Visual Coding for Machines

by

Bardia Azizian

M.Sc., Sharif University of Technology, 2018
B.Sc., University of Tehran, 2015

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2022

Declaration of Committee

Name:	**Bardia Azizian**
Degree:	**Master of Applied Science**
Thesis title:	**Visual Coding for Machines**
Committee:	**Chair:** Andrew Rawicz Professor, Engineering Science
	Ivan V. Bajić Supervisor Professor, Engineering Science
	Hadi Hadizadeh Committee Member Research Assistant, Engineering Science
	Jie Liang Examiner Professor, Engineering Science

Abstract

In this thesis, we study different approaches to visual data coding for machines and develop new methods to address some issues in this area. We mainly focus on the Image and Video signals processed by a Deep Neural Network (DNN)-based computer vision model. Our proposed methods are designed to be utilized in DNN-based machines deployed collaboratively on the edge and cloud. This framework is called Collaborative Intelligence (CI), in which a DNN model is split into two parts such that the edge device runs the first few layers, and the remaining layers are executed on the cloud. To that end, the intermediate feature tensors need to be coded and transferred to the cloud. This research explicitly attempts to provide solutions for efficient coding of these tensors, considering challenges such as motion estimation and compensation for videos in the latent space, and privacy for images.

Keywords: coding for machines, feature coding, collaborative intelligence, deep neural network, motion estimation and compensation, privacy

Table of Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
BD	Bjøntegaard delta
BPP	Bits Per Pixel
CI	Collaborative Intelligence
CNN	Convolutional Neural Network
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
DP	Differential Privacy
DPCM	Differential Pulse Code Modulation
HDR	High Dynamic Range
HEVC	High Efficiency Video Coding
IDCT	Inverse Discrete Cosine Transform
IoT	Internet of Things
IoU	Intersection over Union
M2M	Machine-to-Machine
mAP	mean Average Precision
MC	Motion Compensation
ME	Motion Estimation
MI	Mutual Information
ML	Machine Learning
MV	Motion Vector
MPEG	Moving Picture Experts Group
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
QP	Quantization Parameter
RDO	Rate-Distortion Optimization
VCM	Video Coding for Machines

Chapter 1

Introduction

The rapid growth of Artificial Intelligence (AI) applications such as Internet of Things (IoT), smart cities, visual surveillance, autonomous driving, industrial machine vision, etc., has resulted in a proliferation of "intelligent" edge devices, sensors, and their associated infrastructure. Our smartphones running a voice-controlled virtual assistant, traffic control cameras together with their built-in object detection model, or a temperature sensor in a smart home, are only some examples of such nodes that are usually a part of a wider intelligent network. These nodes need to communicate with each other and cloud-based services to accomplish specific tasks. According to Cisco's annual Internet report [23], most globally connected devices will be allocated to Machine-to-Machine (M2M) connections by 2023. Therefore, there is an urgent need to develop efficient data coding schemes specifically for machines. Accordingly, significant research effort is being devoted to this area. Furthermore, standardization activities such as JPEG-AI [44] and MPEG-VCM [33] have also been initiated to standardize image and video communications between machines and possibly humans. We also focus on image and video data in this thesis.

For decades, many image/video coding standards have been designed for two ultimate goals: decreasing the number of bits representing the input and maintaining the fidelity between the original and reconstructed input. The latter is pursued since the end-users of image/video codecs have always been humans. The higher quality of the reconstructed input is indeed appealing to the human visual system, especially when the fidelity is measured using perceptual metrics [56]. Increasingly, however, much of the visual content is only "seen" by machines in applications like autonomous driving and navigation, traffic monitoring, surveillance, etc. In such cases, it might be more efficient to code task-relevant features derived from the input rather than the inputs themselves. The utility of feature coding has been recognized even prior to the current wave of interest in DNNs through MPEG standards on Compact Descriptors for Visual Search (CDVS) [67] and Compact Descriptors for Visual Analysis (CDVA) [66].

More recently, feature compression has been studied in the context of Collaborative Intelligence (CI) [7, 84], where a DNN is distributed between an edge device and the cloud,

and intermediate features need to be uploaded from the edge to the cloud to complete the inference. We will introduce CI in more detail in the following section.

1.1 Collaborative Intelligence

The widespread popularity of AI and deep learning methods have given rise to deploying these models in our daily life for various automated tasks with remarkably high accuracies. Face recognition, speech recognition, natural language processing, and fingerprint detection are some instances of deep learning-based tasks that our smartphones are capable of performing. There are many other examples in larger-scale applications. However, the most critical challenge in exploiting AI-based schemes is the limited computational power of edge devices. Running a sophisticated DNN model entirely on an edge device would be severely time and energy-consuming. The common practice to overcome this problem is transferring the inputs to the cloud, where the whole DNN model can be executed on powerful GPUs and CPUs. However, this approach has its own downsides. Transmitting a massive amount of input data to the cloud would cause high bandwidth consumption. Besides, compressing, coding, and transmitting this amount of data is not a simple task either for an edge device and could be energy-consuming. Also, running the whole model on the cloud would lower its throughput by escalating the server load. And lastly, the end-to-end latency to accomplish a specific task becomes high [49].

Collaborative Intelligence paves a promising way to overcome the abovementioned issues. With advancements in the hardware design and computational power of edge devices, it is possible to run the first few layers of DNN models on them. Then, the obtained feature tensors can be coded and sent to the cloud to execute the remaining part of the model and complete the inference. This idea would reduce the amount of transmitted data, relieve the burden on the cloud and lower the end-to-end latency. In fact, the overall system would be more efficient in terms of time and energy. References [31, 49, 29, 10] verified that deploying the DNN models collaboratively on the edge and cloud would result in better performance. They also studied the tradeoff between the latency and energy that existed in the CI frameworks and attempted to optimize the utility of the system based on these constraints.

CI has another benefit, in addition to lower latency and energy consumption compared to the cloud-only approaches, which is providing better privacy. In a cloud-only framework, the edge device has to transfer the original input signal to the cloud, which could bring privacy risks if the communication channel or the cloud is untrusted. In a CI framework, however, the intermediate feature tensors are sent to the cloud. The authors of [32] claimed that recovering the input data from the latent feature tensors is impossible without having knowledge of the counterpart decoder that brings back the input data from the intercepted features. However, that is not a correct assumption since some methods, like *model inver-*

sion attack [39] in the context of DNN models, aim exactly to recover the input from the intermediate features. Nevertheless, deeper layers of DNN models carry less information about the input, which offers better privacy for sensitive data.

Collaborative Intelligence is known by other terms in the literature as well, such as Collaborative Inference [39], Edge-Cloud Inference [59], Edge-Cloud AI [10], etc. There is also another framework in the CI domain, which is employed in the training stage of the DNN models [63, 29, 85, 34]. In this framework, called federated learning, multiple parties contribute to training a single model using individual private datasets. These nodes share some information about their own models during training, such as weight parameters or gradient values, in order to have a synchronized model across all the nodes. Nonetheless, we focus on CI at the inference stage in this thesis.

1.2 Challenges of Feature Coding

The previous section explained the advantages of CI over could-only and edge-only approaches in deploying DNN models. In order to utilize CI in a practical pipeline, a coding scheme should be used to compress the intermediate feature tensors at the split point of the DNN model. Then, the compressed features are decoded on the cloud and passed to the remainder of the model to obtain the outcomes. Since deep learning is relatively new, feature tensor coding is not as mature as natural input data coding. Image and video coding standards have evolved for decades and now are able to exploit the redundancies that exist in the natural signals to a great extent. But, deep feature coding has yet to be grown enough to achieve good compression efficiency.

Dealing with spatial and especially temporal redundancies in video signals is more challenging. In traditional video coding standards (e.g., H.265/HEVC [87] and H.266/VVC [16]), the spatial redundancies are removed using advanced intra-prediction modules, and temporal redundancies are taken care of through inter-prediction using sophisticated Motion Estimation (ME) and Motion Compensation (MC) schemes. But, ME and MC in the latent space are not as easy as in the original input domain for the reasons elaborated in Chapter 3, where we aim to propose learning-based ME and MC methods in the latent space.

Privacy is another critical challenge in the deep learning context [65, 58] which will be addressed in this thesis. Privacy issue becomes more serious in CI-based applications since some features of input data are transferred to the cloud. As mentioned earlier, it is still possible to recover the input data from the intermediate feature tensors to some extent. As a result, an adversary eavesdropping on the communication channel or a malicious agent in an untrusted cloud could gain access to sensitive data and jeopardize users' privacy. To overcome this threat, we propose a method in which the sensitive parts of the data are removed from the feature tensors without significantly affecting the machine vision accuracy.

That is possible because the exact visual content of input data is not necessary for machine vision.

1.3 Thesis Outline and Contributions

The rest of this thesis is organized as follows. Chapter 2 provides a privacy-preserving solution for image coding for machines in a CI framework. In Chapter 3, we study the motion relation in videos between the input and latent space and propose a couple of DNN models for ME and MC in the latent space. The conclusions and future works are also presented in Chapter 4.

Thus far, the research reported in this thesis has resulted in one conference paper:

- B. Azizian, I. V. Bajić, "Privacy-preserving feature coding for machines," to be presented at *2022 Picture Coding Symposium*

Chapter 2

Privacy in Image Coding for Machines

This chapter is intended to elucidate the privacy issue in Collaborative Intelligence frameworks and provide a solution for image coding for machines in a way that protects the privacy of the end users. First and foremost, the importance of privacy in CI and the concept of our proposed idea is demonstrated in Section 2.1. Then, a brief literature review is given in Section 2.2, although the field of privacy in CI is less explored. We describe our proposed method in Section 2.3 and provide the experimental results in Section 2.4 before concluding this chapter.

2.1 Introduction

With the proliferation of AI-based applications such as IoT, smart home, smart city, autonomous driving, etc., intelligent nodes (e.g., cameras, sensors) whose purpose is collecting data are found in many places. This phenomenon brings about critical privacy challenges, especially in edge-cloud inference applications, where the input data or some features of it are transmitted to the cloud. In this case, an untrusted party or an adversary eavesdropping on the communication channel can easily gain access to private data. Assume a scenario where surveillance and traffic monitoring cameras are available in many public places; people and their cars can be simply tracked. In a smart home application, a person's private information is also at risk even in their own home in case a data transmission to the cloud happens.

As mentioned earlier, CI is not a perfect solution for privacy-preserving since some input data information is recoverable from the intermediate features of a DNN model as will be seen in our experiments. These sorts of privacy attacks on the learning models are mostly explored in the context of pure machine learning or deep learning [65, 58], without taking the concepts of CI and data coding for machines into account. The security and privacy attacks on the deep/machine learning models fall into four general categories: model extraction,

model inversion, adversarial, and poisoning attacks. In the model extraction attack, the adversary tries to obtain the model parameters, which is a threat to the intellectual property of the provider. The adversarial attack is aimed at fooling a DNN model into outputting wrong predictions with high confidence by applying imperceptible perturbations to the input, which is invisible to the human eye [89]. The poisoning attack focuses on polluting the training dataset by injecting malicious samples such that the resulting model would not have a desirable accuracy in the inference stage [68]. On the other side, the adversary's goal in the model inversion attack is to access unexposed data of a DNN model during the inference stage. The action of recovering sensitive data from the output or the intercepted features of a DNN model is called "model inversion attack", which is the primary concern in edge-cloud inference applications.

Membership inference [86] is a type of model inversion attack where the adversary tries to know whether a specific data point is in the training set of a model. In this research, however, our focus is on resisting model inversion attack in which the adversary intends to reconstruct the input from the latent space of an object detection DNN model. The model inversion attack can be further classified into three subcategories based on the adversary's level of access to the model [39]:

1. White-box setting: The adversary knows everything about the DNN model, including model architecture and parameters.

2. Black-box setting: The adversary knows nothing about the model but can query it through the available APIs, i.e., it can feed the network with specific data and observe the corresponding output.

3. Black-box query-free setting: The adversary knows nothing about the model, nor can it query it.

The common approach to counter privacy threats of DNN models is adding noise to the input or intermediate features insofar as the model's accuracy is not affected considerably. This way, the accuracy of the adversary in recovering the original input could drop. Cryptography methods [83] also provide one possible solution to protect the data, although they have their own risks and challenges. But, in the context of data coding for machines, one can simply reduce or remove private information from data while retaining task-relevant information because machine vision generally needs higher-level information, rather than details of each pixel, in order to perform a given task. For example, the details of a vehicle's license plate or a person's face are not necessary if the machine vision model is only supposed to detect cars and pedestrians on the street [6]. To remove sensitive information from the latent space of a model, we have adopted an adversarial training technique [35] with proper loss functions, which will be explained in Section 2.3.

In this project, we pursue another goal besides privacy: decent compression efficiency. To that end, a compressibility loss term is also employed during training to reduce the overall bitrate of the system.

2.2 Related Work

Reference [39] was the first research work that categorized different settings of model inversion attacks (black-box, white-box, and black-box query-free) and provided well-defined methods to accomplish each attack. Its authors assumed an edge-cloud collaborative inference pipeline where the cloud is untrusted. They demonstrated that recovering the input image from the intermediate feature tensors of a classification model is feasible even in the black-box query-free setting. In the white-box setting, regularized maximum likelihood estimation technique is used where the model inversion is treated as an optimization problem. On the contrary, computing the gradients is impossible in the black-box setting since the parameters and probably the architecture of the front-end model are unknown to the adversary. Thus, an auxiliary DNN model (Inverse-Network) is employed and trained on the adversary's dataset by querying the front-end network. According to their experimental results, there is not much difference in the accuracy of the recovered input between the black-box and white-box settings. In the black-box query-free setting, a shadow model is constructed first to imitate the front-end. Then, the adversary could query the shadow model unlimitedly. In this case, the quality of the reconstructed input is less than that in the white-box and the black-box settings. In this chapter, our adversary exploits the Inverse-Network technique in the black-box setting, which would result in the best quality of the reconstructed input among all the abovementioned settings, according to [39].

Although the privacy of the DNN models in the deployment phase is less studied, the privacy of the models and their utilized training data has been on the radar of researchers since a long time ago. The concept of Differential Privacy (DP) was first introduced by Dwork et al. [28], whose intention was to protect the private information of individuals contributing to a dataset used by a machine learning or analysis model. DP is defined as follows [65]:

For $\epsilon \geq 0$, an algorithm A satisfies ϵ-DP if and only if for any pair of datasets D and D' that differ in only one element:

$$\mathcal{P}[A(D) = t] \leq e^{\epsilon}\mathcal{P}[A(D') = t] \quad \forall t \tag{2.1}$$

where, $\mathcal{P}[A(D) = t]$ denotes the probability that the algorithm A outputs t. DP attempts to approximate the effect of an individual opting out of contributing to the dataset by ensuring that any effect due to the inclusion of one's data is small. In other words, DP mathematically guarantees that anyone seeing the result of a differentially private analysis will essentially make roughly the same inference about any individual's private information, whether or

not that individual's private information is included in the dataset of the analysis model. Note that lower values of ϵ, which is called privacy budget, offer better differential privacy.

It can be proved [28] that adding a Laplacian noise, $Lap(\frac{\Delta_f}{\epsilon})$, to a deterministic function $f(\cdot)$ would generate a randomized algorithm that satisfies ϵ-DP equation (2.1). Δ_f is the global sensitivity of $f(\cdot)$ and is defined as $\Delta_f = \sup |f(D) - f(D')|$ over all the dataset pairs (D, D') that differ in only one element.

In fact, DP was initially designed to withstand the membership inference attack based on its definition. For example, in [45], a multi-party collaborative inference setting is assumed, where each party owns a private model trained on its respective data, and the client submits a set of data for inference to these parties. The goal is to prevent the client or an adversary from performing a membership inference attack on the parties' private data. DP is employed to achieve this goal by adding noise to the inference results that should be returned to the client.

On the other hand, some research works have adopted DP to protect neural networks against input reconstruction in model inversion attack [78, 96, 45, 40]. For instance, the authors of [96] add Laplacian noise to the intermediate features of the DNN model that should be transmitted to the cloud, as suggested in DP. Since finding the global sensitivity of the features is difficult, they first bound the infinity norm of the tensors by a specific value. In addition to the Laplacian noise, a nullification operation is also conducted to enhance privacy. To increase the robustness of the back-end to these feature perturbations, it is retrained on the manipulated features. However, the performance of their model is severely impacted by increasing the intensity of the noise and nullification process.

Reference [78] has studied the privacy-utility trade-off based on the DP approach. This trade-off can be controlled by the privacy budget's parameter (ϵ). Similar to [96], a Laplacian noise is added to the intermediate features of the DNN model. The model inversion attack for the white-box setting, introduced in [39], is used to measure privacy.

The authors of [39] have also improved their research in a more recent paper [40] by providing solutions to defend against their proposed types of model inversion attacks introduced in [39]. Noise obfuscation, using Laplacian and Gaussian noise, and dropout technique have been studied in their paper. Dropout, which deactivates random neurons in a layer of a DNN model, turned out to be substantially better than adding noise in terms of the privacy-utility trade-off.

