalability in feature compression. We draw inspiration from principal component analysis (PCA) to perform dimensionality reduction along the channel dimension through linear transform. PCA has the property that each channel in the transform coefficients are uncorrelated, and that lowest MSE reconstruction can be achieved using the least number of channels, making it well-suited for scalable compression. But whereas the transform in PCA minimizes the MSE of the reconstructed signal, our model is trained end-to-end to minimize the rate-detection loss.

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 network 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:

$$\hat{y}_l = \sum_{m=1}^{l} \mathcal{E}_m(\hat{z}_m), l \in \{1, 2, ...M\}. \tag{4.1}$$

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. 4.1.

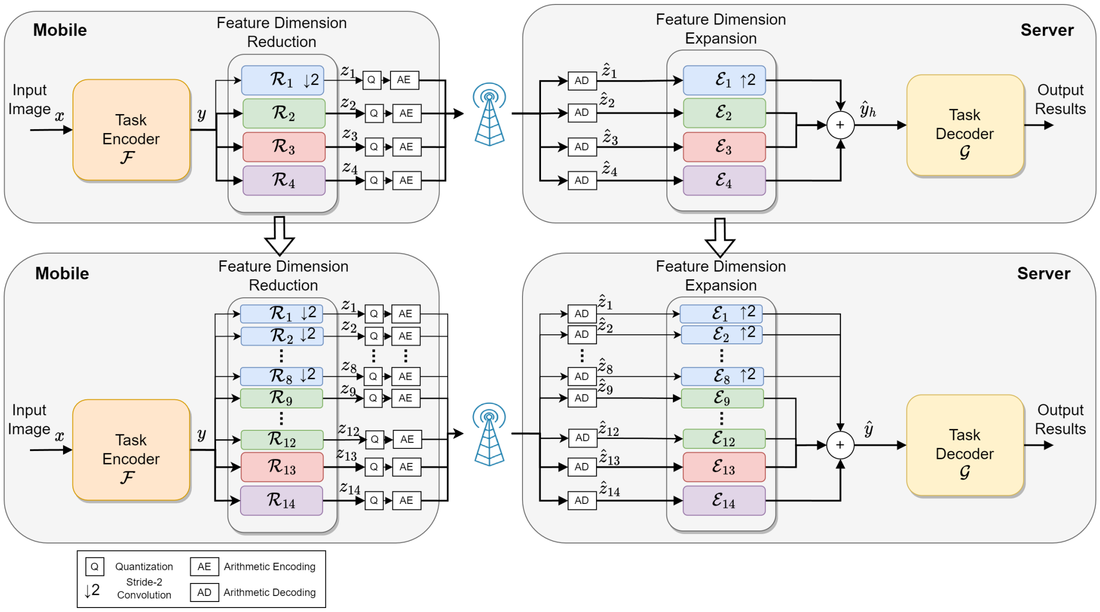


Figure 4.1: Overview of the proposed scalable compression model. The model creates scalable layers by generating and entropy coding different groups of reduced feature channels from the task encoder. In this example, we first train a model with four layers with each layer having 8 channels. We then split 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.

4.4 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:

$$L_l = \sum_{m=1}^{l} L^m_{Rate} + \lambda_l \cdot L^l_{Task}, \tag{4.2}$$

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 reported the best-performing model.

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.

4.4.1 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, shown in Figure 4.1. 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 the new convolution layers created from the split can be operated independently.

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}, ..., $N'_{R_{M'}}$ 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 Eq. (4.2), for both the newly splitted layers and the un-split layers.

4.5 Experiment and Results

We trained and evaluated our scalable model on the COCO-Traffic dataset. To further evaluate the model's performance on high resolution image, we also evaluated the model trained on the COCO-Traffic dataset on the TJU-DHD dataset [39], which is a high-resolution object detection dataset for traffic scenes.

We compare our results to two non-scalable baselines. The first one is the non-scalable split computing approach proposed in Section 3.4. With the second baseline, the mobile compresses an image using the learned image compression model by [3], then the server runs the object detection model on the decompressed image. With both baselines, different task modules and the compression/decompression modules are trained using the rate-task loss for different rate points.

The rate vs. detection accuracy performance of the scalable model is shown in Fig. 4.2. We show results for two scalable models: one with 4 layers and one with 14 layers as described in Section 4.4.1. 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

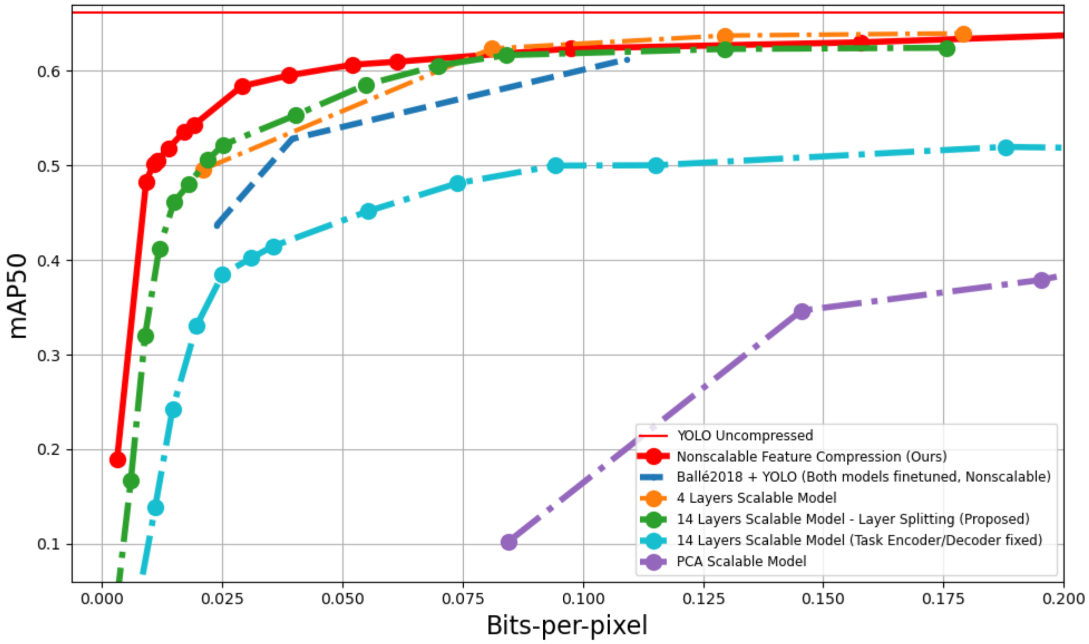


Figure 4.2: Rate-Accuracy performance for the proposed scalable model compared to non-scalable compression baselines, for the COCO-Traffic dataset