Unlike the abovementioned schemes, in which noise addition was the most common practice, [5] incorporated the privacy-preserving method into the feature compression task. Features with less private and more non-private information are only lightly compressed, whereas those that carry more private information are compressed more heavily to protect privacy. Distinguishing between private and non-private data is conducted through an information-theoretic privacy approach called *privacy fan*. Here, features from a DNN are scored according to the Mutual Information (MI) [25] relative to the private and non-private

tasks. At its core, privacy fan is a feature selection method based on MI between features and private/non-private tasks, which is difficult to estimate in high-dimensional spaces [51]. However, our proposed approach avoids this challenge by using an autoencoder to reduce the dimensionality of the bottleneck. For this reason, it is also more flexible than the privacy fan because it enables not just feature selection but also feature modification.

Furthermore, our approach employs an adversarial training strategy to modify the bottleneck's features via proper loss functions. In this sense, the DNN model would be optimized in a way to preserve the crucial task-relevant information and remove or manipulate the information related to accurate input reconstruction. Optimization through learning could definitely be more efficient than general nonadaptive approaches such as noise obfuscation.

Besides privacy, the compression efficiency of feature coding schemes should also be studied since one of our main objectives is reducing the bitrate to the extent that justifies using feature compression instead of input compression. Like images, features derived from intermediate DNN layers can be encoded either using traditional or DNN-based codecs. Early works on this topic [18, 30, 4] preferred traditional codecs; such approaches are still appealing due to traditional codecs' computational simplicity relative to DNN-based codecs, and their wide availability in the existing cameras and devices. In order to use a conventional codec for feature coding, the feature tensor usually needs to be tiled into an image, scaled, and pre-quantized [18]. The authors of [30] additionally reduced the dimensionality of the latent space in terms of both the number of channels and spatial resolution using an autoencoder prior to coding the bottleneck tensors via JPEG. Such dimensionality reduction usually helps compression efficiency.

The abovementioned schemes replaced the utilized traditional codec with an identity function in the backpropagation phase of the training. In other words, the effect of compression/decompression and the bitrate of the generated bitstream has not been taken into account in their training processes. On the other hand, more recent schemes [52, 104] employ DNN-based coding tools, especially advanced entropy models, to code features. An advantage of such schemes is their ability to be trained end-to-end, with a loss function that combines a differentiable rate estimate and task accuracy. In such end-to-end approaches, the rate estimate is usually done by measuring an approximate entropy of the bottleneck's elements whose distribution is assumed to be Gaussian [8].

Although end-to-end trainable coding pipelines are more flexible, the downside is increased complexity and the fact that their coding engines are not widely available in existing devices. For that reason, some research works are still being conducted to provide proper differentiable proxies for the hand-crafted codecs and their associated bitrates [36, 79, 37]. This way, standard-codec-based approaches can be trained in an end-to-end manner as well. The authors of [36, 37] have designed a differentiable proxy for JPEG and claimed that it suffices to represent more complex codecs such as HEIC. Their main objective is to deploy a DNN-based pre-processor and post-processor together with a standard codec for the purpose

of transporting high-resolution (super-resolution) or High Dynamic Range (HDR) images. In their framework, the standard codec is substituted with a quantizer proxy, an 8×8 Discrete Cosine Transform (DCT), and an 8×8 Inverse Discrete Cosine Transform (IDCT). The corresponding rate is also estimated via a smooth logarithmic differentiable function computed on the DCT coefficients. Reference [79] attempts to approximate the bitrate of an HEVC encoder for the same scenario of exploiting neural pre-processing and post-processing modules with that. It combines the rate estimation idea of the end-to-end trainable hyper-prior model [9] with some heuristic approximation of the existing modules in HEVC to create a differentiable rate proxy.

The mentioned proxies have not considered the effect of prediction in conventional codecs. Prediction is the foundation of all the hand-crafted codecs through which a huge portion of the input redundancies is removed. The authors of [4] have provided a rate proxy assuming a simple model to imitate the effect of prediction. In their proposed rate estimation, spatial prediction is modeled as a simple averaging over the top and left neighbors of each pixel. This approach is less complicated compared to the abovementioned estimators. We have also employed the same proxy in our training loss functions to encourage more compressibility of the bottleneck features with a traditional codec like H.266/VVC. More detail is presented in Section 2.3.3.

2.3 Proposed Method

In our proposed method, we essentially follow two key objectives: 1) resistance to model inversion attack and 2) good compression efficiency. In order to achieve these goals, we train an autoencoder, inserted in the middle of a machine vision model, together with an auxiliary network trying to reconstruct the input in an adversarial manner [35]. The overall block diagram of our design is shown in Fig. 2.2. In this chapter, we chose object detection by YOLOv5 [48], although our scheme is applicable to other DNN models and tasks as well. The overall architecture of YOLOv5 is depicted in Fig. 2.1.

YOLOv5 was released by a company named Ultralytics in 2020. There has not been a published paper for this model yet, but its code is available in a GitHub repository[1] by Glenn Jocher, Founder and CEO of Ultralytics. This model is quite similar to YOLOv4 [14]. YOLOv5 has been developed in the Pytorch framework as opposed to official versions of YOLOv3 or YOLOv4. YOLOv5 has a variety of models with varying complexity and accuracy. We selected YOLOv5m that has been trained on images of the COCO dataset [55], resized to the width (or height) of 640 with preserved aspect ratio.

[1] https://github.com/ultralytics/yolov5

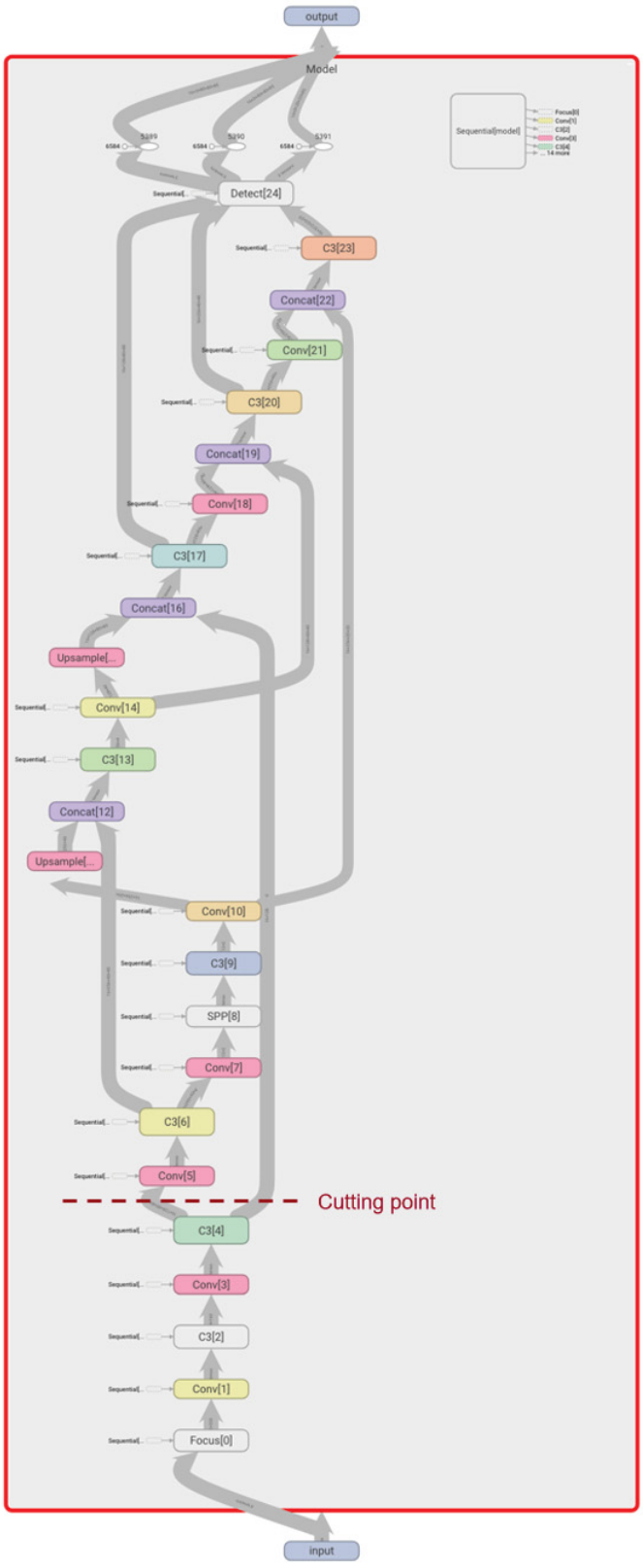


Figure 2.1: The overall architecture of YOLOv5 with the selected split point

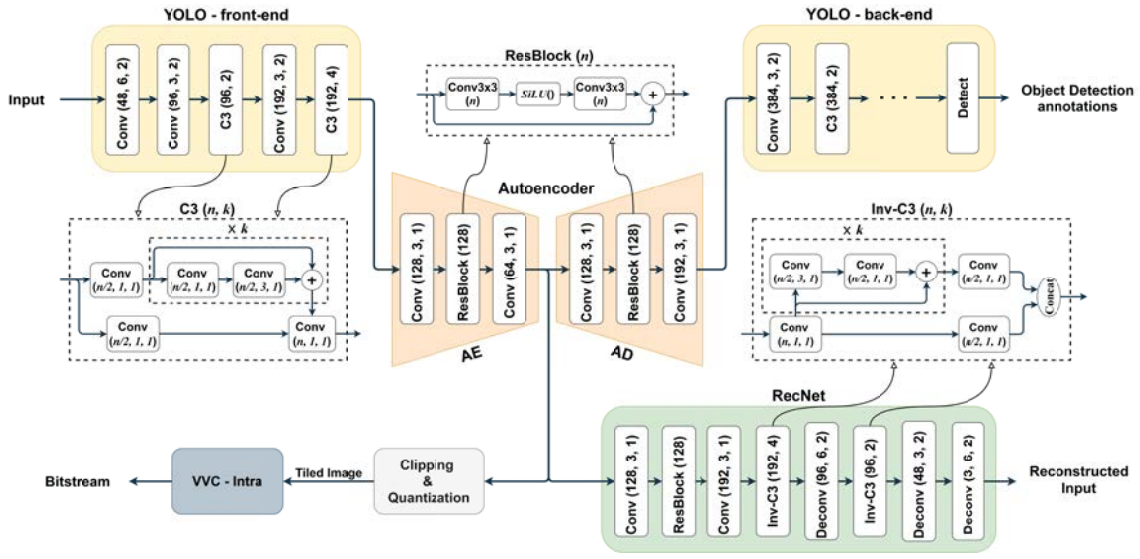


Figure 2.2: The overall block diagram of the proposed method. $Conv(n, k, s)$ is a 2D Convolution layer followed by a Batch normalization layer and a $SiLU()$ activation, with n being the number of output channels, a kernel size of $k \times k$, and stride=s. $Deconv(n, k, s)$ is the same as $Conv(n, k, s)$ except with a Convolution Transpose layer. Note that the $Conv$ layers in the autoencoder do not have batch normalization, and the last $Conv$ in AE does not have the $SiLU()$ activation either. $Conv3 \times 3(n)$ is a single 3×3 Convolution layer with stride=1 and n output channels. "$\times k$" indicates k times repetition of the blocks specified by a dashed frame.

2.3.1 Split Point

Choosing a split point is a design issue [49, 29], which depends on energy considerations, computational resources at the edge, the type of connection between the edge and the cloud, and so on. Here, we should also consider privacy as a determinant factor in choosing an appropriate split point. In [39], it is practically shown that the accuracy of the input recovery from the deeper feature tensors is lower. In the following, we try to argue this claim based on information-theoretic considerations [6].

Let $\mathbf{X}$ be the input image and $\boldsymbol{\mathcal{Y}}_i$ be the feature tensor at the i-th layer. Since $\mathbf{X} \to \boldsymbol{\mathcal{Y}}_i \to \boldsymbol{\mathcal{Y}}_{i+1}$ forms a Markov chain, by the data processing inequality [25] we have that

$$I(\mathbf{X}; \boldsymbol{\mathcal{Y}}_i) \geq I(\mathbf{X}; \boldsymbol{\mathcal{Y}}_{i+1}), \tag{2.2}$$

where $I(\cdot; \cdot)$ is the mutual information. Hence, deeper layers carry less information about the input and are, therefore, more resilient to model inversion attacks. Also, as shown in [21], deeper layers are more compressible. These arguments would suggest choosing a split point as deep as possible.

On the other hand, limited computation and energy resources on the edge device favor selecting a shallower split point. Moreover, the YOLOv5m model branches out at layer 5

(see Fig. 2.1), meaning that if we select the split point after layer 5, we would have to encode and transmit multiple feature tensors, which would increase both the complexity and the total bitrate. Hence, we choose to split the YOLOv5m model at layer 5.

2.3.2 Autoencoder

At the split point, we insert an autoencoder, aiming at reducing the dimensionality of the original latent space and creating features with improved compressibility and resistance to model inversion attacks. This is a plug-and-play strategy and can be used for other models and tasks as well. The autoencoder's architecture is shown in Fig. 2.2: its encoder portion is referred to as AE and decoder as AD.

AE is supposed to transform the YOLO's native latent space at the split point (with 192 channels) into a lower-dimensional space called the bottleneck (with 64 channels). Besides, AE is responsible for removing private information and preserving the required information for the object detection task. AD consequently tries to provide the back-end with desirable tensors having the same dimensionality as the native latent space. Note that the spatial dimensions of the tensors remain unchanged in AE to preserve the spatial precision of subsequent object detection.

In the end, the resulting bottleneck feature tensor is tiled, pre-quantized to 8-bits per element, and encoded using Versatile Video Coding (VVC)-Intra [16] via VVenC [97] video coded software with the *lowdelay-faster* preset. At the cloud side, the encoded bitstream is decoded by a VVC decoder and AD, then fed to the YOLOv5m back-end to get the inference results.

2.3.3 Loss functions

To train our network, we use three main loss functions that will be described in the following:

- **Object Detection:** This is the native object detection loss function utilized in YOLOv5 [48] and computed as follows.

$$L_{obj\_det} = w_{obj} \cdot L_{obj} + w_{box} \cdot L_{box} + w_{cls} \cdot L_{cls} \tag{2.3}$$

  where L_{obj}, L_{box}, and L_{cls} are objectness, bounding box localization, and classification losses that are combined with respective weights of w_{obj}, w_{box}, and w_{cls}.

- **Compressibility:** L_{cmprs} is a loss function borrowed from [4] and encourages compressibility of the tensors with traditional image codecs. It attempts to emulate the intra-prediction and transformation processes, which are the foundation of hand-crafted codecs. Intra-prediction is performed based on the neighboring pixels, and transformation is applied to the residuals (the difference between the original and predicted pixels). In the end, the transformed coefficients are quantized and encoded

into the bitstream. The authors of [4] introduced a 2-D Differential Pulse Code Modulation (DPCM)-based matrix that computes the average difference between the current pixel and its upper and left neighbors. The resulting values are somewhat equivalent to the residuals obtained in the traditional codecs. Then, a DCT is performed, and the ℓ_1-norm of its coefficients forms the compressibility loss, L_{cmprs}.

- **Reconstruction:** This loss function (L_{rec}) is for reconstructing the input image from the intermediate tensors. L_{rec} itself consists of two terms. One is the typical ℓ_1-norm of the error, and the other is to emphasize the edges since private information is usually associated with fine details. L_{rec} is computed as follows:

$$L_{rec} = \frac{1}{n}\|\mathbf{X} - \widehat{\mathbf{X}}\|_1 + \frac{\beta}{n}\|S_x * \mathbf{X} - S_x * \widehat{\mathbf{X}}\|_1 + \frac{\beta}{n}\|S_y * \mathbf{X} - S_y * \widehat{\mathbf{X}}\|_1, \tag{2.4}$$

where $\mathbf{X}$ and $\widehat{\mathbf{X}}$ are the original and reconstructed images, $\|\cdot\|_1$ is the ℓ_1-norm, S_x and S_y are the horizontal and vertical Sobel filters, respectively, $*$ is the convolution operator, and n is the total number of tensor's elements in the batch. The value of β is empirically set to 5.

The kernels of the horizontal and vertical Sobel filters are shown in Fig. 2.3.

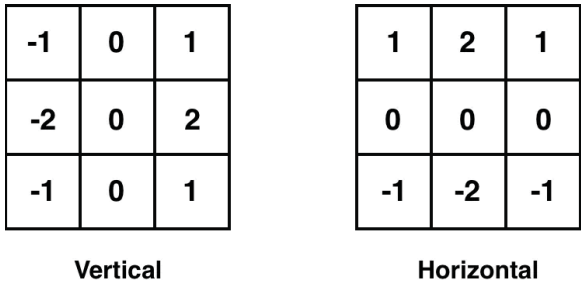


Figure 2.3: The kernels of horizontal and vertical Sobel filters

2.3.4 Adversarial Training

We have adopted a specific training strategy to generate compressible bottleneck features with resilience towards model inversion attacks. This is done through an auxiliary DNN model, called Reconstruction Network (RecNet), whose purpose is to reconstruct the input image from bottleneck features. As depicted in Fig. 2.2, the architecture of RecNet is roughly a mirror of the YOLOv5m front-end and AE. We adversarially train the autoencoder and RecNet together. During the training process, RecNet tries to recover the input image from the bottleneck features as best it can, while the AE simultaneously tries to disrupt RecNet's performance by manipulating the generated bottleneck features. At the same time, both AE and AD attempt to keep object detection accuracy high. Besides, the bottleneck features become more compressible over the training stage through the compressibility loss function.

Note that the pre-trained YOLOv5m model is kept intact, and its weights are frozen during the entire training process. The training process for each batch of data is summarized in the following steps:

1. The input images $\mathbf{X}$ go through the front-end, AE, and RecNet, and the respective reconstruction loss is computed according to (2.4).

2. Gradient of L_{rec} backpropagates only through the RecNet and updates its weights. The autoencoder's weights are frozen at this step.

3. The same batch of images goes through the whole network (YOLO front-end, AE, AD, YOLO back-end), and the total loss is computed as follows:

$$L_{tot} = L_{obj\_det} + w_{cmprs} \cdot L_{cmprs} - w_{rec} \cdot L_{rec}, \tag{2.5}$$

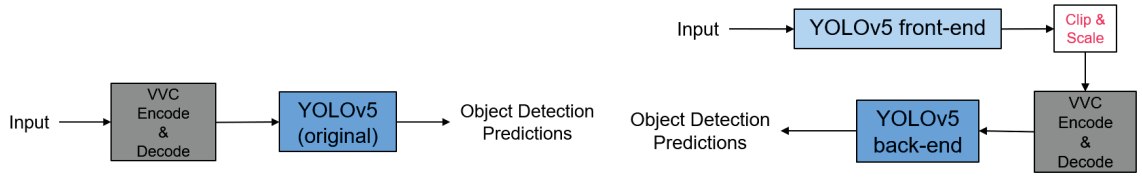
   where L_{obj_det}, L_{cmprs}, and L_{rec} are the loss terms defined in section 2.3.3. w_{cmprs} and w_{rec} are the balancing weights for combining these loss terms.

4. Gradient of L_{tot} backpropagates only through the autoencoder and updates its weights. RecNet's weights are frozen in this step. Note that the negative sign of the L_{rec} in (2.5) leads AE to make reconstruction more difficult for RecNet. Meanwhile, the positive sign of L_{obj_det} and L_{cmprs} cause AE and AD to improve object detection accuracy and increase the compressibility of the feature channels with traditional image codecs.

2.4 Experimental Results

We trained our networks on the COCO-2017 object detection dataset [55] using an NVIDIA Tesla V100-SXM2 GPU with 32GB memory. In the first step, only the autoencoder got trained with L_{obj_det} (2.3) for 50 epochs. Next, we trained only the RecNet with L_{rec} (2.4) for 20 epochs, with the autoencoder's weights frozen to those obtained in the first step. Finally, with the autoencoder and RecNet initialized to the previously obtained weights, the adversarial training was conducted as described in Section 2.3.4 for 40 epochs. In all the steps, we used Stochastic Gradient Descent (SGD) optimizer with the initial learning rate equal to 0.01, changing by a cosine learning rate decay [38], which is claimed to be better than common exponential step decay methods, over the training.

The results of our proposed method are compared with two benchmarks. "Anchor-input" refers to the case where the input image is encoded by VVC, and the decoded image is passed to the original object detection YOLOv5m model to get the inference results, as shown in Fig. 2.4a. "Anchor-latent" corresponds to the case where the original YOLOv5m latent space at layer 5 (without the AE) is clipped, scaled, encoded by VVC, and the decoded tensor feeds the original YOLOv5m back-end, as depicted in Fig. 2.4b.



(a) Anchor-input (b) Anchor-latent

Figure 2.4: The benchmark pipelines

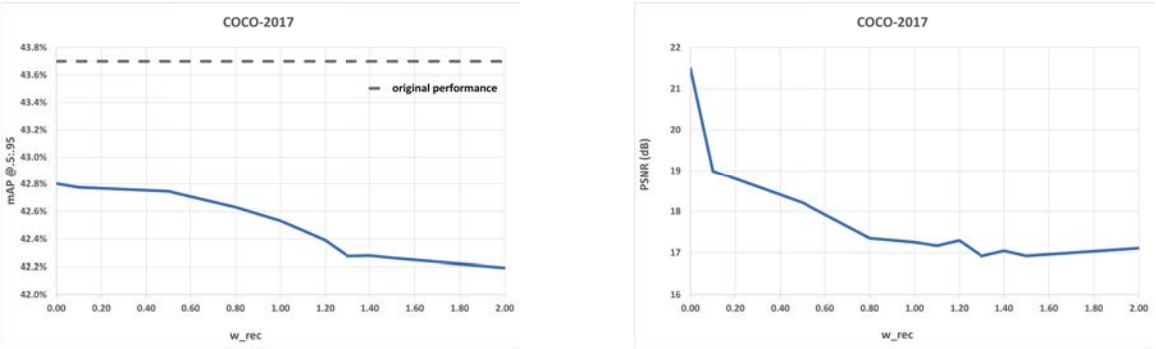
All the encoding and decoding processes take place using VVC-Intra. Prior to that, the channels in the bottleneck (or latent space) are clipped, quantized to 8 bits, and tiled into an 8×8 matrix for the bottleneck (or 12×16 matrix for the Anchor-latent) to create a gray-scale image. On the decoder side, the bitstream is decoded, and the resulting tensors are passed to the rest of the model for inference. As noted in [24], feature compression performance can be improved via clipping. We empirically found 6σ the best clipping range, where σ is the standard deviation of the to-be-coded tensors on the COCO validation set.

2.4.1 Resistance to model inversion attack

As mentioned before, RecNet is an auxiliary DNN model exploited in the adversarial training stage. So, it is not part of the final pipeline. In a real situation, however, if an adversary can get hold of the edge device in a black-box setting, they can try to train their own input reconstruction model using input-bottleneck pairs. In order to test our model against this attack, we train a new, randomly initialized RecNet on the bottleneck features generated by the autoencoder obtained in the adversarial training stage. This RecNet is trained with a ℓ_1-norm loss (the first term in (2.4)) as it has been claimed in [106] that ℓ_1-norm usually provides better quality for human vision compared to ℓ_2-norm. All the following results are based on the output of the mentioned RecNet. In addition to the main pipeline, we have also trained another RecNet, whose first three layers are removed, on the original YOLOv5m latent space at layer 5. This RecNet is associated with the Anchor-latent.

At first, we trained a number of autoencoders with $w_{cmprs} = 0$ and different values of w_{rec} to see the effect of reconstruction loss (L_{rec}) on the models' performance without doing VVC compression. Fig. 2.5a shows the mAP@.5:.95 (mean Average Precision over the IoU thresholds of .5 to .95) curve versus w_{rec} on the COCO-2017 validation set. The average Peak Signal to Noise Ration (PSNR) of the recovered input images using the adversary's RecNet models versus w_{rec} is also shown in Fig. 2.5b. As expected, increasing the value of w_{rec} makes the object detection model less accurate, although the mean Average Precision (mAP) drop is negligible. Conversely, w_{rec} can significantly affect the quality of the recovered input image in the model inversion attack. As seen, the PSNR drop is approximately 4.5 dB when w_{rec} changes from 0.0 to 2.0

Some visual examples are also given in Fig. 2.6. We can deduce from these images that the adversary's RecNet models are able to reconstruct the input at a relatively good quality from the YOLOv5m's original latent space at layer 5 and also from the bottleneck of the autoencoder trained with $w_{rec} = 0.0$. But, the quality of the recovered input drops noticeably by increasing the value of w_{rec}, especially near the edges and textured areas due to using the Sobel filter in the reconstruction loss term as presented in equation (2.4). This edge-concentrated distortion makes the faces hard to recognize and the text hard to read, which is favorable in terms of privacy.



(a) w_{rec} vs mAP@.5:.95 (b) w_{rec} vs PSNR

Figure 2.5: Model performance in terms of object detection accuracy and resistance against model inversion attack for different values of w_{rec}

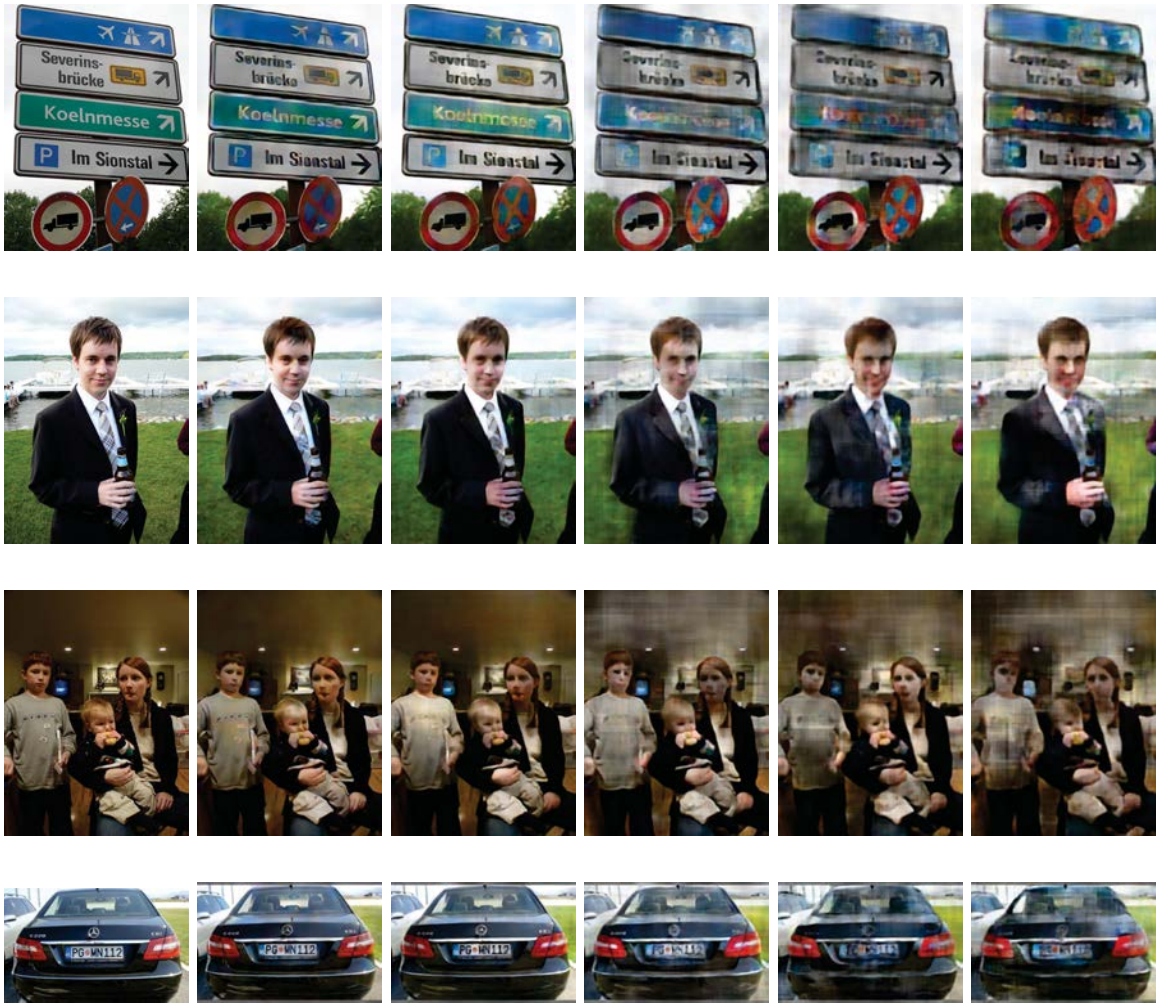
2.4.2 Feature Compression Results

The compression efficiency of the system was examined by plotting the rate-accuracy curves, where the rate is computed as the number of bits per pixel, and the accuracy is mAP at the IoU threshold of 0.5. The points of a curve were obtained by changing the value of Quantization Parameter (QP) in the VVC compression process. Fig. 2.7a shows the rate-accuracy curves for different models and various values of w_{rec} and w_{cmprs}. Only a limited number of (w_{rec}, w_{cmprs}) pairs are included in Fig. 2.7a such as not to make the chart unclear.

We have also plotted another set of curves, called quality-accuracy, that could represent privacy efficiency. These curves are depicted in Fig. 2.7b, where the x-axis is the average PSNR of the recovered input images, and the y-axis is the mAP of the object detection model on the COCO-2017 validation set. The points lying on the top-left part of the chart are better since high mAP and low PSNR are desirable in our privacy-preserving application.

2.4.3 Best Weights

The most important hyperparameters in our DNN model are w_{rec} and w_{cmprs} through which one can adjust the bitrate of the coded bitstream and the amount of existing private

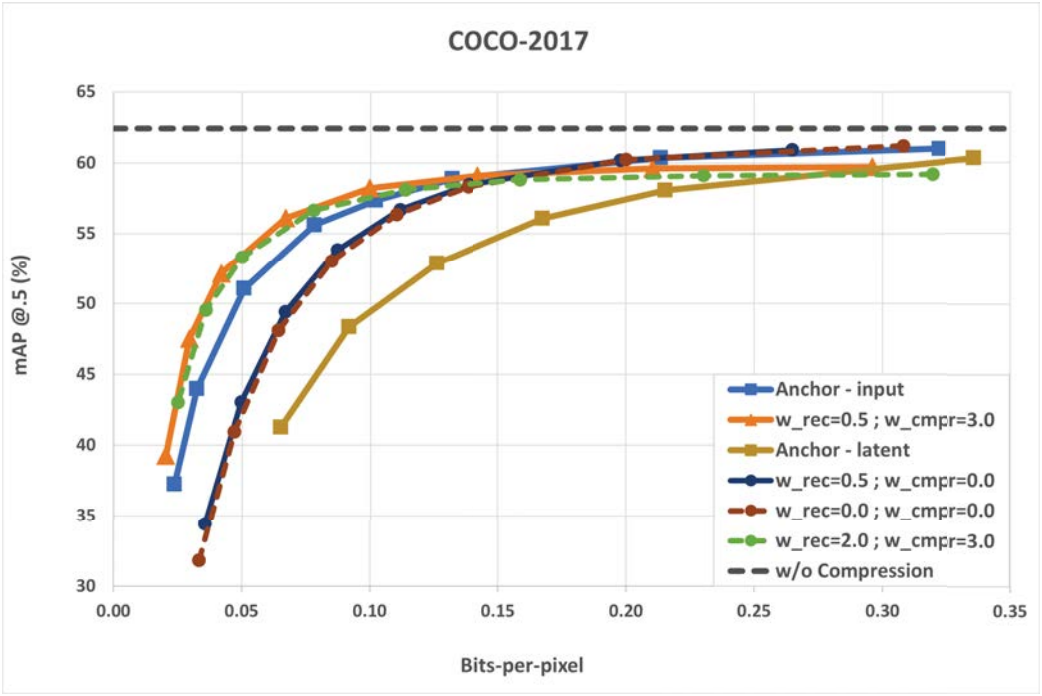


(a) (b) (c) (d) (e) (f)

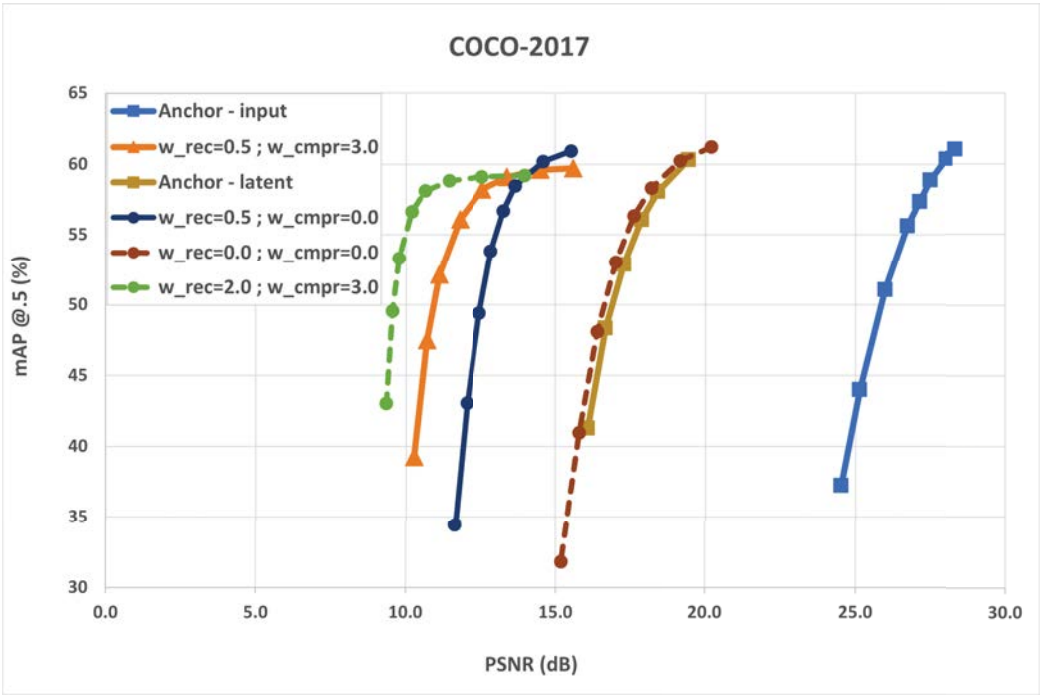
Figure 2.6: Visual examples of input reconstruction in model inversion attack (a) Original (b) Anchor-latent (c) $w_{rec} = 0.0$ (d) $w_{rec} = 0.5$ (e) $w_{rec} = 1.0$ (f) $w_{rec} = 2.0$

information in the bottleneck. To obtain suitable weights, we ran a 2-step grid search on the $\mathbf{W} = (w_{rec}, w_{cmprs})$ space. In the first step, 16 different points with $w_{rec} \in \{0.5, 1, 1.5, 2\}$ and $w_{cmprs} \in \{1, 2, 3, 4\}$ were tested. In order to quantitatively compare their performance Bjøntegaard-Delta values [13] between their associated curves[2], similar to those presented in Fig. 2.7, are computed. The first two columns of Table 2.1 are BD-Rate and BD-mAP between the rate-accuracy curves, and the third is BD-PSNR between the quality-accuracy curves. Note that the accuracy metric is mAP@.5:.95, and the Bjøntegaard-Delta values are computed with respect to "Anchor-input".