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

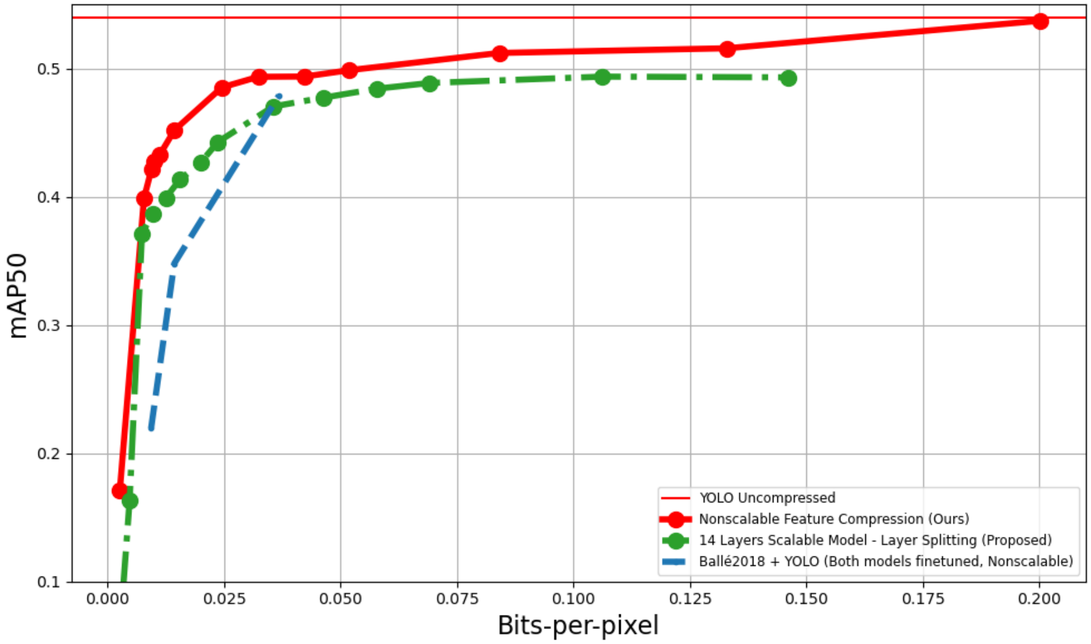


Figure 4.3: Rate-Accuracy performance for the proposed scalable model compared to non-scalable compression baselines. Trained on the COCO-Traffic dataset, evaluated on the TJU-DHD dataset.

model achieved similar performance curves (Fig. 4.3). 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.

4.6 Comparison with Principal Component Analysis

We performed an experiment to support the necessity of training the model end-to-end. Another method of achieving scalable compression is to perform principal component analysis (PCA) to perform dimensionality reduction along the channel dimension, and then order the channels from highest to lowest variance to form different layers. This method ensures that each channel in the feature are uncorrelated, while the reconstructed feature is the closest to the original in terms of MSE using the least number of channels.

We initialized the task encoder and decoder with weights from the scalable model trained with steps in Section 4.4. We performed PCA on the output features of the encoder model. We ordered the channels from highest to lowest variance. The four channels with the highest variance forms the base layer, the four channels with the next highest variance forms the first enhance layer, and so on. Since PCA reconstructs the original features, whereas the decoder trained in Section 4.4 was not trained to take the reconstructed features with compression artifacts from the decoder, directly using the PCA would not lead to good results. Therefore, we additionally train a transform layer after the inverse PCA transform to transform the decoded features to a form that the decoder expects.

As results shown in Figure 4.4 suggests, using PCA directly on the output of the encoder will lead to significantly lower performance. Training an additional transform layer increased the performance, but not enough to be comparable with the baseline trained by the scalable training procedure, even though the same task encoder and decoder are used. This suggests that in addition to decorrelation

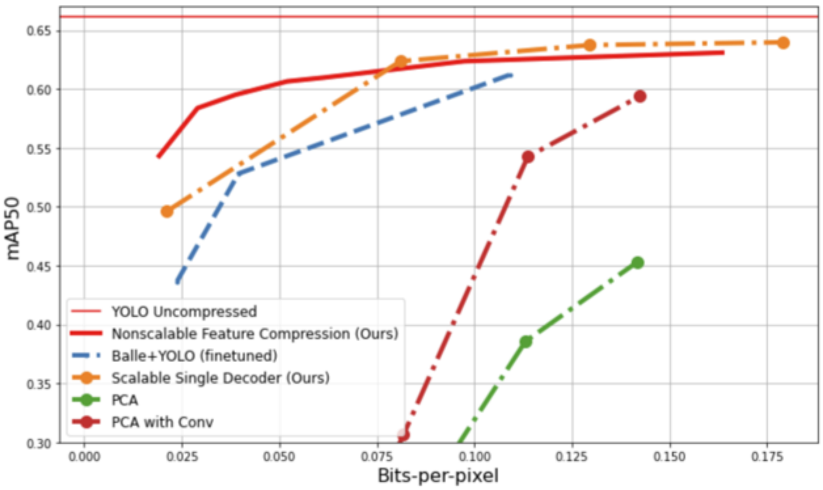


Figure 4.4: Detection accuracy vs. Bitrate of PCA scalable compression models

between the channels, is also important to train the model end-to-end to have a high performing scalable model,

4.7 Complexity Scalability

In addition to rate and performance scalability, the design of our scalable model also introduces complexity scalability. The scalable compression model compresses the features into different number of channels depending on the target bitrate. The number of channels directly affects the runtime for feature dimension reduction and expansion module, and arithmetic encoding and decoding. Since more computation is required to generate and compress the features as more layers are being sent, the computation complexity scales with the number of layers.

To simulate the hardware runtime of split computing pipeline, we used a 1.1GHz CPU as the mobile device to run the task encoder and the RTX8000 GPU on the NYU High Performance Computing server to run the task decoder. The entropy coding time was measured using a 1.1GHz CPU processor. We measured the inference time of the scalable model using different numbers of scalable layers. The results are shown in Figure 4.5. The computation increases as more layers are generated and send, as expected. However, the complexity at the mobile side is dominated by the task encoder. Because the client device is often also constrained by battery capacity, the split computing strategy must also consider computation complexity and the power consumption when selecting the number of layers to send.

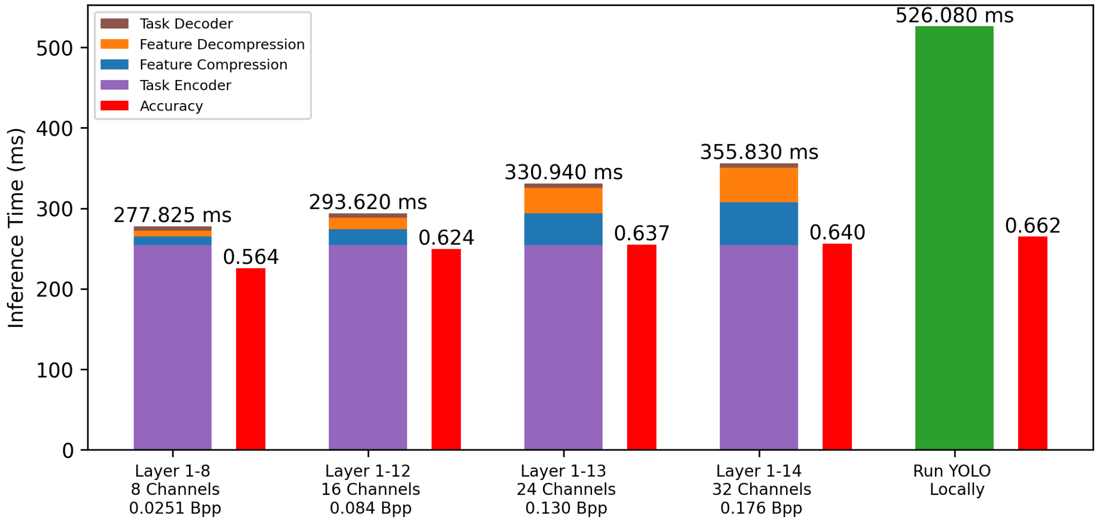


Figure 4.5: Inference time of using different numbers of scalable layers

4.7.1 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 simulates 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 assumed as dropped 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, because the channels have no particular order of importance, the channels are randomly dropped. For both models, channels that are not received by the receiver will be filled with zeros before being passed to the feature decompressor and task decoder model.