[2]Calculated based on the MPEG-VCM reporting template [41]



(a) Rate vs mAP@.5



(b) PSNR vs mAP@.5

Figure 2.7: Compression and privacy efficiency curves of the proposed and benchmark models

Table 2.1: Bjøntegaard-Delta values with respect to "Anchor-input" based on mAP@.5:.95 (first step of the grid search)

	BD-Rate	**BD-mAP**	**BD-PSNR**
$w_{rec} = 0.5\ ;\ w_{cmprs} = 1.0$	-7.1%	0.4	-13.87 dB
$w_{rec} = 0.5\ ;\ w_{cmprs} = 2.0$	-12.9%	0.7	-13.59 dB
$w_{rec} = 0.5\ ;\ w_{cmprs} = 3.0$	-17.4%	0.9	-14.70 dB
$w_{rec} = 0.5\ ;\ w_{cmprs} = 4.0$	-14.0%	0.5	-12.85 dB
$w_{rec} = 1.0\ ;\ w_{cmprs} = 1.0$	-1.3%	0.0	-14.32 dB
$w_{rec} = 1.0\ ;\ w_{cmprs} = 2.0$	-7.3%	0.1	-15.31 dB
$w_{rec} = 1.0\ ;\ w_{cmprs} = 3.0$	-11.8%	0.4	-16.20 dB
$w_{rec} = 1.0\ ;\ w_{cmprs} = 4.0$	-10.3%	0.1	-16.01 dB
$w_{rec} = 1.5\ ;\ w_{cmprs} = 1.0$	-2.2%	0.0	-15.04 dB
$w_{rec} = 1.5\ ;\ w_{cmprs} = 2.0$	-11.0%	0.4	-15.53 dB
$w_{rec} = 1.5\ ;\ w_{cmprs} = 3.0$	-10.3%	0.2	-16.27 dB
$w_{rec} = 1.5\ ;\ w_{cmprs} = 4.0$	-11.7%	0.2	-17.26 dB
$w_{rec} = 2.0\ ;\ w_{cmprs} = 1.0$	-0.3%	-0.2	-14.93 dB
$w_{rec} = 2.0\ ;\ w_{cmprs} = 2.0$	-8.4%	0.2	-15.71 dB
$w_{rec} = 2.0\ ;\ w_{cmprs} = 3.0$	-11.2%	0.3	-16.19 dB
$w_{rec} = 2.0\ ;\ w_{cmprs} = 4.0$	-13.0%	0.3	-16.17 dB

According to these results, $\mathbf{W}_0 = (0.5, 3.0)$ provides the best compression efficiency and $\mathbf{W}_1 = (1.5, 4.0)$ shows the best resistance to model inversion attack. However, the quality of the recovered input from the model trained with $\mathbf{W}_0$ is 14.70 dB below the Anchor-input, which is quite significant. Looking at the visual examples of Fig. 2.6, it can be seen that the text, licence plate, and faces are not recognizable even for $w_{rec} = 0.5$. Thus, $\mathbf{W}_0$ is also acceptable in terms of privacy performance. We select $\mathbf{W}_0$ as the best weight and perform a finer search around it in the second step.

In the second step, nine different points with finer precision around $\mathbf{W}_0 = (0.5, 3.0)$ were examined, where $w_{rec} \in \{0.25, 0.5, 0.75\}$ and $w_{cmprs} \in \{2.5, 3.0, 3.5\}$. The corresponding Bjøntegaard-Delta values are given in Table 2.2. As seen, $\mathbf{W}_0 = (0.5, 3.0)$ still has the best compression efficiency among all the tested weights. Since its privacy performance is acceptable as well, we choose $\mathbf{W}_0 = (0.5, 3.0)$ and its respective model as the best.

2.4.4 Discussion

We call the model trained with the weights of $\mathbf{W}_0 = (0.5, 3.0)$, "Proposed" model. The rate-accuracy and quality-accuracy curves of the Proposed model are shown in Fig. 2.7 with orange color. The Bjøntegaard-Delta values between the curves, presented in Fig. 2.7, are also given in Table 2.3. Note that in this table, the anchor is our Proposed model based on which all the values are computed.

Table 2.2: Bjøntegaard-Delta values with respect to "Anchor-input" based on mAP@.5:.95 (second step of the grid search)

	BD-Rate	**BD-mAP**	**BD-PSNR**
$w_{rec} = 0.25$; $w_{cmprs} = 2.5$	-13.9%	0.7	-14.82 dB
$w_{rec} = 0.25$; $w_{cmprs} = 3.0$	-13.1%	0.5	-13.95 dB
$w_{rec} = 0.25$; $w_{cmprs} = 3.5$	-16.9%	0.7	-11.93 dB
$w_{rec} = 0.50$; $w_{cmprs} = 2.5$	-13.8%	0.7	-15.06 dB
$w_{rec} = 0.50$; $w_{cmprs} = 3.0$	-17.4%	0.9	-14.70 dB
$w_{rec} = 0.50$; $w_{cmprs} = 3.5$	-8.9%	0.1	-13.90 dB
$w_{rec} = 0.75$; $w_{cmprs} = 2.5$	-15.3%	0.7	-15.45 dB
$w_{rec} = 0.75$; $w_{cmprs} = 3.0$	-14.5%	0.5	-15.10 dB
$w_{rec} = 0.75$; $w_{cmprs} = 3.5$	-15.0%	0.5	-16.32 dB

These results show that our proposed method is able to reduce the bitrate by almost 21% and 153% on average compared to Anchor-input and Anchor-latent, respectively. The corresponding mAP value gain for the same anchors is also 0.9 and 4.2. Comparing the Anchor-latent with $(w_{rec}, w_{cmprs}) = (0.0, 0.0)$, we notice that the autoencoder itself drops the reconstruction quality by only $5.80 - 5.55 = 0.25$ dB and reduce the bitrate by $153.3 - 80.7 = 72.6\%$ (relative to the bitrate of "Proposed"). Thus, part of the bitrate reduction is due to the autoencoder's dimensionality reduction. However, L_{cmprs} plays the main role in bitrate reduction since $(w_{rec}, w_{cmprs}) = (0.5, 3.0)$ (Proposed) has 76.5% lower bitrate compared to $(w_{rec}, w_{cmprs}) = (0.5, 0.0)$. Also, increasing w_{rec} from 0.0 to 0.5 (when $w_{cmprs} = 0$) causes a PSNR drop of $5.55 - 1.44 = 4.11$ dB, which proves the efficacy of L_{rec} and the adversarial training scheme. Despite that, higher values of w_{rec} are not significantly effective. For instance, $(w_{rec}, w_{cmprs}) = (2.0, 3.0)$ has 4 times larger w_{rec} relative to Proposed, which reduces the PSNR by 1.41 dB. This phenomenon can be also seen in Fig. 2.5b.

The major problem of high values of (w_{rec} and w_{cmprs}) is the object detection performance at the high bitrates, as shown in Fig. 2.7a. For instance, $(w_{rec}, w_{cmprs}) = (2.0, 3.0)$ has the highest values of (w_{rec} and w_{cmprs}) and consequently, the worst performance on the high-end. Nonetheless, this mAP drop at the high bitrates is negligible, especially for our Proposed model with the selected weights.

2.5 Conclusion

In this chapter, we presented a novel feature coding scheme for machines with improved resistance to model inversion attacks. The features of the YOLOv5m latent space were transformed into a lower-dimensional space using an autoencoder, which was trained in an adversarial manner to reduce the ability for input reconstruction while maintaining object-detection accuracy. Visual and quantitative results showed that our method is able

Table 2.3: Bjøntegaard-Delta values with respect to "Proposed" based on mAP@.5:.95

	BD-Rate	**BD-mAP**	**BD-PSNR**
Anchor-input	21.0%	−0.9	14.70 dB
Anchor-latent	153.3%	−4.2	5.80 dB
$w_{rec} = 0.5\,;\ w_{cmprs} = 0.0$	76.5%	−3.3	1.44 dB
$w_{rec} = 0.0\,;\ w_{cmprs} = 0.0$	80.7%	−3.5	5.55 dB
$w_{rec} = 2.0\,;\ w_{cmprs} = 3.0$	7.9%	−0.5	−1.41 dB

to degrade the quality of the recovered image using a model inversion attack, especially near the edges. The PSNR drop between the cases, where the model inversion attack is applied to the original latent space of YOLOv5m at layer 5 and the bottleneck of the adversarially trained autoencoder, is 5.8 dB. Meanwhile, coding the features produced by our autoencoder resulted in more than 20% bit savings, on average, at the same object detection accuracy compared to coding the input image, and more than 153% compared to coding the original YOLOv5m features. Therefore, our proposed method provides both privacy and a desirable compression efficiency at the same time for an object detection model.

Chapter 3

Motion in Video Coding for Machines

In this chapter, we attempt to propose a method for efficient video coding in the latent space of a machine vision model. First, an introduction to traditional video codecs and their central core, motion estimation and compensation, is given in Section 3.1, where we also discuss the challenges that exist in ME and MC in the latent space. Then, a literature review is presented in Section 3.2. Before proposing our DNN-based ME and MC modules in Section 3.4, a study of motion in the latent space and its relationship to the input-space motion is conducted in Section 3.3.

3.1 Introduction

Video signal is currently the largest source of data consumed globally. The amount of data existing in a video is far more than in other signals like image, audio, etc. Therefore, the need to efficiently compress video signals has always been vital, and consequently, video codecs have become more complicated. However, these codecs are designed for human use and may not be as efficient for machines. As the amount of data being used by machines is rapidly increasing, the need for an efficient video coding scheme optimized for machines should accordingly be acknowledged.

3.1.1 Traditional Video Codecs

Different video coding standards have been introduced since the early 1990s, in parallel with increasing popularity of digital videos having various applications in entertainment, archiving, surveillance, etc. The two prominent standardization bodies in this area are ITU[1]

[1]International Telecommunications Union

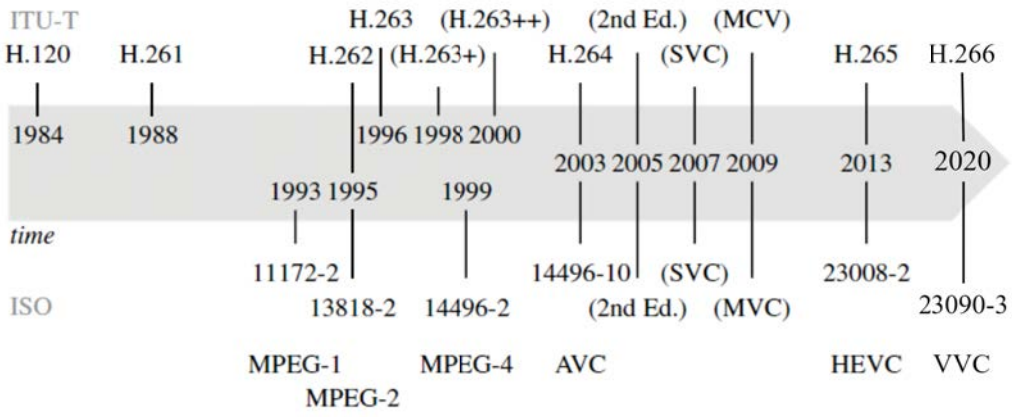


Figure 3.1: Timeline of video coding evolution [99]

and ISO/IEC[2]. An overview of the timeline for the major standards proposed by these two standardization bodies is shown in Fig. 3.1.

The high volume of video distribution over the Internet made a significant need for improving compression efficiency. This demand might be one of the reasons for the Joint Video Team (JVT) of both organizations being formed. Their first collaboration led to the development of H.264/AVC [3] [98] standard in 2003. This team continued its activity under a new name, Joint Collaborative Team on Video Coding (JCT-VC), to develop future standards. They introduced H.265/HEVC [87] in 2013, and their most recent efforts resulted in finalizing the first edition of H.266/VVC [16] in August 2020. The second edition was released in April 2022. The mentioned standards have been successful and are widely used due to their high compression efficiency. Each of these standards outperforms its predecessors by saving roughly 50% bitrate at the same quality.

The main goal of all these video coding standards is to reduce the bitrate of a video as much as possible by removing the redundancies. There are two types of redundancy in a given video. One is between the pixels within a frame, referred to as spatial redundancy, and the other is between the pixels of different frames, called temporal redundancy. Video compression algorithms try to predict the pixel values inside a video by exploiting these redundancies and correlations. Intra-prediction and inter-prediction refer to the predictions used to remove spatial and temporal redundancies, respectively. After prediction, the encoder obtains the residuals, i.e., the difference between the original and predicted values, applies a DCT-type transform to them, quantizes the coefficients, and runs the entropy coding on the results. The general block diagram of a video encoder is depicted in Fig 3.2. The temporal correlations between video frames are considerably higher than the spatial

[2]International Standardization Organization/International Electrotechnical Commission

[3]Advanced Video Coding

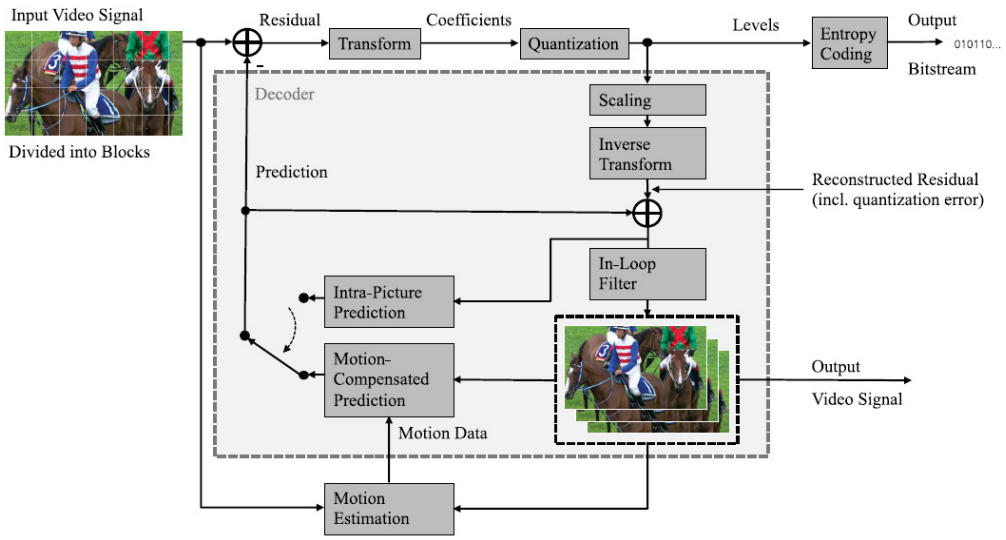


Figure 3.2: Block diagram of a video encoder [15]

correlations between the pixels of a single frame. As a result, Inter-prediction plays a more critical role in video compression, which is the focus of this research as well.

The key component of inter-prediction is motion estimation, in which the encoder should find the best match for a block of the to-be-coded frame in the reference frame. The displacement between the current Prediction Unit (PU) and the selected block in the reference frame is called Motion Vector (MV). The encoder considers the selected block in the reference frame as a prediction for the current PU and codes only the residuals. This procedure is called motion compensation. In the end, the encoder signals the MVs and the quantized transformed residuals, through which the decoder can reconstruct the input video.

Inter-prediction could be done in two ways regarding the number of reference frames. Uni-prediction is the case where only one reference frame is used for the ME process of a PU. In contrast, Bi-prediction uses two reference frames, and the two selected blocks of the references are combined in a specific way to form the final prediction for the current PU. In Bi-Prediction, both reference frames are obviously before the current frame in the decoding order but not necessarily in the displaying order.

The video frames are categorized into three different types based on the predictions used for their PUs. In I-frames, all the PUs are coded via intra-prediction, while P-frames comprise both intra-prediction and Uni-prediction blocks. B-frames are like P-frames with at least one Bi-predicted PU.

3.1.2 ME Challenges in the Latent Space

As mentioned in Section 1.2, video coding in the latent space of a machine vision model has some challenges due to the difficulty of ME and MC in the feature channels. These challenges

can cause compression inefficiency in the case of using traditional ME approaches. Some of these challenges are listed below:

1. **Low resolution of the channels:** As we go deeper into convolutional DNN models, the resolution of the feature channels typically decreases. The pooling layers in these models are employed to reduce the spatial dimension and increase the number of channels. Therefore, the resolution of the channels in the cutting layer of DNN models, where the channels should be coded, is usually a fraction of the original resolution. In traditional codecs such as H.264/AVC and H.265/HEVC, ME is performed with a quarter-pel precision, i.e., the MVs could be fractional numbers with 1/4 pixel precision. However, the ME precision has increased to 1/16-pel in H.266/VVC. Since these codecs are designed to apply to the input-space videos with high resolutions, using the same precision in the latent space does not seem to work as efficiently.