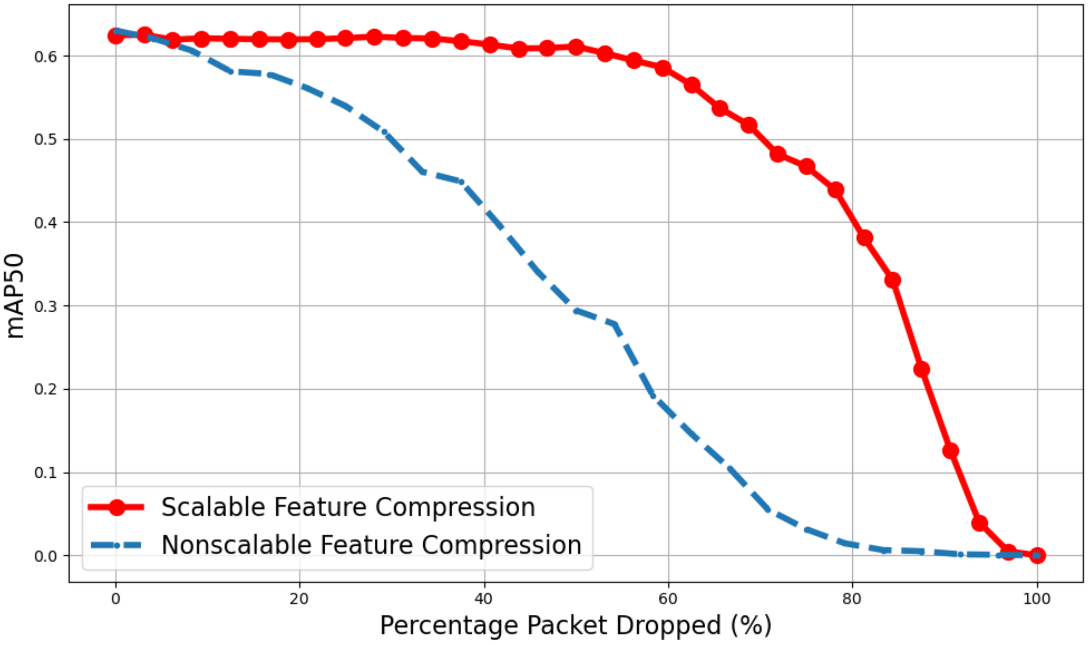


Figure 4.6: Detection accuracy of the proposed scalable model compared to the non-scalable model under different packet drop rate

Fig. 4.6 shows the detection performance under different packet drop rates, evaluated on the COCO-Traffic dataset. The performance of the non-scalable

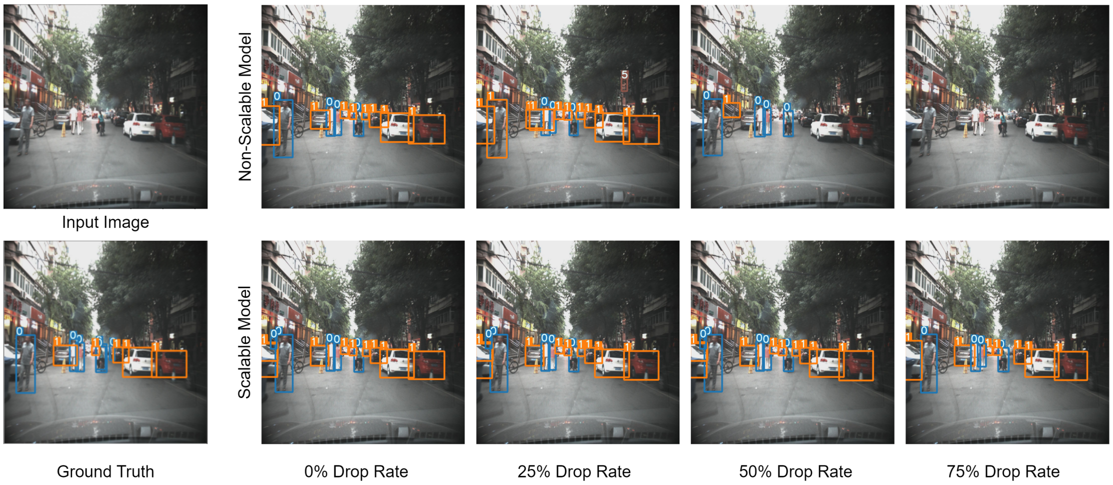


Figure 4.7: Examples of detection results from the scalable and non-scalable model under different packet drop rates. Image samples taken from the TJU-DHD dataset.

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. 4.7. 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 constantly moving and the bandwidth may change rapidly as the environment changes.

Chapter 5

Adaptive Video Compression for Real-time Analytics

5.1 Introduction

Technology in smart wearables is advancing rapidly with an increasing integration of high resolution cameras. High resolution video streams captured by wearable devices combined with analytics by computer vision models offers solution to many challenging problems such as providing navigation assistance for the blind-and-visually impaired (BVI) population. A key challenge of deploying advanced computer vision models in wearable settings is that state-of-the-art deep neural networks are computationally demanding, particularly for use cases that have stringent delay requirement.

The feature compression models developed in the previous chapters offer a promising solution to this problem. Although it offers many benefits over the image compression framework, much work in standardization and software development remains to be done for the widespread adoption of feature compression in present systems. On the other hand, video compression designed for human visual perception have long been studied, and standardization efforts have led to an ecosystem of software and hardware for video codecs that is easy to integrate into present edge computing systems. Furthermore, video compression utilizes inter-frame dependence to further reduce bitrate, whereas the proposed feature compression model only consider each frame as independent inputs.

To assess the possibility of transmitting compressed video stream through 5G network for object detection on the edge, we conducted a detailed evaluation of the requirements of achieving high performance detection in real-time. We consider the use of object detection in a smart wearable system for the BVI called the VIS^{4}ION system [6], [47]. The system is implemented as a backpack with camera mounted on the strap to capture video of the user's surrounding, and an Nvidia Jetson board contained within the backpack to perform local processing. The goal of the VIS^{4}ION system is to provide localization, navigation, and obstacle avoidance for the user in a dynamically changing environment. A 100 ms end-to-end delay requirement for the object detection results is needed to provide timely feedback for the user [26].

In the previous generation of the VIS^{4}ION system, all machine vision is performed locally by the Nvidia Jetson board, which limited the image resolution and the frame rate at which object detection can be performed. Furthermore, the battery needed to support prolonged operation of the object detection model adds considerably to the backpack weight. Hence, we investigate the possibility of utilizing edge computing to perform object detection inference for the VIS^{4}ION system. The video captured by the system is compressed using the H.265 video coding standard and is streamed to an edge server via 5G mmWave wireless connection.

Similar to compression by image codecs, compression by H.265 introduces distortion to the video frames that can affect the detection performance. Given a target rate for a camera, the video can be compressed at different spatial resolutions (frame size in terms of pixels) and temporal resolutions (frame rate), as illustrated in Fig 5.1. With the chosen spatiotemporal resolution, the bitrate is controlled by the quantization stepsize, which controls the amplitude resolution and affects the pixel quality. While there has been significant work in relating spatial, temporal, and amplitude resolution (STAR) to perceptual video quality [20, 30, 38], the effect of STAR on object detection accuracy is less understood. Here, we conduct a study to systematically evaluate the impact of spatial and amplitude resolution on the object detection accuracy using the YOLOv5s model. We leave out the consideration of the temporal resolution at this time because the YOLO model works on video frames independently. This study enables us to determine the optimal spatial resolution for a given bitrate, and the achievable detection accuracy under the optimal resolution at this rate.

5.2 Creation of the NYU-Street Scene Dataset

To test the performance of the YOLO model for detecting objects of interest for pedestrian navigation, we recorded a set of videos while wearing the VISION backpack which has a single stereo camera on the front shoulder strap. The camera model is the ZED camera from StereoLabs [24]. The recording was taken in the streets near the NYU Tandon Campus and the NYU Washington Square Campus, with pedestrian traffic and street crossings. A total of 9 videos were captured with

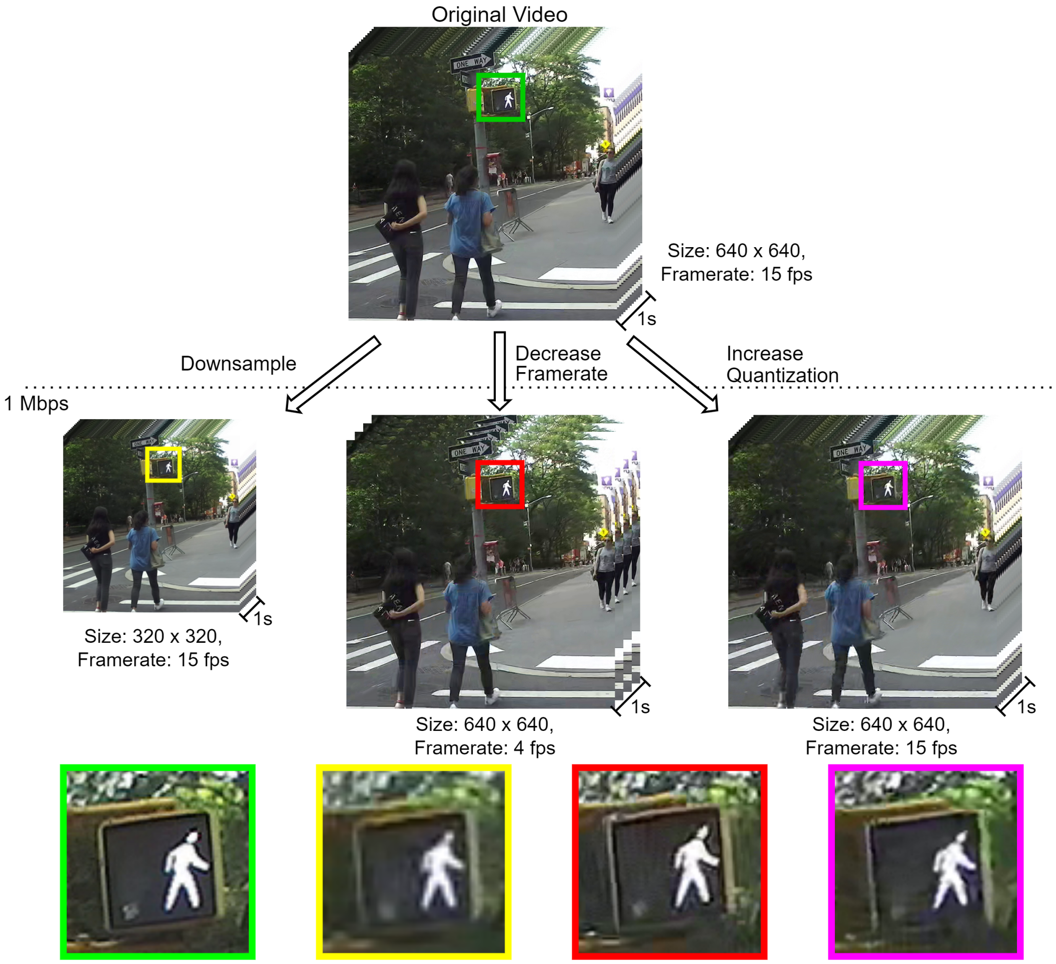


Figure 5.1: Under the same bitrate constraint, one can represent a video using different combinations of spatial, temporal, and amplitude resolutions as shown here with an example video compressed to 1 Mbps. The bottom row shows a crop from each version of the video to better illustrate the differences in compression artifacts.

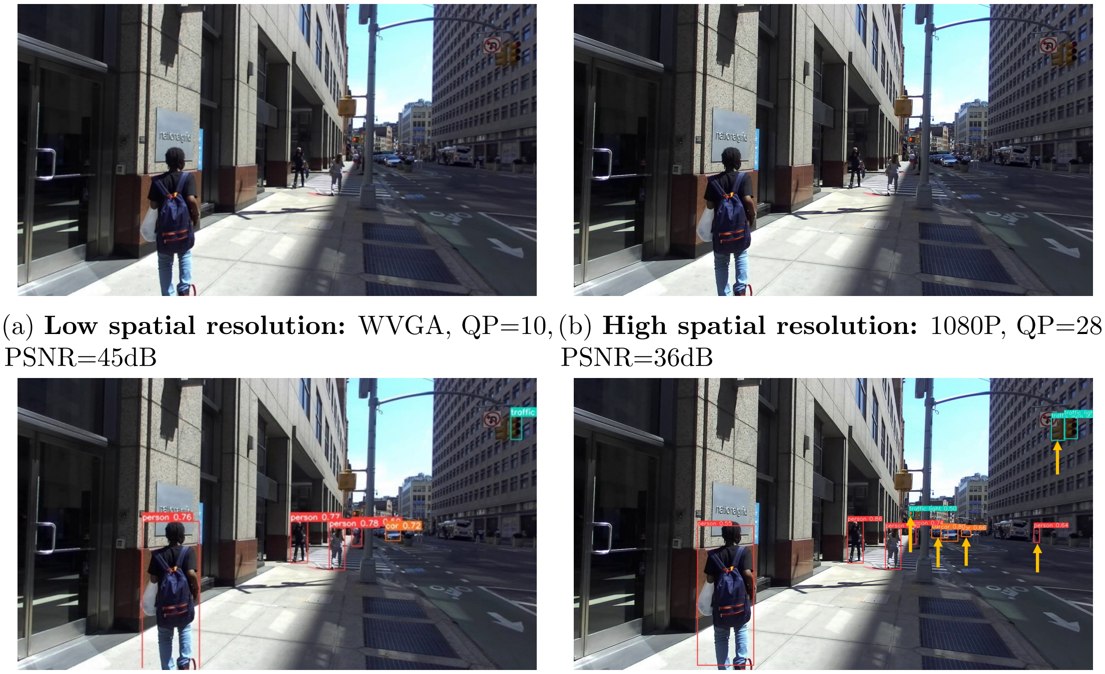
the ZED at the 2.2K spatial resolution and 15 fps temporal resolution, with a total video length of 43 minutes. The videos were manually annotated with bounding boxes for 15 objects that are related to traffic applications.

A sample frame from the dataset is shown in Figure 5.2. The frame was taken from one of the video in the dataset compressed at 10 Mbps but with different spatial and amplitude resolutions. For the higher spatial resolution frame, the YOLO model was able to detect more objects that are far away. However, it is not always the case that higher spatial resolution gives better detection results. As we will show through our experiment, the optimal spatial and amplitude trade-off to achieve the best detection result varies according to the bitrate.