2. **Noise-like channels:** The resolution reduction in the deep layers of DNN models contributes to the coarseness of feature channels. However, the size is not the only contributing factor. In addition, the deeper layers in DNN models tend to carry more abstract and task-relevant features of the input, while the shallower layers contain a more general and smoother representation of the input images. In other words, some deeper feature channels look like noisy images. Hence, the dependence between the neighboring pixels in the deep layers is smaller in comparison with the input space. In fact, the fluctuation across the pixels in the feature channels is far more than that in a natural input image. These fluctuations disrupt the utility of interpolation filters aimed at providing proper pixel values in the fractional positions. As a result, high-precision ME may not be suitable in the feature channels of DNN's deep layers.

3. **Location-dependant representation in the latent space:** In convolutional DNN models, the value of a component in a feature channel is affected by the neighboring pixels when a convolution layer with a kernel size bigger than 1×1 is employed. The set of pixels of the input image contributing to the value of a channel's component is called the *receptive field*. The receptive field typically gets larger for the deeper layers since the number of prior convolution layers increases. Accordingly, a single object in the input image does not have a unique representation in different locations of a specific feature channel. This phenomenon could disrupt the ME in the latent space since the pixel values of a single object change as it moves.

4. **Difference in motions of different channels:** As it is well-known, the DNN models are essentially nonlinear functions of the input. Some of these nonlinear layers, like max-pooling, could cause the same motion of the input to emerge slightly differently in various channels. Therefore, considering the same motion for all the channels is not reasonable. Even using similar interpolation filters might not be effective.

All the points mentioned above suggest that traditional video codecs may not be a good fit in the latent space. In Section 3.3, however, we try to prove this claim experimentally.

3.2 Related Work

Research on video coding in the latent space is fairly scarce in the literature. Most of the studies focus on optimizing the coding performance for humans, and the machine-targeted schemes are not taken into account thoroughly. However, the ME and MC algorithms that are being used in the human-targeted codecs could probably apply to machine-targeted codecs as well. Therefore, we review some of these methods in this section.

Before the current wave of interest in DNN-based image/video codecs, most of the studies had focused on optimizing the ME and MC tasks of the conventional video codecs, especially in terms of complexity. According to [80], most of the execution time of the HM [46], an H.265/HEVC encoder, is spent on motion estimation. The encoder takes almost 96% of the overall execution time for motion estimation in the Full Search algorithm. This percentage is roughly 70% and still high when using a more practical search algorithm like TZ Search.

To perform a good inter-prediction, efficient partitioning is needed in the first place. Different parts of a frame having independent motions should lie in separate partitions. The best way of partitioning a Coding Tree Unit (CTU) is, indeed, the exhaustive search, where all the possible partitions are executed by running the full Rate-Distortion Optimization (RDO) process. However, such a method is not feasible due to the huge number of partitioning modes and the time-consuming process of running the whole encoding-decoding loop to obtain the RDO cost for each case. Therefore, many algorithms have been proposed to narrow the search space down and speed up the motion estimation process [105, 94, 71, 50, 90, 101, 2, 72, 75].

The main idea of such algorithms is to conjecture the optimal partitioning based on the previously coded blocks (or frames) and some other information related to the frames' contents. For instance, the authors of [81] and [53] have presented a method for boosting the partition size decision process and skipping some unnecessary depths of Coding Units (CUs) that are less likely to be selected. They skip specific CU depths based on the depth of the neighboring or co-located CUs. In [53], the depths that are rarely used in the previous frame are skipped in the current frame for all the CUs as well.

In addition to the complexity of the ME, an important concern in video coding for machines is the utility and efficiency of ME itself. In order to encode the latent-space feature channels, one simple solution is to convert them to an image with pixels in the typical range of $[0, 255]$ and use traditional image/video codecs [18]. But, this approach might not be the best since the conventional codecs have been optimized for the human visual system.

On the other hand, the advantage of using trainable DNN models in image/video coding is their flexibility in getting tuned for different purposes via an appropriate loss function and

training procedure. Deep neural networks entered the area of image/video coding through the article of Ballé et al. [8], published in 2016, and evolved in 2018 with [64, 9]. Currently, there are a lot of end-to-end image codecs performing better than Ballé's models in the literature, such as [17], but still not completely better than VVC-Intra in all the well-known quality metrics. The main idea that enabled the DNN models to be used as an entropy model was estimating the bitrate in a differentiable manner by assuming a Gaussian distribution for the elements of the latent space. Since then, end-to-end trainable video codecs have started to emerge as well [62, 12].

The research works of [60] and [100] were the first end-to-end video encoders in which a frame interpolation takes place using a DNN model based on the previous (or future) frames and some motion information. In the end, the residuals will be coded through an entropy model similar to that employed in [8]. The authors of [60] also use a DNN-based optical flow estimation [73] as a ME engine. They improved their method [61] by implementing new ideas, such as an adaptive quantization layer for different bitrates to reduce the number of parameters for the entropy coder. Various models with different complexity and performance have been introduced in their new paper as well.

The end-to-end video codecs usually use a simple linear warping for the motion compensation, and the motion data comes from another part of the DNN model that processes the reference and current frames. The same procedure is followed in more recent works [76, 57, 27, 1] as well. Reference [76] introduces a modified structure for the whole DNN model, which benefits from a trainable state of tensors. This tensor state plays the role of a system memory with a mechanism to automatically populate the useful information and update it across time steps. In fact, the current frame uses the information of all the previous frames to exploit the existing redundancy better. Multi-scale ME and MC are the innovation of [57]. In this method, the motion estimation network generates the flow map in multiple scales. The warping is also done in those scales, and the final prediction is a combination of them. We also follow the same idea in our proposed models since the motion would be captured more accurately with different precision.

The authors of [27] engage two reference frames based on which two flow maps are produced by the optical flow network of [88]. In a sense, their proposed model conducts Bi-prediction. They also utilize an interpolation network that takes the optical flow maps, warped reference frames, and the current frames as inputs and creates a latent tensor to be coded in the entropy model. The decoder extracts two blending coefficient maps in addition to the flow maps from the coded data, and the final predicted frame is constructed with the weighted combination of the warped reference frames. The recent end-to-end video codec introduced by Google [1] has a relatively simpler design compared to other learning-based video encoders. This method focuses on addressing the challenges that exist in video coding, such as disocclusions and fast motions. In order to produce smaller residuals in such cases, various Gaussian-blurred versions of a frame with different intensities are generated, and

the model uses them in case of uncertainty. Thus, the entropy model encodes an additional dimension other than the 2-D motion flow, which specifies the proper filtered version of the reference frame to be used for making the prediction.

On the other hand, there is another technique to shift the pixels based on motion information other than pure warping. Deformable Convolutional Networks [26] are able to warp their input tensors based on an offset map that is a trainable tensor itself. In fact, warping is a particular type of deformable convolution in which the kernel value is 1 at the center and 0 elsewhere. Thus, the flexibility of deformable convolution layers is higher since the warped sampling points are combined with trainable weights. References [42] is an end-to-end video encoder based on deformable convolution layers. We have also used this technique in our proposed methods. The concept of deformable convolutional networks is described in more detail in Sections 3.4.2.

Apart from the end-to-end video codecs in the literature, some studies aim to replace a specific module of the traditional video codecs with DNN models to get a better gain. For example, [54] has designed a DNN model that receives the neighboring pixels of a block as inputs and predicts the pixels of that block. Since the DNN models' flexibility is substantially higher than hand-crafted methods, the authors of [54] provided the network with more rows of nearby pixels compared to the regular intra-prediction performed in traditional codecs. In [103], the fractional motion compensation module of the HM encoder is replaced with a DNN model. In other words, a trainable network handles the interpolation task to generate the pixel values for fractional positions. Reference [3] also attempts to improve the quantization process of the HEVC to make the final output more pleasant to human vision.

There is also another research field whose primary goal is frame interpolation to increase the frame rate of a video for a better and smoother quality [11, 82, 69, 70]. These learning-based methods could also be employed in video compression as they can create an intermediate virtual frame between two actual frames. Therefore, the same techniques could be used in frame prediction based on the information of nearby frames. The authors of [107, 108] have proposed the same idea and inserted the virtual interpolated frames into the reference picture buffer of HEVC to be used as references for coding future frames. Inspired by [70], the authors of [19, 20] predict the to-be-coded frame based on two previously decoded frames and define a new syntax element in the HEVC bit-stream, indicating three types of prediction for the current block: regular inter-prediction, regular intra-prediction, and the prediction generated by frame interpolation.

3.3 Study of ME using H.265/HEVC

The primary purpose of this section is to study the motion in the latent space and its relationship with input-space motion. If a suitably strong relationship exists between input-

space and latent-space motion, this can be exploited in various ways, for example, by using input motion to predict the latent space motion. This may further be used to optimize latent-space compression efficiency by improving motion estimation, motion compensation, and motion coding. This motion relation between the input space and latent space has been demonstrated in [92]. Yet we want to study the selected motions by an H.265/HEVC video encoder, which are obtained through minimizing a Lagrangian cost function:

$$\mathcal{L} = D + \lambda \cdot R \tag{3.1}$$

where R is the rate, D is the distortion, and λ is the Lagrangian multiplier.

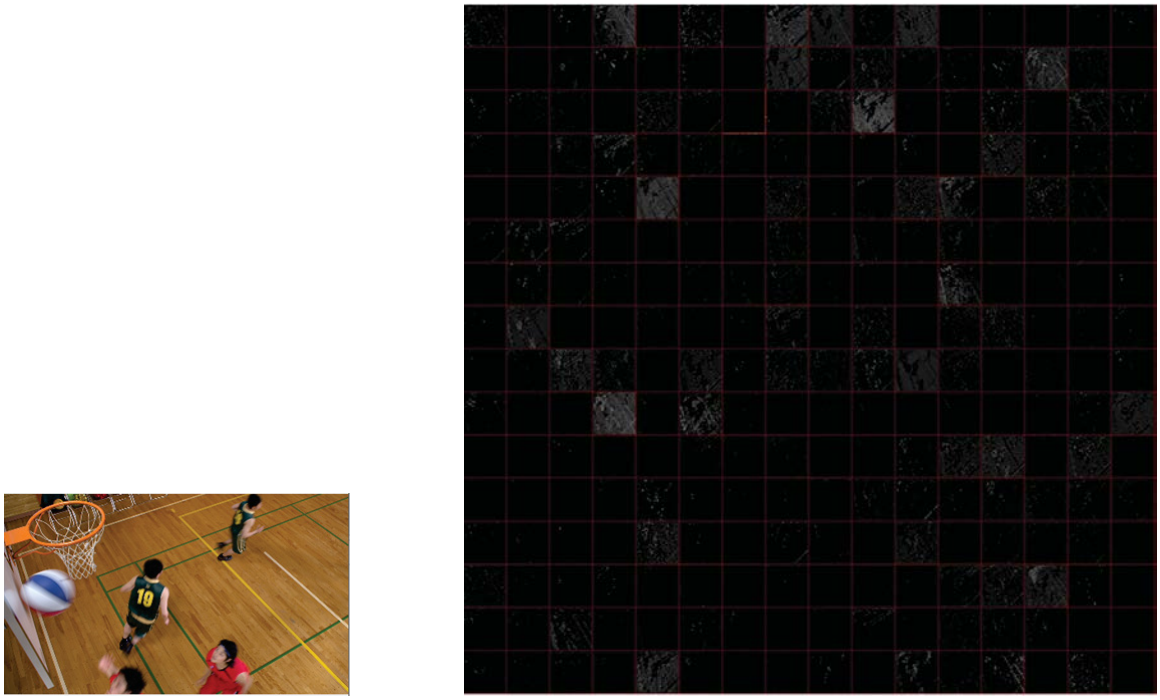
3.3.1 Experimental Setup

In this section, we use HM-16.20 [47] as an H.265/HEVC video encoder. YOLOv3 [74] object detection is our computer vision model whose Darknet implementation in C++ [93] is employed. YOLOv3 is split into two parts at layer 12. The latent space of YOLOv3 at layer 12 consists of 256 channels with the resolution of 64×64, assuming an input resolution of 512×512. In order to encode the latent space by HM, the channels are quantized to 8 bits and tiled into a 16×16 matrix to create a gray-scale image. An example of the tiled tensor representation of an input image is shown in Fig. 3.3. Then, the tiled tensor video is encoded via HM with the "low-delay P" configuration.

Since our experiments are conducted using HEVC, first, we need to increase the precision of HEVC motion vectors to be able to capture subtle motion in the latent space due to the low resolution of the feature channels. In Section 3.3.2, we describe the procedure of increasing the motion vector precision in HEVC and provide some experimental results. Afterward, we would be able to study the motion relation between input space and the latent space quantitatively in Section 3.3.3.

3.3.2 High-precision motion estimation in HEVC

As we go deeper into the DNN models, the resolution of the feature channels gets reduced. As a result, we need to increase the precision of the motion vectors if we want to capture more subtle motion, and relate the input-space motion to the latent-space motion. The standard accuracy of the motion vectors in HEVC is 1/4-pel precision, while it is 1/16-pel in VVC. One solution is to encode the tiled tensor videos in the latent space using VVC. However, the complexity of VVC is considerably higher than HEVC, which does not help the study of motion analysis. So, we decided to implement the interpolation filters of VVC into HEVC reference software and adjust the motion estimation and motion compensation parts of the HM to search for the best motion vectors with higher precision (e.g., 1/8-pel or 1/16-pel). Note that the entropy coding procedure of the motion vectors remains the same.



(a) (b)

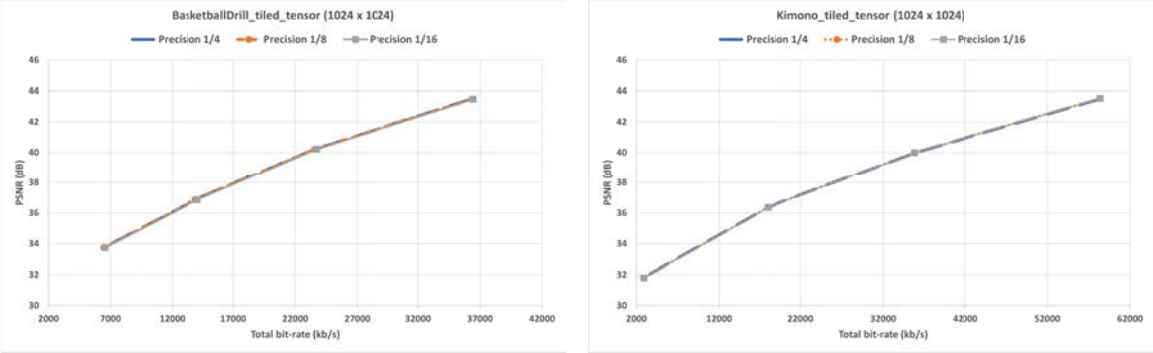
Figure 3.3: (a) Original frame (b) Tiled tensor representation at layer 12 of YOLOv3

The bitstream of the HEVC-coded videos for P-frames (or B-frames) mainly consists of the motion vector and residual data. By increasing the motion vectors' precision, the amount of data needed for representing the motion vectors would increase. Also, the amount of the data related to the residuals would possibly decrease if the higher precision search occurs around the best 1/4-pel precision motion vectors. However, the encoder's decisions in RDO would also change because of the higher cost of representing the motion vectors. In other words, the video encoder may end up selecting smaller motion vectors, due to their lower rate cost, when the precision is higher. Therefore, there is no guarantee that the residual data will decrease.

To better examine the effect of increasing motion vector precision, we have considered two cases for the RDO. In one case, RDO is performed based on the motion vectors' real cost, i.e., higher-precision motion vectors have a higher representation cost. We refer to this case as "normal." In the other case, the motion vector cost is based on 1/4-pel precision motion vectors, even though the actual precision is higher. We refer to the second case as "Q-pel-based" (Quarter-pel-based). It is evident that the amount of residual data in the Q-pel-based case would decrease (or remain the same) compared to the normal case since a finer search may find lower residuals without incurring extra cost (because the extra cost of motion vector representation is ignored in this case). At the same time, the amount of

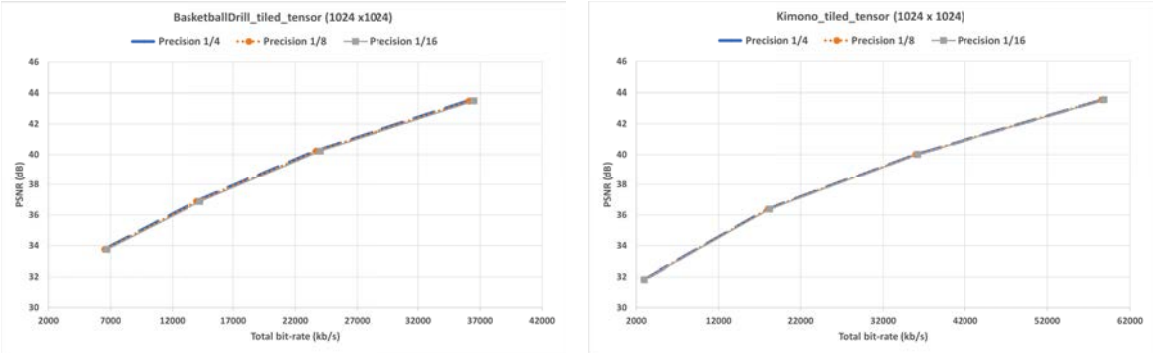
motion data would increase (or remain the same, in the best case) since the actual cost of motion vectors has not been considered in the RDO process.