5.3 Effect of Spatial and Amplitude Resolution on Object Detection Accuracy

We compressed all videos (left view only) in the StreetScene dataset using the FFmpeg software with the x265 codec [16, 49], which follows the latest international video coding standard H.265/HEVC [46]. We kept the same temporal resolution and compressed the video either at the original 2.2K spatial resolution or reduced spatial resolutions under different quantization parameters (QPs). Default downsampling filters ('bicubic') in FFmpeg were used for the spatial downsampling. Considering the low-delay requirement of the navigation application, we used a Group of Picture (GOP) length of 60 frames, without B-frames, i.e, each GOP starts with one I-frame, followed by 59 P-frames.

The decompressed video was used to evaluate the performance of the pretrained YOLOv5s model on different spatial resolutions and bitrates. Figure 5.3 shows the



(a) **Low spatial resolution:** WVGA, QP=10, PSNR=45dB

(b) **High spatial resolution:** 1080P, QP=28, PSNR=36dB

(c) YOLO detection results for (a)

(d) YOLO detection results for (b)

Figure 5.2: Sample detection results from videos both compressed at 10 Mbps but using different spatial and amplitude resolutions. Objects that are indicated by yellow arrows in (d) are missed in (c). Note that although the image in (a) and (c) have lower spatial resolution than those in (b) and (d), we display them at the same size for easier visual comparison.

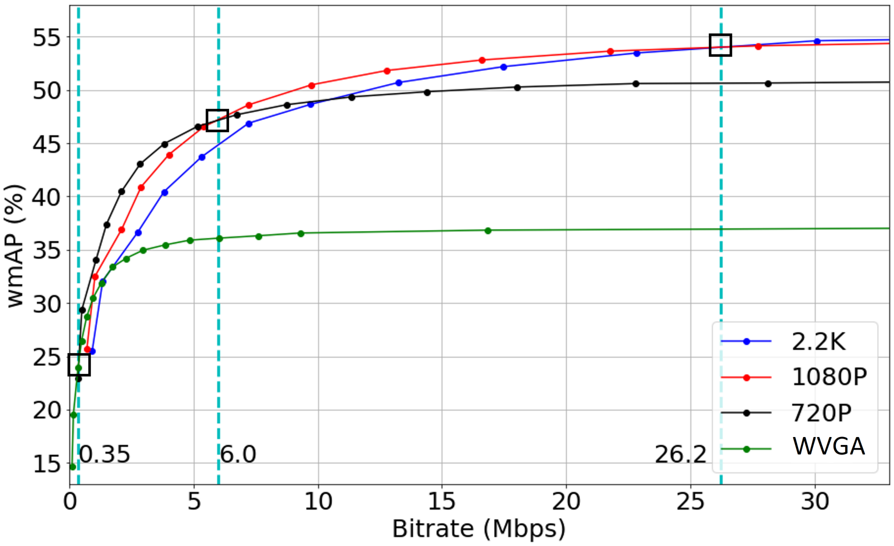


Figure 5.3: Mean detection accuracy (weighted mean AP) for 11 object types vs. bitrate for different spatial resolutions. Different points in the same curve correspond to different QPs.

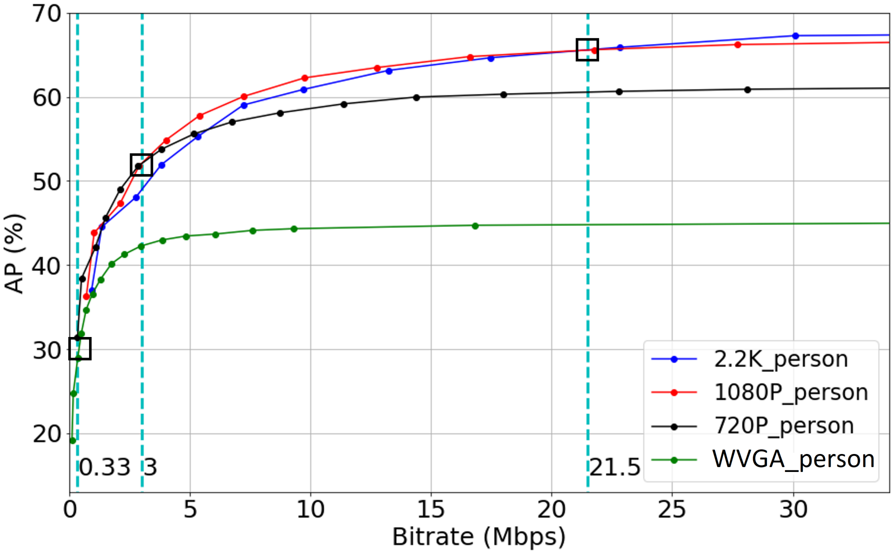


Figure 5.4: Detection accuracy (AP) for person vs. bitrate for different spatial resolutions. Different points in the same curve correspond to different QPs.

weighted mean average precision (wmAP) over 11 objects in the dataset vs. bitrate. The figure reveals that there is an optimal spatial resolution at each bitrate that will maximize the wmAP. Specifically, 720P is best for 0.35-6.0 Mbps, 1080P for 6.0-26.2 Mbps, 2.2K for higher bitrates. However, 2.2K provides only marginal improvement over 1080P above 26.2 Mbps. We note that this could be because the YOLO model was trained mainly on low-resolution images. Additionally, we note that since the YOLOv5s network was trained with uncompressed images, it led to lower performance when used for detection on compressed video frames. Nevertheless, we expect that the effect of spatial resolution on mAP will remain the same when a model trained on compressed images is used.

Fig. 5.4 presents the detection result for the person category. We see a similar trend as in Fig. 5.3, although the specific rate points where higher resolutions take over the lower resolutions are slightly different. The AP for the person category is higher than the wmAP over 11 objects at similar bitrates, which shows that the YOLO model is more effective in detecting people than other object categories.

5.4 Computational Complexity vs. Spatial Resolution

To measure the computation complexity of running the YOLO model, we used the Jetson Xavier NX to measure the runtime of performing local processing and used an RTX 8000 GPU to measure the runtime of edge server inference. Table 5.1 summarizes the computation complexity (measured by the FLOP count), the inference time per video frame, and corresponding speed (frame/sec or fps) for videos at different spatial resolutions for the two options.

To assess the feasibility of offloading computation through wireless connection, we conducted a realistic wireless network simulation of a user commuting in a similar environment to those from which the NYU-Pedestrian dataset was captured. The median round-trip delay of 5G mmWave and the 4G Lon Term Evolution (LTE) carrier are both measured. These wireless network measurements, combined with the detection model inference delay in Table 5.1 allow us to compare the different configurations that can be used to deploy the VIS^{4}ION system.

Table 5.2 summarizes the key results for using different configurations. Local only refers to the local processing scenario, where the captured video is processed entirely on the Jetson device. The "Local only" configuration provide a basic level of service when the wireless connection is poor or not available to the user. Using LTE and mmWave connection for computation offloading substantially reduced median round-trip delay, and thereby increased the availability of service within the delay budget. The detection accuracy increased significantly due to the availability of 1080P video. With the high network bandwidth of mmWave connection, it is possible to support a 4 camera system to provide a wider field of view for the user. We further considered an adaptive processing configuration, where the system switch between local and edge computing depending on the network condition, and the resolution of the video is adaptively switched according to the available bandwidth. The adaptive configuration enable even higher availability of service and detection accuracy compared to the other configurations.

Resolution	GFLOP	Local Inference Time (ms)	Local Inference Speed (fps)	Server Inference Time (ms)	Server Inference Speed (fps)
2.2K	57.22	232.02	4.31	23.4	42.7
1080P	43.38	178.25	5.6	18.7	53.5
720P	19.56	95.69	10.5	10.4	96.2
WVGA	5.36	75.02	13.3	5.1	196.1

Table 5.1: Impact of spatial resolution on the detection model complexity, running time on local processor and edge server, respectively. The Jetson Xavier NX is used as the local processor, while the server uses an RTX 8000 GPU.