Fig. 3.4 shows the bit-rate vs. PSNR curves for two tiled tensor video samples with the "normal" RDO and three different precisions. As seen in the figure, there is no noticeable difference between the curves corresponding to different motion vector precision. The same is true in the Q-pel-based RDO, shown in Fig. 3.5.



(a) (b)

Figure 3.4: Bit-rate vs. PSNR curves with normal RDO for tiled tensor sequences from (a) BasketballDrill, and (b) Kimono



(a) (b)

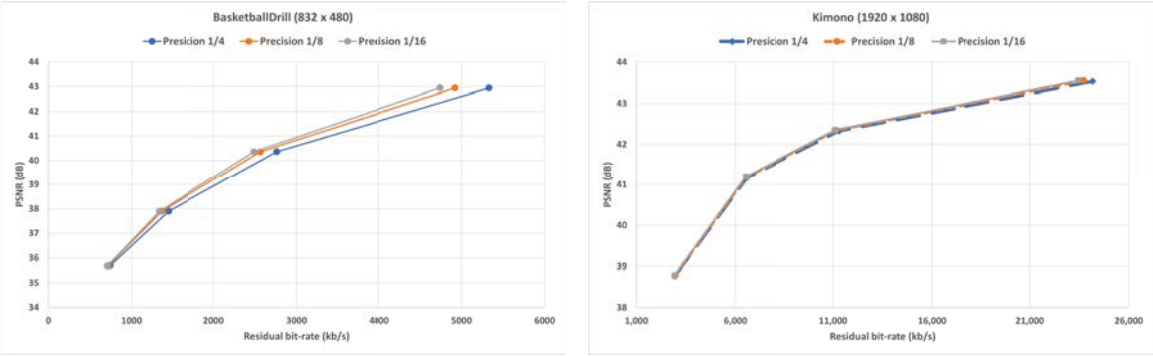
Figure 3.5: Bit-rate vs. PSNR curves with Q-pel-based RDO for tiled tensor sequences from (a) BasketballDrill, and (b) Kimono

To better understand the differences between various cases, we have measured the BD-bitrate related to the entire bitstream and the residual part of the bitstream with respect to the regular HEVC precision, which is 1/4. The results are shown in Table 3.1 for four test sequences: the original BasketballDrill and Kimono videos (input videos), and their tiled tensor videos. In this table, different RDO cases have also been considered. For completeness, Fig. 3.6 shows the bit-rate vs. PSNR curves with the Q-pel-based RDO for input sequences.

As shown in Table 3.1, for the tiled tensor test sequences with normal RDO, increasing the motion vector precision causes a loss in both residual and total bit-rate in most cases.

Table 3.1: Total BD-bitrate and residual BD-bitrate for different videos, RDO type, and precision; The anchor is HEVC with 1/4-pel motion precision.

Video	RDO mode	Precision	Total BD-bitrate	Residual BD-bitrate
BasketballDrill_tiled_tensor	normal	1/8	0.42 %	0.77 %
		1/16	1.18 %	2.55 %
	Q-pel-based	1/8	0.87 %	-1.60 %
		1/16	2.43 %	-2.12 %
Kimono_tiled_tensor	normal	1/8	-0.08 %	0.48 %
		1/16	0.30 %	1.58 %
	Q-pel-based	1/8	0.17 %	-1.31 %
		1/16	1.12 %	-1.43 %
BasketballDrill_original	Q-pel-based	1/8	-3.23 %	-6.17 %
		1/16	-2.76 %	-8.73 %
Kimono_original	Q-pel-based	1/8	0.42 %	-1.49 %
		1/16	1.46 %	-2.14 %



(a) (b)

Figure 3.6: Bit-rate vs. PSNR curves with Q-pel-based RDO for original (input) sequences (a) BasketballDrill, and (b) Kimono

However, ignoring the cost of high-precision motion vectors in the Q-pel-based mode RDO leads to a gain in the residual part and a loss in the motion vector part (since the total BD-bitrate is positive). An interesting point about these results is that the coding gain in the residual part due to increasing the motion vector precision is higher for the input videos than the latent-space tensor sequences. That might be because of the interpolation filters' inefficiency in the low-resolution feature channels, as explained in Section 3.1.2.

3.3.3 Motion Relationship between Input and Latent Space

Analysis of optical flow through various processing blocks in a Convolutional Neural Network (CNN) [92] shows that an approximate relationship between an input-space motion vector $\mathbf{v}$ and the latent-space motion vector $\mathbf{v}'$ at the same location is

$$\mathbf{v}' \approx \mathbf{v}/n^k, \tag{3.2}$$

where k is the number of pooling layers between the input and the location of the latent-space feature tensor in the CNN, and n is the downsampling factor (stride) used in the pooling layers. While this explains why motion in the latent space looks roughly similar to the input motion, the question is whether this approximate relationship is accurate enough for improving latent-space compression efficiency. In other words, "is $\mathbf{v}'$ the selected motion vector in the RDO process when encoding the latent-space video?"

In order to study the motion relationship, we have performed an experiment on the BasketballDrill test sequence. The steps of this experiment are as follows:

1. A frame of the input video is considered.
2. The whole frame is shifted in the horizontal (or vertical) direction by a specific vector like $S = (s_x, s_y)$.
3. The corresponding tiled tensor images of the original and the shifted frames are extracted.
4. Inter-prediction using HM is performed on the tiled tensor image obtained from the shifted frame with a reference frame equal to the tiled tensor image obtained from the main input frame.
5. The estimated latent-space motion vectors $MV = (mv_x, mv_y)$ are compared with input motion S to examine whether (3.2) holds. Suppose the resolution of the input video is $m \times n$ and the resolution of the feature channels is $m' \times n'$. If (3.2) holds, latent space motion vectors should be equal to $(s_x \cdot \frac{m'}{m}, s_y \cdot \frac{n'}{n})$.

The resolution of the BasketballDrill sequence is 832×480, and frames are rescaled to 512×512 prior to feeding them to the YOLOv3 model. With the input size of 512×512, the resolution of the feature channels at layer 12 of YOLOv3 is 64×64. So, the ratio of the resolutions is 13 horizontally and 7.5 vertically. This means that, according to (3.2), a horizontal shift of 13 pixels in the input video should translate to a 1-pixel horizontal shift in the feature channels, and a vertical shift of 7.5 pixels in the input video should translate to a 1-pixel vertical shift in the feature channels. Since the precision of the latent-space motion vectors is 1/16, we will consider the two closest 1/16-pel precision motion vectors as the expected motion in the tiled tensor.

The following tables show the results of our experiments. In these tables, rows correspond to the different values of the shift in the input space in the horizontal (or vertical) direction, i.e., the values of s_x (or s_y). Columns correspond to different values of mv_x (or mv_y) with 1/16-pel precision, such that column i corresponds to an MV in the latent space equal to $(i/16, 0)$ for horizontal input shifts, and $(0, i/16)$ for vertical input shifts. The last column corresponds to motion vectors with values other than those specified in the first 17 columns. The values in the tables are the percentages of the pixels in the tiled tensor images having

an MV specified by its column. For example, in Table 3.2, the value of 1.2 in the blue cell means that 1.2% of the pixels in the tiled tensor image have the motion vector equal to $MV = (6/16, 0)$ when the input frame is shifted by 7 pixels in the horizontal direction. The green cells are the expected motion vectors. If (3.2) were completely accurate, the values in green cells would add up to 100% in each row. Meanwhile, the red cells are the peaks (i.e., relatively large number of MVs) away from the expected MVs. We call these tables "Motion Relationship Tables." In each motion relationship table, we compute the average of the sum of the green cells in each row and refer to it as "AGC" (Average of Green Cells). Tables 3.2 and 3.3 are the motion relationship tables for all the feature channels in the case of horizontal and vertical input shifts, respectively.

Note that the green cell in the last row in Table 3.2 has a value of 94.9. This means that when the input horizontal shift is 13 and corresponds to the latent-space horizontal shift of $16/16 = 1$, equation (3.2) holds for 94.4% of the MVs in the latent space. In this case, (3.2) is a good model. However, in other rows of the table, the values in the green cells add up to a much smaller percentage, indicating that (3.2) is less accurate in these cases. In summary, (3.2) is a reasonably good model when the expected latent-space motion corresponds to an integer-pixel shift, but not a good model for fractional-pixel motion. This is probably because of the HEVC and VVC interpolation filters' inefficiency in the latent space.

In order to further examine the relationship between input-space and latent-space motion, we also looked at different subsets of feature channels in terms of their texture content to see the effects of texture on the latent-space motion. Specifically, we obtained the motion relationship tables for a subset of 20 channels with maximum horizontal and vertical gradient, minimum horizontal and vertical gradient, and maximum and minimum standard deviation. The results are given in Tables 3.4-3.9 for the horizontal shift. The motion relationship tables for the vertical shifts look similar to those for the horizontal shifts, and are therefore not included. As seen in the tables, the value of AGC is higher in channels with a stronger gradient and larger standard deviation, indicating better agreement with (3.2) in such channels. Meanwhile, low-textured channels with lower gradient and lower standard deviation show poorer agreement with (3.2), as illustrated by the higher number of red cells and lower AGC values.

The summary of AGC values from these experiments is given in Table 3.10. According to the results, it can be inferred that input motion tends to be better preserved (in accordance with (3.2)) in textured channels, i.e., channels with a strong gradient or high standard deviation. Therefore, texture analysis may indicate which channels or regions in a channel could benefit from estimated input motion. This conclusion confirms claim number 4 in Section 3.1.2. However, the values of AGC are relatively low to deduce that there is a linear relationship (according to (3.2)) between the input-space motion and the latent-space motion, especially given the fact that the largest contributing factor for AGC is the

Table 3.2: Motion relationship table for all the channels, horizontal shift, $AGC = 22.3\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	55.1	14.4	3.1	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	27.0
2	45.0	4.8	10.5	2.9	0.7	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	35.7
3	39.3	1.7	3.8	6.9	3.2	0.4	0.3	0.1	0.1	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	44.2
4	21.5	0.5	1.6	2.6	6.8	4.3	2.1	0.5	0.2	0.1	0.1	0.0	0.1	0.0	0.0	0.0	0.0	59.7
5	9.0	0.1	0.3	0.7	2.0	5.7	11.5	2.3	1.0	0.2	0.1	0.1	0.0	0.0	0.1	0.0	0.0	66.8
6	5.6	0.0	0.2	0.2	0.4	1.0	6.9	7.5	7.4	1.1	0.7	0.3	0.3	0.0	0.1	0.0	0.0	68.2
7	4.0	0.0	0.2	0.1	0.2	0.4	1.2	0.8	6.1	8.5	6.6	1.0	0.5	0.2	0.2	0.1	0.1	69.9
8	2.4	0.2	0.1	0.1	0.1	0.1	0.3	0.4	0.9	3.6	12.1	11.7	1.6	0.7	0.3	0.2	0.1	65.4
9	2.4	0.0	0.0	0.1	0.1	0.0	0.1	0.0	0.3	0.2	2.8	5.9	9.9	2.3	1.3	0.4	0.5	73.7
10	1.8	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.6	0.9	6.4	16.0	5.7	1.5	1.1	65.8
11	1.5	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.1	0.1	1.0	4.1	15.8	10.0	6.1	61.0
12	1.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.4	3.9	21.7	29.1	42.7
13	1.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	94.9	3.0

Table 3.3: Motion relationship table for all the channels, vertical shift, $AGC = 26.5\%$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	9.8	24.2	4.0	0.9	0.1	0.1	0.0	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	60.6
2	0.8	4.1	11.9	23.8	4.9	2.0	0.5	0.1	0.1	0.2	0.1	0.0	0.0	0.0	0.0	0.0	51.3
3	0.6	1.0	1.7	2.2	7.3	17.8	6.5	2.0	0.6	0.7	0.1	0.3	0.1	0.2	0.1	0.0	58.9
4	0.1	0.2	0.2	0.4	0.8	2.0	3.0	9.8	7.9	10.3	2.1	1.3	1.0	0.7	0.7	0.1	59.4
5	0.0	0.0	0.1	0.2	0.1	0.3	0.2	0.6	0.8	11.0	14.3	8.6	3.2	3.7	1.2	1.1	54.7
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.7	4.2	14.2	12.6	13.8	5.7	48.1
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.2	2.9	40.3	37.6	18.7

Table 3.4: Motion relationship table for 20 channels with maximum horizontal gradient, shift direction = horizontal, $AGC = 26.1\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	34.7	19.2	4.1	0.4	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	41.4
2	20.1	5.4	12.5	4.8	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	56.2
3	15.0	0.7	4.4	10.9	5.1	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	63.5
4	7.0	0.3	0.6	2.6	5.8	5.6	1.1	0.8	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	76.0
5	2.3	0.0	0.1	0.1	0.3	8.3	12.2	1.0	1.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	74.5
6	1.2	0.0	0.0	0.1	0.1	0.6	3.4	7.7	7.0	0.5	0.1	0.4	0.1	0.0	0.0	0.0	0.0	78.7
7	0.5	0.0	0.1	0.0	0.0	0.1	0.1	0.0	5.0	9.8	3.3	0.9	0.0	0.0	0.0	0.1	0.1	80.1
8	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.6	1.9	14.7	14.7	0.6	0.0	0.0	0.0	0.0	67.3
9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	1.1	7.8	11.1	0.3	0.3	0.0	0.4	78.5
10	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	1.2	7.9	21.9	4.5	0.7	0.5	63.0
11	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.4	7.2	20.8	8.3	3.0	60.1
12	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	4.8	31.0	12.8	50.5
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	99.7	0.2

Table 3.5: Motion relationship table for 20 channels with minimum horizontal gradient, shift direction = horizontal, $AGC = 12.0\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	62.1	7.7	1.6	0.3	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.0
2	54.9	6.3	6.1	0.9	2.1	0.9	1.2	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	27.4
3	53.3	2.6	4.9	0.3	2.0	0.2	0.3	0.0	0.9	0.1	0.0	0.2	0.0	0.0	0.0	0.0	0.0	35.1
4	45.5	0.4	2.9	3.2	5.6	1.5	3.9	0.5	0.9	0.0	0.4	0.4	0.4	0.0	0.1	0.0	0.0	34.4
5	35.1	0.0	0.9	4.5	7.2	0.0	3.6	6.6	1.5	0.3	0.5	0.0	0.0	0.0	0.0	0.0	0.0	39.8
6	42.5	0.0	0.7	0.7	0.7	2.5	1.4	1.4	1.4	2.1	0.8	0.1	0.1	0.0	0.8	0.1	0.0	44.6
7	40.0	0.0	0.5	1.2	0.1	0.5	2.8	1.1	0.1	0.4	6.8	1.1	1.3	0.8	0.5	0.3	0.1	42.2
8	32.6	5.6	0.0	1.0	0.2	0.2	0.5	1.4	0.0	1.2	0.7	1.7	0.3	5.7	1.0	0.2	0.0	47.6
9	37.2	0.0	0.0	0.6	0.3	0.0	0.0	0.0	0.8	0.0	1.8	1.2	3.0	0.3	0.6	0.5	0.5	53.2
10	25.5	0.0	0.1	0.9	0.1	0.0	0.0	0.4	0.2	0.7	1.1	0.9	1.6	1.6	5.2	1.6	2.1	57.9
11	11.1	0.0	0.0	0.0	2.3	0.1	0.0	0.0	0.6	0.6	0.0	0.0	1.1	2.0	13.3	12.6	4.3	52.0
12	14.2	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	4.8	13.5	20.4	45.8
13	13.5	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	1.6	73.9	10.8

Table 3.6: Motion relationship table for 20 channels with maximum vertical gradient, shift direction = horizontal, $AGC = 35.6\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	34.2	26.9	7.6	0.4	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	30.8
2	21.3	5.3	21.9	7.3	1.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	43.0
3	16.1	0.3	5.3	16.6	10.3	0.5	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.6
4	7.0	0.2	0.9	5.0	10.9	9.9	1.8	0.6	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	63.5
5	2.8	0.0	0.2	0.1	0.8	13.1	17.1	1.5	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	63.5
6	1.1	0.0	0.0	0.1	0.2	0.7	5.2	12.8	10.5	0.0	0.2	0.5	0.1	0.0	0.1	0.0	0.0	68.6
7	0.6	0.0	0.0	0.0	0.0	0.1	0.1	0.8	13.8	15.8	4.4	0.7	0.3	0.0	0.0	0.1	0.1	63.3
8	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.4	5.2	22.2	17.1	0.6	0.2	0.1	0.0	0.0	52.5
9	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	2.5	15.4	12.7	0.6	0.2	0.3	0.4	67.2
10	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.4	8.7	33.3	2.9	0.8	0.9	50.9
11	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.7	11.7	27.5	10.0	3.1	46.5
12	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	8.8	35.5	13.0	41.7
13	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	99.4	0.5

Table 3.7: Motion relationship table for 20 channels with minimum vertical gradient, shift direction = horizontal, $AGC = 11.4\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	60.6	8.7	1.3	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	29.0
2	53.2	6.4	5.8	0.7	2.1	0.2	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	31.1
3	49.8	3.0	4.9	1.5	2.1	0.1	0.4	0.0	0.0	0.0	0.0	0.2	0.1	0.0	0.0	0.0	0.0	38.0
4	45.2	0.0	2.0	3.6	5.4	1.8	1.8	0.7	1.0	0.0	0.3	0.4	0.0	0.0	0.0	0.0	0.0	37.9
5	33.0	0.0	0.9	4.7	7.6	0.0	2.9	5.8	1.8	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	43.0
6	38.8	0.0	0.4	0.5	0.1	1.7	1.3	0.4	1.8	1.8	0.5	0.4	0.1	0.0	0.5	0.0	0.0	51.7
7	39.6	0.0	0.3	1.1	0.0	0.3	0.0	1.1	0.7	0.0	7.0	1.1	0.9	0.9	0.3	0.3	0.0	46.4
8	31.7	5.7	0.0	1.4	0.0	0.4	0.5	1.4	0.0	1.1	1.6	0.4	0.4	1.6	0.5	0.0	0.0	53.4
9	35.1	0.0	0.0	1.0	0.1	0.0	0.0	0.3	0.7	0.0	1.7	2.0	2.9	0.3	0.6	0.4	0.4	54.3
10	23.3	0.0	0.1	0.8	0.0	0.0	0.0	0.2	0.1	0.0	0.9	0.7	0.7	1.8	5.5	1.5	1.9	62.3
11	11.5	0.0	0.0	0.0	2.3	0.0	0.0	0.0	0.6	0.6	0.0	0.0	1.9	2.0	11.9	12.0	5.7	51.5
12	14.2	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	1.2	4.0	8.8	19.7	52.1
13	13.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.6	72.0	13.2

Table 3.8: Motion relationship table for 20 channels with maximum standard deviation, shift direction = horizontal, $AGC = 30.3\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	36.7	23.0	4.7	0.6	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	34.8
2	22.0	5.4	15.5	7.0	1.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	49.0
3	16.7	1.5	4.7	12.8	7.1	0.2	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	56.6
4	8.3	0.2	1.0	3.4	8.5	8.7	1.0	0.6	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	68.0
5	3.2	0.0	0.1	0.1	0.5	12.5	15.5	0.7	2.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	65.1
6	1.6	0.0	0.0	0.1	0.1	0.9	4.0	8.5	8.4	0.9	0.1	0.4	0.2	0.0	0.0	0.0	0.0	74.8
7	1.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	8.4	13.2	2.7	1.0	0.0	0.0	0.1	0.1	0.1	73.2
8	0.3	0.0	0.0	0.0	0.0	0.1	0.0	0.3	0.7	3.5	17.5	18.2	0.6	0.0	0.1	0.0	0.0	58.6
9	0.4	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.2	0.3	2.4	10.8	13.8	0.1	0.4	0.3	0.4	70.9
10	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	1.3	7.9	29.2	3.5	1.1	0.9	55.1
11	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.6	10.7	24.2	6.4	3.1	54.7
12	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.8	7.6	31.1	13.6	45.8
13	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	96.0	2.2

Table 3.9: Motion relationship table for 20 channels with minimum standard deviation, shift direction = horizontal, $AGC = 12.2\%$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	~
1	61.3	7.5	1.6	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	29.0
2	53.4	6.8	6.2	0.5	2.4	0.9	1.2	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	28.5
3	52.3	2.6	2.2	1.2	1.2	0.6	0.4	0.1	0.9	0.1	0.0	0.2	0.0	0.0	0.0	0.0	0.0	38.1
4	43.2	0.3	2.0	0.9	5.5	3.8	2.7	0.7	0.9	0.0	0.3	0.3	0.1	0.0	0.0	0.0	0.0	39.3
5	32.1	0.0	0.9	4.0	6.9	0.0	1.4	5.3	1.4	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	47.4
6	34.2	0.0	0.3	0.5	1.1	2.2	1.3	1.4	0.6	1.8	0.6	0.1	0.0	0.0	0.5	0.1	0.0	55.3
7	34.6	0.0	0.3	1.2	0.5	0.0	3.1	0.8	0.0	2.1	6.1	1.0	0.8	0.8	0.0	0.3	0.1	48.4
8	25.7	5.2	0.3	1.0	0.0	0.2	0.5	1.0	0.3	0.7	0.7	2.3	0.7	5.4	1.0	0.0	0.0	55.1
9	31.0	0.0	0.0	0.6	0.2	0.0	0.0	0.0	0.8	0.0	7.7	1.1	3.8	0.3	0.3	0.5	0.5	53.5
10	19.6	0.0	0.1	0.9	0.0	0.0	0.0	0.4	0.1	0.7	1.1	0.9	0.9	1.4	5.7	1.7	1.1	65.3
11	9.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.6	0.0	0.0	1.8	2.7	11.2	13.0	5.6	55.3
12	13.1	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.2	1.2	2.6	18.7	16.2	48.0
13	11.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.6	76.8	9.6

high value of the green cell in the last row, which corresponds to a full-pixel shift in the latent space. Some possible reasons for deviation from (3.2) are as follows:

- Interpolation filters might not be very effective in the low-resolution latent-space feature channels, so fractional motion estimates might not be very reliable.
- Latent-space motion was estimated using HEVC motion estimation tools by minimizing the RDO cost of (3.1), and it is well-known that these motions do not necessarily equal the accurate optical flow motions.
- Max-pooling operations may cause shifts in edge locations compared to conventional downsampling.
- Analysis in [92] shows that input motion remains one possible solution to optical flow after the convolution operation, as well as pointwise nonlinear activations. However, these operations may also introduce other solutions to the optical flow, meaning that motion may simply change when the input signal passes through such operations.

Table 3.10: AGC values for different subsets of feature channels in the latent space

Subset of Channels	Average of Green Cells (%)	
	Horizontal Shift	Vertical Shift
Entire Channels	22.3	26.5
Max Horizontal Gradient	26.1	55.3
Min Horizontal Gradient	12.0	10.3
Max Vertical Gradient	35.6	42.4
Min Vertical Gradient	11.4	13.1
Max Std Channels	30.3	46.3
Min Std Channels	12.2	10.2

3.4 DNN-based temporal prediction in the latent space

In Section 3.3.3, we examined whether equation (3.2) is accurate enough to be used effectively for predicting latent-space motion from input-space motion. It was experimentally shown that, while (3.2) holds approximately for integer-pixel shifts in the latent space, and in channels with high texture, it is not accurate enough for high-quality prediction of latent-space motion from input motion. Also, in Section 3.3.2, we figured out that high-precision interpolation filters are not very effective in the latent space.

According to our observations and the challenges stated in Section 3.1.2, a non-linear approach could be more beneficial for ME in the latent space. Therefore, a DNN model can be used to do the ME and MC task in the latent space of another machine vision DNN

model. As it is well-known, the DNN models are capable of handling a variety of tasks, including optical flow estimation [88, 43, 91]. Hence, they are able to generate appropriate motions exclusively for each feature channel in the latent space, even for the low-resolution ones, if a proper architecture and training procedure are employed.

3.4.1 Experimental Setup

In this section, the base object detection model is YOLOv5 [48], like Chapter 2. However, here, we chose the best and most complex model, YOLOv5x6. This model has been trained on images of the COCO dataset [55] resized to the width (or height) of 1280 with preserved aspect ratio. The cutting point is at layer 5 in this section, too (see Fig. 2.1.)

The effective stride (downsampling factor) up to layer 5 is 2^3. In other words, the size of the input image to the model is reduced by a factor of 2^3 in both width and height up to this split layer. Also, this latent space of YOLOv5 consists of 320 channels. For encoding the tensors, we have also used H.266/VVC [16], specifically, its VVenC [97] implementation with the *lowdelay-faster* preset.

Since VVC encodes the residuals, it would be better to make the size of the channels in the tiled tensor frames compatible with the size of the CTUs in VVC. The size of the channels in the latent space will be 128×128 if we resize the input frames to 1024×1024. The YOLOv5x6 model's mAP@.5:.95 on the COCO validation set with inputs resized to 1024×1024 is 52.9%. This mAP calculation is done with the built-in function in the YOLOv5 source code. Similar to the steps taken in the previous section for encoding the latent space, the feature channels are quantized to 8 bits and tiled into a proper matrix to create a gray-scale image.

3.4.2 Proposed Methods

The core of our proposed methods is a ME and MC engine based on DNNs. This engine is designed in a way to predict the bottleneck tensors generated from the YOLOv5 model at layer 5. The inputs to our DNN inter-predictor models are the two previous frames of the latent tensors ($\widehat{T}_{n-1}$ and $\widehat{T}_n$), and the output is a prediction for the current frame of the tensors ($\widehat{\widehat{T}}_{n+1}$). The predicted tensors are subtracted from the original tensors, and the obtained residuals will be clipped, scaled to the range of $[0, 255]$, and coded using VVC-Intra without making a further temporal prediction (see Fig. 3.7.) The coded bitstream on the cloud is decoded, scaled back to the original range, added to the predictions, fed to the decoder part of the autoencoder, and passed to the YOLOv5 back-end to obtain the inference results.

Since the final residuals will be coded by VVC, the resolution of the tiled tensors in the latent space is an important factor in compression efficiency. So, dimensionality reduction in the latent space could be advantageous. To this end, we insert an autoencoder at the split point in order to reduce the number of channels. Note that the spatial dimensions remain

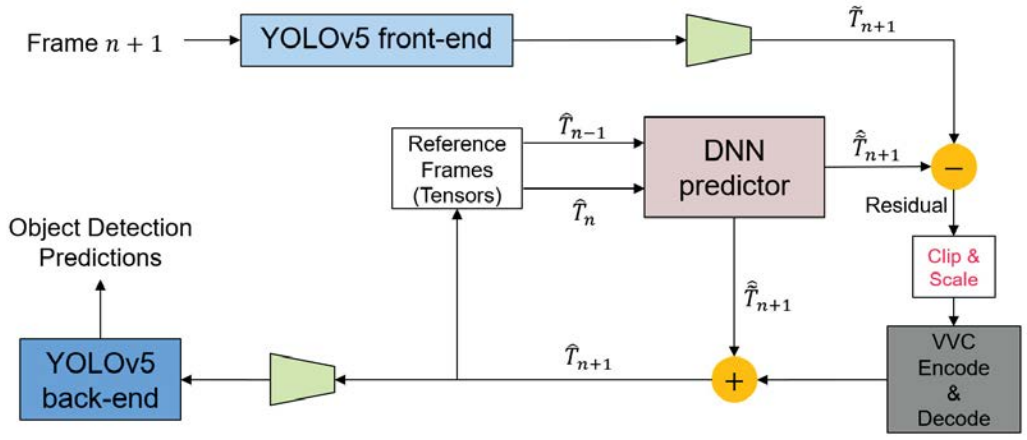


Figure 3.7: The block diagram of the proposed video coding pipeline in deployment

unchanged to preserve the spatial precision of subsequent object detection. The encoder and decoder parts of the autoencoder are specified by green trapezoids in Fig. 3.7.

Autoencoder

As shown in Fig. 3.8, the autoencoder is placed at the split point, and it can be trained on an image dataset with object detection annotations like COCO [55] independent of the DNN predictor and with the native object detection loss. This is an important point because the amount of object-annotated image data is much larger than that of object-annotated video data. Afterward, the DNN predictor could be built upon this shrunk latent space and trained with typical loss functions like MAE or MSE between the predicted and original tensors (see Fig. 3.9).

Training the predictor on the object-annotated video data using object-detection loss would likely lead to better results, but the amount of such annotated video data is very limited. In fact, to the best of our knowledge, SFU-HW-Objects-v1 [22] is one of the rare video datasets with object detection annotations that is publicly available and will be kept for test purposes, and not training.

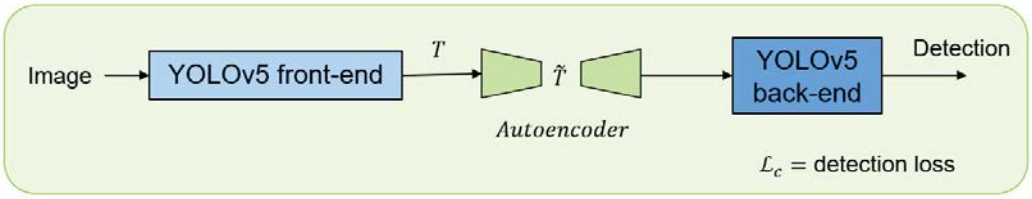


Figure 3.8: The overall block diagram of YOLOv5 with the autoencoder in the training stage

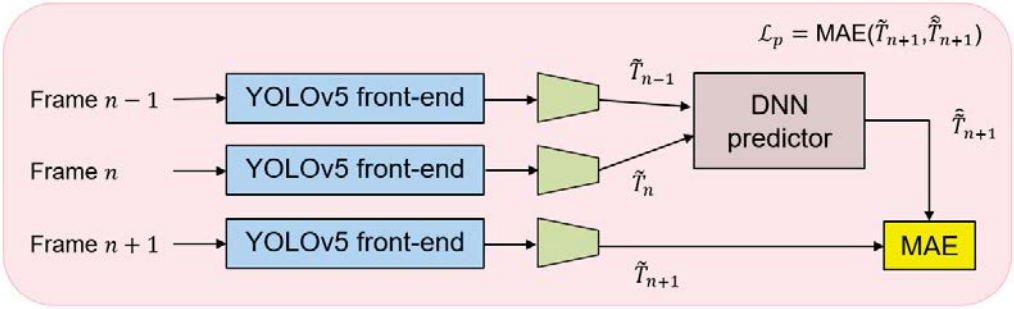


Figure 3.9: The overall block diagram of the DNN predictor in the training stage

The autoencoder's architecture is relatively simple. Both the encoder and decoder parts consist of two 3×3 convolution layers followed by SiLU activations (see Fig. 3.10). The number of channels is 320, 192, and 64 in the encoder's layers. These numbers are in the reverse order in the decoder. So, the 64 channels in the encoder's last layer constitute the pipeline's bottleneck, which will be encoded. Reducing the number of channels in the latent space has pros and cons. The main advantage is that it reduces the data volume in the bottleneck, which should lead to better compression efficiency. However, this redundancy reduction could make the back-end of the model more vulnerable to quantization errors in the bottleneck.

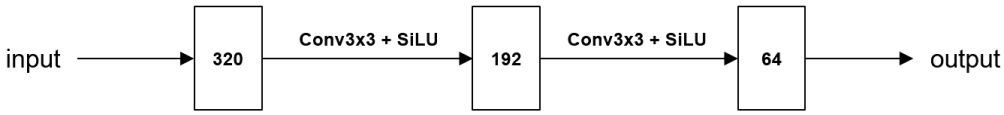


Figure 3.10: The encoder portion of the autoencoder

For tuning the pipeline depicted in Fig. 3.8, we trained the model in two stages. In both stages, the weights of the YOLO model were initialized to the original pre-trained weights of the YOLOv5x6 model. In the first stage, the entire YOLOv5 model was frozen, and only the autoencoder was trained for 140 epochs. In the second stage, the autoencoder and the YOLOv5 front-end were frozen, and only the YOLOv5 back-end was tuned to adapt to the new latent space generated by the autoencoder. We trained the pipeline in both stages on the COCO dataset using Stochastic Gradient Descent (SGD) optimizer with the main YOLOv5 object detection loss. We also utilized a cosine learning rate decay scheduler (the default scheduler in YOLOv5) for training. The initial learning rates were 10^{-2} and 2×10^{-3} for the first and second stages, respectively. The best mAP that the whole model reached in the first stage was 52.65%. In the second stage, the model converged quickly, and the best mAP was 52.81% (only 0.1% below the benchmark mAP achieved without the autoencoder).

Deformable Convolution

The basis of the proposed inter-prediction models is Deformable Convolution Blocks [26, 109]. Although deformable convolutions are primarily designed to improve computer vision tasks, they are also getting popular in image/video compression. Deformable convolutions generally encourage the receptive field of the whole network to adapt to the shape of the objects by tuning their offset fields. The offset fields in deformable convolutional networks could play the role of motion vectors in the inter-prediction models. As shown in Fig. 3.11, for each pixel in the output, there is an exclusive offset map with a size equal to the size of the kernels. These offset values determine the sampling point of the input where the convolution kernel should be applied. The main advantage of the offset maps is that they are trainable parameters themselves and can adapt to the input.

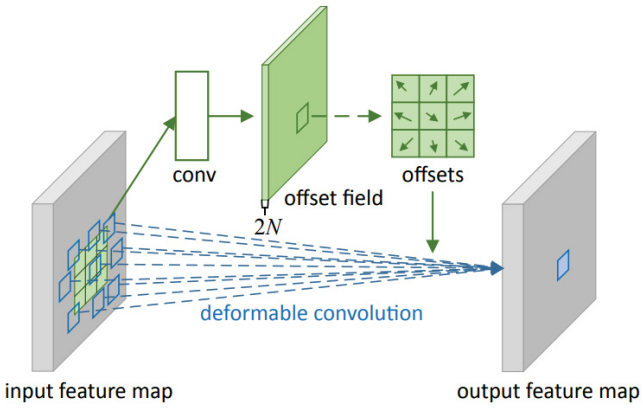


Figure 3.11: Illustration of a 3×3 deformable convolution [26]

As explained in [26], regular convolution layers are applied to the input by sampling using a rectangular grid $\mathcal{R}$ and summation of sampled values weighted by $\mathbf{w}$. The grid $\mathcal{R}$ defines the receptive field size and dilation. For example,

$$\mathcal{R} = \{(-1,-1),(-1,0),\ldots,(0,1),(1,1)\}$$

is a simple 3×3 kernel with dilation 1.