Item	Local only		LTE only	mmWave+LTE		Adaptive	Remarks
Video configuration and object detection performance							
Number of monocular cameras	🟥 1	🟥 1	🟥 1	🟥 1	🟩 4	🟩 Variable (1-4)*	*1 camera if throughput $\leq$ 26 Mbps, more cameras if throughput >26 Mbps
Camera resolution	WVGA	720P	1080P	1080P	1080P	mostly 1080P	
wmAP (%) for multiple objects	🟥 36.6	🟧 50.3	🟩 54.0	🟩 54.0	🟩 54.0	🟩 54.0 avg†	See Table 4
AP (%) for person	44.3	60.3	66.1	66.1	66.1	65.8 avg†	See Table 4
Detection range for person (meter)	6	9	12	12	12	mostly 12	See Fig. 8
Bandwidth, Delay, and Availability							
Total uplink data rate (Mbps)	0	0	26	26	104	variable	We assume each camera stream takes 26 Mbps except in the adaptive case.
Video frame delay (ms)	33	33	33	33	33	33	We assume video is captured at 30 Hz.
Video encoding delay (ms)	0	0	17	17	17	17	We assume that video encoding takes at most 1/60 (s) per frame, because today's smart phones can capture video at 60 Hz. Local processing does not need to compress video.
Inference time (ms)	75	96	19	19	19	variable	See Table 4. We assume multiple GPUs process separate frames simultaneously when multiple camera data are uploaded.
Median round-trip time (RTT) (ms)	-	-	37	15	15	15	See Fig. 13(b).
Median total delay (ms)	🟧 108	🟥 129	🟧 106	🟩 84	🟩 84	🟩 84	Frame delay+encoding delay+inference time+RTT
Availability with total delay $\leq$ 100 ms	🟥 0%	🟥 0%	🟥 0%	🟧 75%	🟧 65%	🟧 78%	Computed from CDF of delay constrained throughput. See text.
Availability with total delay $\leq$ 150 ms	🟩 100%	🟩 100%	🟩 95%	🟩 97%	🟧 67%	🟩 100%	

🟥 Performance value is poor

🟧 Performance value is medium

🟩 Performance value is good

†The average wmAP and AP presented are for a total delay of $\leq$ 100 ms. These numbers are 53.9% and 66.0%, respectively for a total delay $\leq$ 150 ms.
There are many other feasible configurations, including
(1) Processing 1080P video locally for increased detection accuracy, at a total delay of 211 ms.
(2) Sending both views of each stereo camera or one view plus depth map, with increased uplink rate, and reduced availability.

Table 5.2: Comparison summary of example configurations in different connectivity scenarios.

5.5 Delay Compensation for Object Detection

We have shown that the median round trip delay for mmWave edge computing is under the 100 ms requirement for BVI navigation. However, in an environment where objects in the scene have rapid movement, even a short delay may result in a large discrepancy between the received bounding box result and the objects in the current scene. There may also be fluctuation in the channel condition that brings the delay to much higher than the median. The discrepancy between the received detection results for a past frame and the actual objects that are in the current frame lead to an effectively lower detection accuracy. To address this problem, we propose a method for compensating the bounding box error due to round trip delay.

We consider the following scenario in which a video stream is being sent to an edge server for object detection. At time t, the frame x_t is captured by the camera. The frame is compressed by the video encoder, transmitted through the wireless connection, decoded and processed by the server, and the result of the detection is sent back to the mobile device. The detection result $\hat{b}_t$ for frame x_t is recevied by the mobile device at time $t + \Delta$, with the delay Δ being the total duration from the time the frame is captured to the time the result is received.

In evaluating the accuracy of a detection method, we would typically compare b_t, the ground-truth bounding boxes at time t, with the detection results at the same time $\hat{b}_t$. However, considering that in a dynamically changing environment, the position of the objects in the scene would have changed significantly during the short duration, it is more reasonable to compare the detection results $\hat{b}_t$ with the ground-truth results at $b_{t+\Delta}$, corresponding to the frame $x_{t+\Delta}$. Due to changes in positions of the objects in the frame and the possibility of new objects appearing or objects moving out of the frame, the detection accuracy will decrease as the delay

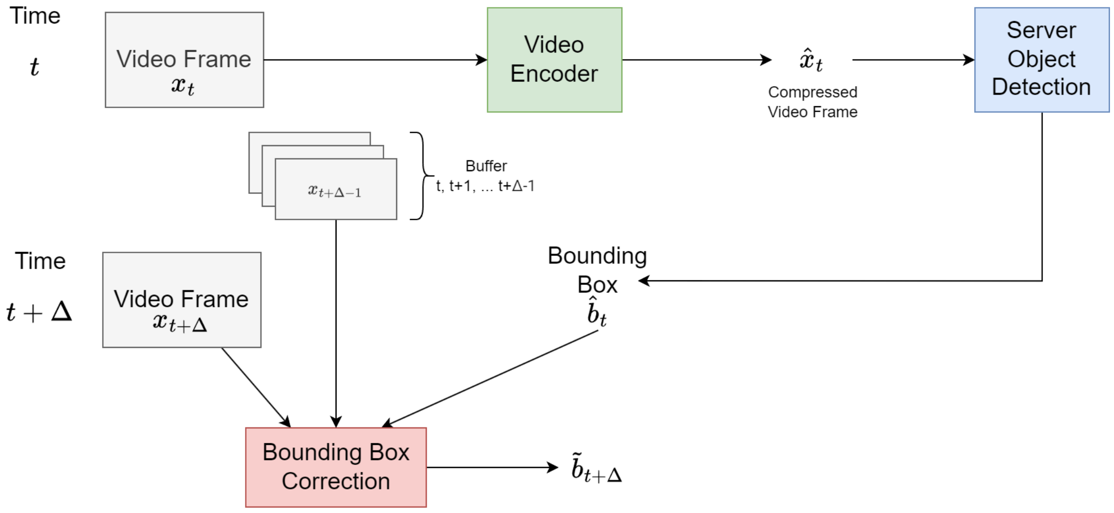


Figure 5.5: Proposed system for correcting the error in bounding box due to delay in the edge computing pipeline.

Δ increases.

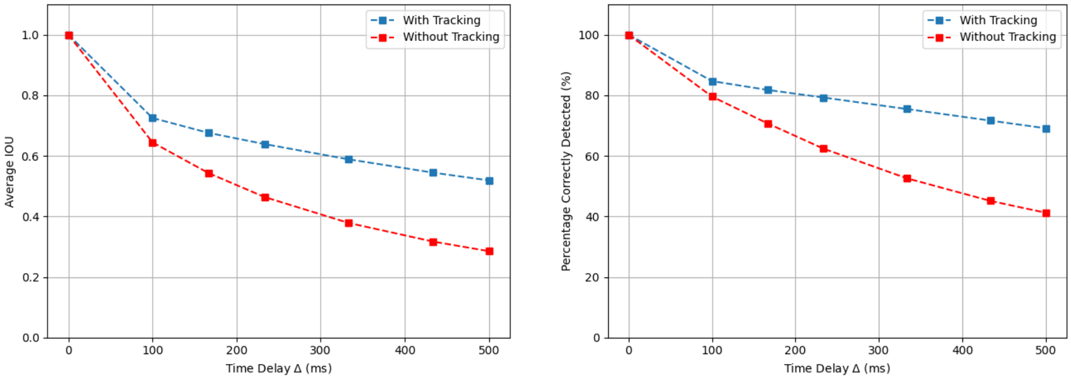
To alleviate the decrease in accuracy due to delay, we propose a system to compensate for the movement of objects in the frame. The system is shown in Figure 5.5. A buffer is maintained on the mobile side to store past frames captured by the camera. We assume that the length of this buffer is appropriately designed so that it exceeds the typical round-trip delay. When a bounding box for time t is received at time $t + \Delta$, the past frames in the buffer $x_t, x_{t+1}, x_{t+\Delta-1}$ and the current frame $x_{t+\Delta}$ are used to adjust the bounding boxes to the current position of the object in frame $x_{t+\Delta}$.

Object tracking is performed to track each object in the received result $\hat{b}_t$ through the series of buffered frames up to the current frame. For each object, we start with the bounded region of the object in the frame x_t as the template. We then apply template matching to search for the region in frame x_{t+1}, or the target frame, that has the least difference from the template. The object's new bounded region in frame x_{t+1} is used as the template to search for the next matching region in frame x_{t+2}, and so on. At each step, the search region is limited to an area that extends the template box's height and width by 0.5 times on both sides in the vertical and horizontal directions respectively.

To handle the case where the object is near the edge of the frame and the search area needs to extend outside of the target frame, we extend the height and width of the target frame by 0.25 times the original height and width on both sides by replicating the values on the frame border. Ideally, we should ignore the difference contributed by the part of the template that extends outside the frame during the search. But for the ease of implementation, we found that good performance can be achieved even if the difference outside the original frame border is counted.

To measure the decrease in detection accuracy due to delay and the performance of the tracking method in recovering the loss in accuracy, we ran an experiment on a 30 fps video of a pedestrian walking in the street. We use average IOU (Intersection Over Union) and the percentage of correct detection as metrics. Average IOU is calculated as follows. For each ground truth object in a frame, we find the detected/tracked object in the same class and has the highest IOU with the ground truth to find the best matching pairs. We then compute the average of IOU of the best matching pairs of ground truth and tracked results in the video. The percentage of correct detection is defined as the percentage of ground truth objects that have an IOU with a detected/tracked object greater than 0.5.

We measured these two metrics with an artificially introduced delay and varied the amount of delay Δ. For each time step, we assume that the detected results without delay is the ground truth, i.e. let $b_t = \hat{b}_t$. This is appropriate since we are only interested in how delay affects detection accuracy over time relative to the initial frame x_t. For a certain Δ, we compute the Average IOU and Percentage



(a) Average IOU

(b) Percentage of correct detection

Figure 5.6: A figure with two subfigures

of correct detection by comparing $\hat{b}_t$ and $b_{t+\Delta}$ over all possible t in the video to obtain the performance without tracking. We then perform template matching for each $\hat{b}_t$ using buffered frames x_t to $x_{t+\Delta}$ to obtain the tracked results $\tilde{b}_{t+\Delta}$. We compute the two metrics between each $\tilde{b}_{t+\Delta}$ and $\hat{b}_{t+\Delta}$ to obtain the performance with tracking.

The results of comparing bounding box accuracy with and without tracking are shown in Figure 5.6. Without tracking, both the Average IOU and the percentage of correctly detected boxes decrease rapidly as the delay in results increases. At 500 ms delay, only 40% of the bounding box are correctly detected. With object tracking, the decrease in detection accuracy is less steep. At 500 ms delay, close to 70% of the bounding boxes are still correctly detected.

Chapter 6

Conclusion

This thesis explored the problem of image and video compression in offloading the computation of visual anaylytics model. Using learned compression methods as well as traditional video codec, we proposed methods that aim to improve split computing and edge computing systems with increased rate-task performance, lower model complexity, and lower delay.

We first investigated the use of learned image compression model to compress the image data for computation offloading. Using image compressor trained for MSE reconstruction leads to low performance when the compressed image is used for visual analytics tasks. Training the task model on compressed images improved the task performance. Joint training of the compressor and task model pipeline using a rate-task loss further improved the rate-task performance.

We next explored feature compression in the split computing framework for computation offloading. We proposed a light-weight trainable feature compression architecture, that includes feature channel/spatial reduction and expansion and hyperprior-based entropy coding/decoding. With end-to-end training of the feature compressor and object detector using rate-detection loss, our approach can achieve higher detection accuracy at low to medium rate range than baseline methods that perform image compression at the mobile device and object detection on the server. Furthermore, our approach has significantly lower runtime at the mobile device than the baseline methods using learned compression models.

Building on the proposed feature compression method, we proposed a scalable feature compression model that allows for variable bitrate compression and efficient adaptation to dynamic network. The intermediate feature is compressed into a layered bitstream, with the number of scalable layers received by the decoder scaling with the analytics performance of the task model. The proposed scalable model using a single set of model weights achieved comparable performance over a large rate range compared to the non-scalable model that requires model switching. In addition, the scalable compression method offers extra protection against packet drop.

Finally, we studied the feasibility of using the H.265 video codec to compress a video stream for computation offloading to an edge server. In particular, we targeted the problem of object detection for navigation assistance for the blind and visually impaired population. A large video dataset of pedestrian street scene is collected and annotated. Using this dataset, we measured the effect of spatial and amplitude resolution on detection accuracy. We proposed a resolution adaptive compression strategy to maximize detection accuracy. To reduce the error caused by delay in receiving the detection results, we proposed a system to perform mobile-side object tracking on buffered frames to update bounding box location. The proposed system was effective in correcting the bounding box error caused by delay and recovering lost detection accuracy.

Bibliography

[1] I. V. Bajić, W. Lin, and Y. Tian. Collaborative intelligence: Challenges and opportunities. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8493–8497. IEEE, 2021.

[2] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[3] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[4] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

[5] F. Bellard. Bpg image format, 2018.

[6] A. Boldini, A. L. Garcia, M. Sorrentino, M. Beheshti, O. Ogedegbe, Y. Fang, M. Porfiri, and J.-R. Rizzo. An inconspicuous, integrated electronic travel

aid for visual impairment. *ASME Letters in Dynamic Systems and Control*, 1(4):041004, 2021.

[7] L. D. Chamain, F. Racapé, J. Bégaint, A. Pushparaja, and S. Feltman. End-to-end optimized image compression for machines, a study. In *2021 Data Compression Conference (DCC)*, pages 163–172. IEEE, 2021.

[8] Z. Chen, K. Fan, S. Wang, L. Duan, W. Lin, and A. C. Kot. Toward intelligent sensing: Intermediate deep feature compression. *IEEE Transactions on Image Processing*, 29:2230–2243, 2019.

[9] H. Choi and I. V. Bajić. Scalable image coding for humans and machines. *IEEE Transactions on Image Processing*, 31:2739–2754, 2022.

[10] J. Choi and B. Han. Task-aware quantization network for jpeg image compression. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pages 309–324. Springer, 2020.

[11] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 113–123, 2019.

[12] 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 *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 182–188. IEEE, 2022.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A

large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[14] L. Duan, J. Liu, W. Yang, T. Huang, and W. Gao. Video coding for machines: A paradigm of collaborative compression and intelligent analytics. *IEEE Transactions on Image Processing*, 29:8680–8695, 2020.

[15] A. E. Eshratifar, A. Esmaili, and M. Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2019.

[16] FFmpeg project. FFmpeg. `https://ffmpeg.org/`.

[17] W. Gao, S. Liu, X. Xu, M. Rafie, Y. Zhang, and I. Curcio. Recent standard development activities on video coding for machines. *arXiv preprint arXiv:2105.12653*, 2021.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] H. Hu, Z. Ma, and Y. Wang. Optimization of spatial, temporal and amplitude resolution for rate-constrained video coding and scalable video adaptation. In *2012 19th IEEE International Conference on Image Processing*, pages 717–720. IEEE, 2012.

[20] H. Hu, Z. Ma, and Y. Wang. Optimization of spatial, temporal and amplitude resolution for rate-constrained video coding and scalable video adaptation. In

2012 19th IEEE International Conference on Image Processing, pages 717–720. IEEE, 2012.

[21] C. Jia, Z. Liu, Y. Wang, S. Ma, and W. Gao. Layered image compression using scalable auto-encoder. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 431–436. IEEE, 2019.

[22] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[23] Y. Kim, S.-Y. Jeong, J. Lee, J. Lee, and M. Kim. Pixel-unshuffled multi-level feature map compression for fcvcm. In *2023 IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2023.

[24] S. Labs. ZED 2 Camera product page. `https://www.stereolabs.com/zed-2`.

[25] N. Le, H. Zhang, F. Cricri, R. Ghaznavi-Youvalari, and E. Rahtu. Image coding for machines: an end-to-end learned approach. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1590–1594. IEEE, 2021.

[26] R. J. Leigh and D. S. Zee. *The neurology of eye movements*. Contemporary Neurology, 2015.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[28] J. Liu, H. Sun, and J. Katto. Learned image compression with mixed transformer-cnn architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14388–14397, 2023.

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

[30] Z. Ma, F. C. Fernandes, and Y. Wang. Analytical rate model for compressed video considering impacts of spatial, temporal and amplitude resolutions. In *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–6. IEEE, 2013.

[31] T. maintainers and contributors. TorchVision: PyTorch's Computer Vision library, Nov. 2016.

[32] Y. Matsubara, M. Levorato, and F. Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–30, 2022.

[33] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2685–2695, 2022.

[34] Y. Mei, L. Li, Z. Li, and F. Li. Learning-based scalable image compression with latent-feature reuse and prediction. *IEEE Transactions on Multimedia*, 24:4143–4157, 2021.

[35] D. Minnen, J. Ballé, and G. Toderici. Joint autoregressive and hierarchical priors for learned image compression. *arXiv preprint*, 2018.

[36] D. Minnen, J. Ballé, and G. D. Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018.

[37] Y.-F. Ou, Y. Xue, and Y. Wang. Q-star: A perceptual video quality model considering impact of spatial, temporal, and amplitude resolutions. *IEEE Transactions on Image Processing*, 23(6):2473–2486, 2014.

[38] Y.-F. Ou, Y. Xue, and Y. Wang. Q-STAR: A perceptual video quality model considering impact of spatial, temporal, and amplitude resolutions. *IEEE Transactions on Image Processing*, 23(6):2473–2486, 2014.

[39] Y. Pang, J. Cao, Y. Li, J. Xie, H. Sun, and J. Gong. Tju-dhd: A diverse high-resolution dataset for object detection. *IEEE Transactions on Image Processing*, 30:207–219, 2020.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[42] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once:

Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[43] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology*, 17(9):1103–1120, 2007.

[44] S. Singh, S. Abu-El-Haija, N. Johnston, J. Ballé, A. Shrivastava, and G. Toderici. End-to-end learning of compressible features. In *2020 IEEE ICIP*, pages 3349–3353. IEEE, 2020.

[45] G. J. Sullivan, J. M. Boyce, Y. Chen, J.-R. Ohm, C. A. Segall, and A. Vetro. Standardized extensions of high efficiency video coding (hevc). *IEEE Journal of selected topics in Signal Processing*, 7(6):1001–1016, 2013.

[46] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[47] t. **Yuan**, T. Azzino, Y. Hao, Y. Lyu, H. Pei, A. Boldini, M. Mezzavilla, M. Beheshti, M. Porfiri, T. E. Hudson, et al. Network-aware 5g edge computing for object detection: Augmenting wearables to “see” more, farther and faster. *IEEE Access*, 10:29612–29632, 2022.

[48] Ultralytics. YOLOv5. `https://github.com/ultralytics/yolov5`, 2020.

[49] VideoLAN Organization. x265. `https://www.videolan.org/developers/x265.html`.

[50] Y. Wang and D. Mukherjee. The discrete cosine transform and its impact on visual compression: Fifty years from its invention [perspectives]. *IEEE Signal Processing Magazine*, 40(6):14–17, 2023.

[51] Y. Wang, J. Ostermann, and Y.-Q. Zhang. *Video processing and communications*, volume 1. Prentice hall Upper Saddle River, NJ, 2002.

[52] Z. Wang, F. Li, Y. Zhang, and Y. Zhang. Low-rate feature compression for collaborative intelligence: Reducing redundancy in spatial and statistical levels. *IEEE Transactions on Multimedia*, 2023.

[53] N. Yan, D. Liu, H. Li, and F. Wu. Semantically scalable image coding with compression of feature maps. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3114–3118. IEEE, 2020.

[54] 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 *2023 IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–5. IEEE, 2023.

[55] Z. Yuan, T. Azzino, Y. Hao, Y. Lyu, H. Pei, A. Boldini, M. Mezzavilla, M. Beheshti, M. Porfiri, T. E. Hudson, et al. Network-aware 5g edge computing for object detection: augmenting wearables to “see” more, farther and faster. *IEEE Access*, 10:29612–29632, 2022.

[56] Z. Yuan, S. Garg, E. Erkip, and Y. Wang. Scalable feature compression for edge-assisted object detection over time-varying networks. In *MLSys 2023 Workshop on Resource-Constrained Learning in Wireless Networks*, 2023.

[57] Z. Yuan, S. Rawlekar, S. Garg, E. Erkip, and Y. Wang. Feature compression for rate constrained object detection on the edge. In *2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 1–6. IEEE, 2022.

Publications List

1. **Z. Yuan**, S. Rawlekar, S. Garg, E. Erkip, and Y. Wang. Split computing with scalable feature compression for visual analytics on the edge. Accepted for publication at *IEEE Multimedia*, 2024.

2. **Z. Yuan**, S. Garg, E. Erkip, and Y. Wang. Scalable feature compression for edge-assisted object detection over time-varying networks. In *MLSys 2023 Workshop on Resource-Constrained Learning in Wireless Networks*, 2023.

3. Y. Hao, H. Pei, Y. Lyu, **Z. Yuan**, J.-R. Rizzo, Y. Wang, and Y. Fang. Understanding the impact of image quality and distance of objects to object detection performance. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11436–11442. IEEE, 2023

4. **Z. Yuan**, S. Rawlekar, S. Garg, E. Erkip, and Y. Wang. Feature compression for rate constrained object detection on the edge. In *2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 1–6. IEEE, 2022.

5. **Z. Yuan**, T. Azzino, Y. Hao, Y. Lyu, H. Pei, A. Boldini, M. Mezzavilla, M. Beheshti, M. Porfiri, T. E. Hudson, et al. Network-aware 5g edge computing

for object detection: Augmenting wearables to "see" more, farther and faster. *IEEE Access*, 10:29612–29632, 2022.

6. **Z. Yuan**, H. Liu, D. Mukherjee, B. Adsumilli, and Y. Wang. Block-based learned image coding with convolutional autoencoder and intra-prediction aided entropy coding. In *2021 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2021.