For each location $\mathbf{p}_0$ on the output feature map $\mathbf{y}$, we have

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n), \tag{3.3}$$

where $\mathbf{p}_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta\mathbf{p}_n | n = 1, ..., N\}$, where $N = |\mathcal{R}|$. Then, 3.3 becomes

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n). \tag{3.4}$$

Now, the sampling is on the irregular and offset locations $\mathbf{p}_n + \Delta\mathbf{p}_n$. As the offset $\Delta\mathbf{p}_n$ is typically fractional, a bilinear interpolation is performed on the 4 closest nearby pixels to obtain the desired value for the fractional positions.

Inspired by [42], we also employed deformable convolutions in our DNN predictor network to improve the motion estimation efficiency.

DNN Inter-Prediction Models

Three different models are examined as the DNN predictors, which will be described in this section. All of these models have an architecture resembling U-Net [77]. The ME and MC are performed in multiple resolutions of the feature channels. We chose the Vimeo-90k triplet dataset [102] to train the DNN predictor model as depicted in Fig. 3.9.

- **Model-1:**

  The architecture of this model is given in Fig. 3.12. "|" operator means concatenation on the right side of the network. The left part of the network contains some deformable convolution layers, while the right side consists of regular convolution blocks. Thus, the first three layers are supposed to do the ME and MC, and the last two layers upsample the tensors to the original resolution.

  See Fig. 3.13 for the detailed illustration of the Model-1 blocks. In deformable convolution layers, there is a parameter called Groups (G). G specifies the number of groups into which the input channels to the layer should be divided. Each group shares a similar offset field. The larger values of G make the channels in the latent space be treated more exclusively. Therefore, a high value of G would result in generating different and probably more efficient offset fields for different channels, as well as increasing the model parameters and complexity. In Model-1, the value of G equals the number of channels in the bottleneck of the autoencoder, which is 64. So, each channel has its own individual offset map.

  We trained Model-1 for 60 epochs using the SGD optimizer with MAE loss between the predicted and the ground-truth feature channels in the bottleneck. The learning rate scheduler was the same as that for training the autoencoder, a cosine learning rate decay, with 10^{-2} initial learning rate. The best values of MAE and MSE on the test set of the Vimeo-90k for this model were 0.1574 and 0.05739, respectively.

- **Model-2:**

  One of the critical problems with Model-1 is the lack of a reliable motion estimation module. According to the architecture of Model-1 shown in Fig. 3.12, the model is

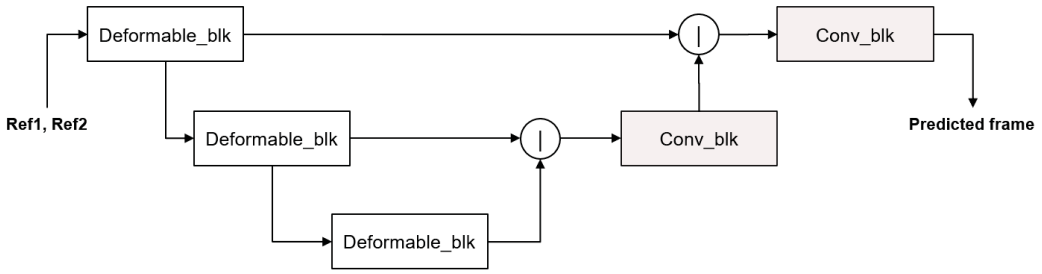
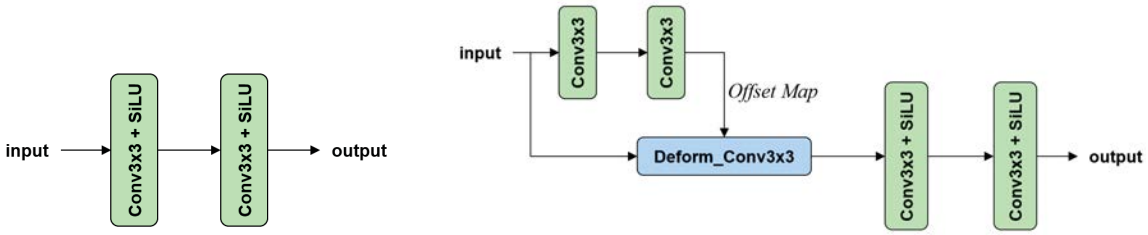


Figure 3.12: The architecture of Model-1



(a) (b)

Figure 3.13: The architecture of the blocks existing in Model-1 (a) Conv_blk (b) Deformable_blk

intended to perform multi-scale motion compensation. However, layers 2 and 3 cannot obtain motion data directly from the two reference inputs because the input to layers 2 and 3 are the output features of their previous layers. From the literature on end-to-end trainable video codecs, we notice a common trend in the motion estimation and compensation part of the networks, which is having a separate network for motion estimation fed by the reference inputs directly, and an independent network for motion compensation fed by the motion information obtained from the motion estimator.

So the architecture of Model-2 is designed in a way to have decoupled parts for motion estimation and compensation. The new model's architecture is shown in Fig. 3.14. The new model still has a U-Net-like architecture and benefits the deformable convolutions but more reasonably. The motion estimation part processes the reference tensors and generates the motion fields for the deformable convolution layers in two different depths of the motion compensation networks. We also added some Residual Blocks to the network to help the gradients flow better. The Refinement module at the end of the network is supposed to combine the motion-compensated versions of the reference frames obtained from the two branches and improve the quality of the final predicted tensors. The parameters on the arrows determine the number of feature channels at that layer (c is the number of channels of each reference tensor, which is 64). Another difference with the previous model is the number of groups (G) in the deformable

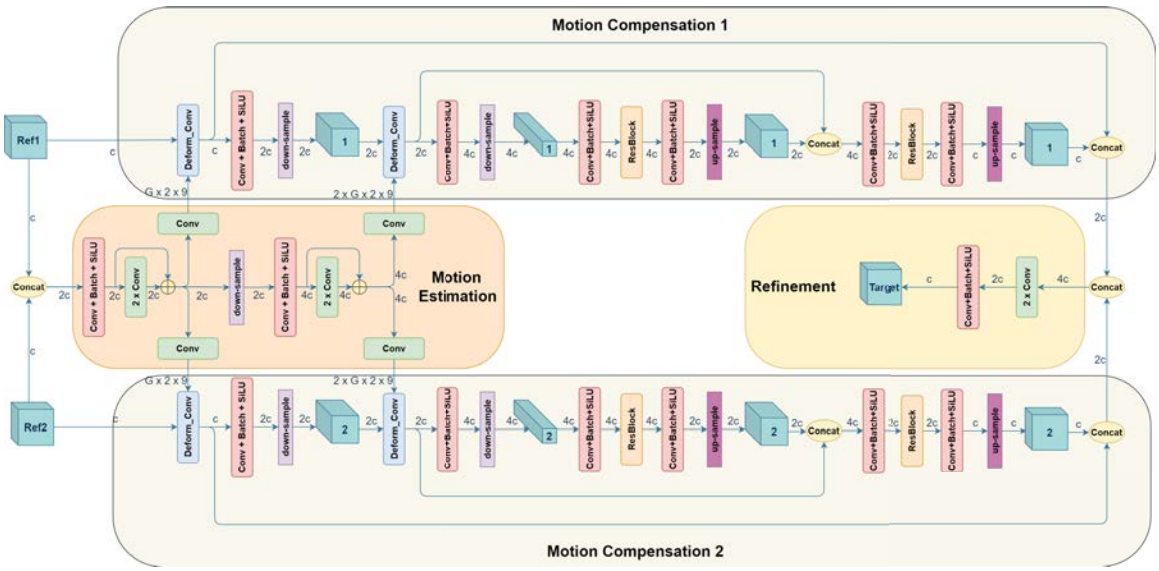


Figure 3.14: The architecture of Model-2

convolution layers. The number of groups has been decreased to reduce the complexity of the model, and it is proportional to the number of channels in the corresponding layer (G for the first depth and $2G$ for the second depth). In this model, G is 8.

We trained Model-2 with the identical setup described for Model-1 for 40 epochs. The best MAE and MSE on the Vimeo-90k test set for this model were 0.1247 and 0.03764, respectively, which is significantly lower than those for Model-1. This could be a sign of better motion compensation in the final predicted feature channels.

- **Model-3:**

  The structure of Model-3 resembles Model-2. The key difference is the places where deformable convolution blocks have been used. In Model-2, these layers existed on the left side of the network (depths 1 and 2), implying a fine-to-coarse motion compensation. However, a more reasonable way is to perform motion compensation in a coarse-to-fine manner, as in the conventional and even DNN-based video coding methods like [57]. Hence, we shifted the location of the deformable convolution layers to the right side of the network. According to some state-of-the-art DNN-based optical flow methods in the literature, like [91], it would also be better to have a pre-processing network on the input references to make them ready for better motion estimation. This is another component added to the new model. As a result, the number of layers is slightly higher than in Model-2. The number of groups in the deformable convolution layers is $4G$, $3G$, and $2G$ from the lowest resolution to the highest resolution, respectively ($c = 64$, $G = 8$).

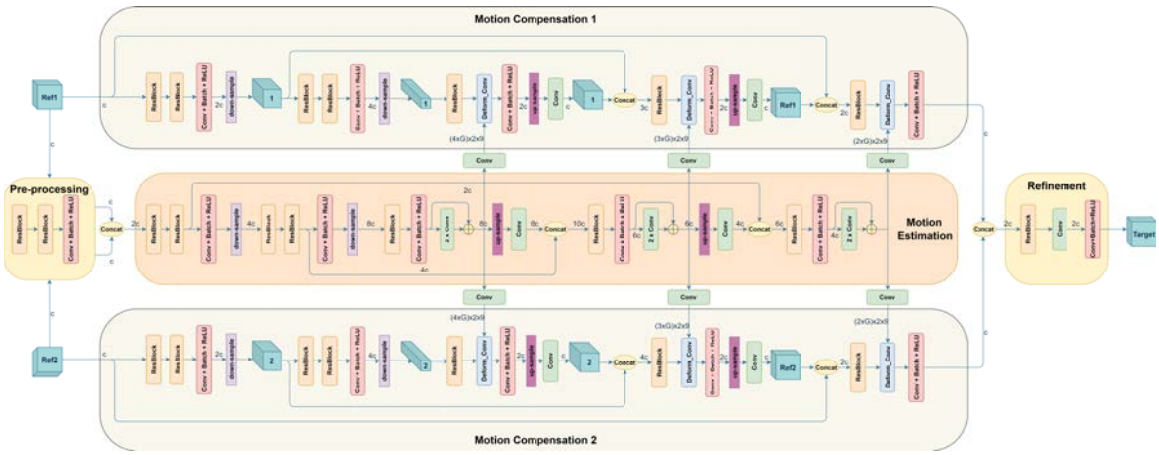


Figure 3.15: The architecture of Model-3

We followed the same procedure as Model-2 and trained the model for 20 epochs. The best MAE and MSE on the Vimeo-90k test set for Model-3 are 0.1564 and 0.04186, respectively, which is slightly worse than Model-2 but still better than Model-1.

3.4.3 Experimental Results

At first, we subjectively compare the performance of the three models in detecting motion by visualizing the offset fields of deformable convolution layers. As depicted in Fig. 3.11, 9 different motion vectors exist for each pixel in the output features. Here, we only show the motion vectors corresponding to the center point among those 9. So, there would be G different channels of motion vectors (G is the number of groups) for each deformable convolution layer, with a resolution equal to the output channel resolution. The visualized results are based on a test sequence shown in Fig. 3.16 using a Hue - Magnitude color map. In this test sequence, the car is moving on the track.

As seen in Fig. 3.17, there are 64 different groups for each layer of Model-1 with deformable convolutions. In layer 1, the object's boundary is visible, and it captures the motion to some extent. On the contrary, in layers 2 and 3, the offset fields are flat and have very

Figure 3.16: A test image

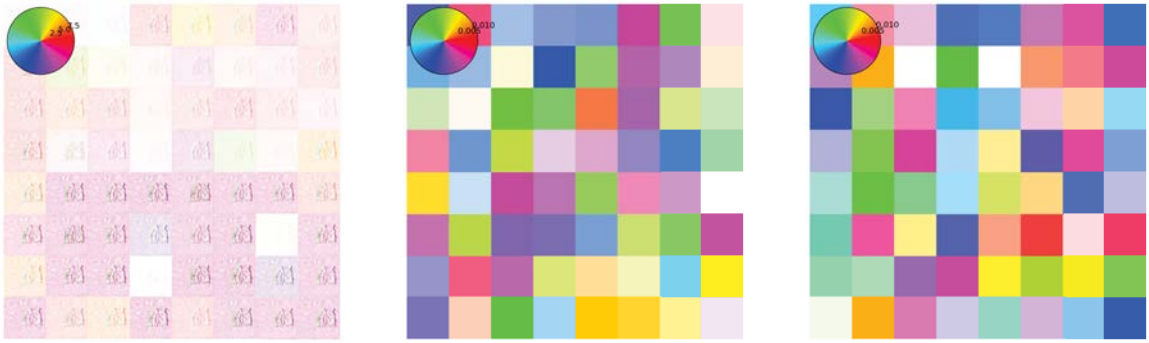
small values. Therefore, it can be concluded that layers 2 and 3 are not doing much for motion compensation. This phenomenon could be due to the fact that the motion estimator part of the deformable convolutions in layers 2 and 3 is not fed by the inputs directly.

On the other hand, both layers of deformable convolutions are able to capture the motion in Model-2 (Fig. 3.18). This is the advantage of having a separate module for motion estimation. This figure shows only the offset map of Ref2 (the bottom branch of Fig. 3.14). Note that the first and second layers have 8 (G) and 16 ($2G$) groups, respectively, for the deformable convolutions.

In Model-3, all three convolution layers also do motion compensation, and the offset fields look reasonable, as shown in Fig. 3.19. According to these offset maps, it can be inferred that the last layer is more responsible for capturing the moving object's motion. The number of deformable convolution groups for Model-3 is 32, 24, and 16 for layers 1, 2, and 3, respectively. These figures are also for Ref2's motion fields only.

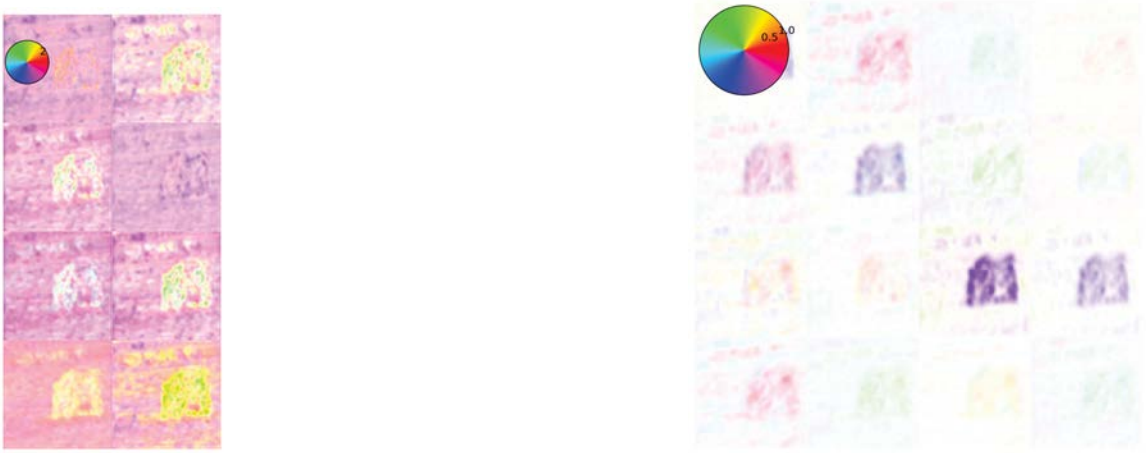
To compare the quality of the predicted tensors generated by the three models, a single channel of the latent space is shown in Fig. 3.20. As can be seen, the channel predicted by Model-1 is the most blurred channel among the three models, and the Model-3 prediction seems to produce the sharpest channel. Also, looking at the channels in sequence, we can infer that the channel generated by Model-3 is the most accurate in terms of motion compensation. To better demonstrate the performance of motion compensation, we run the decoder of the autoencoder and the YOLOv5 back-end on the predicted bottlenecks to see the locations of bounding boxes.

As seen in Fig. 3.21, the closest car bounding box to the ground-truth among the models is for Model-3, indicating an accurate location for the car in the predicted tensors. Also, the confidence score of the car in Model-3 output is exactly the same (0.82) as the ground truth, while it is lower (0.76 and 0.80) for the other models. The sharpness of the predicted feature channels in Model-3 could be a possible reason for that.



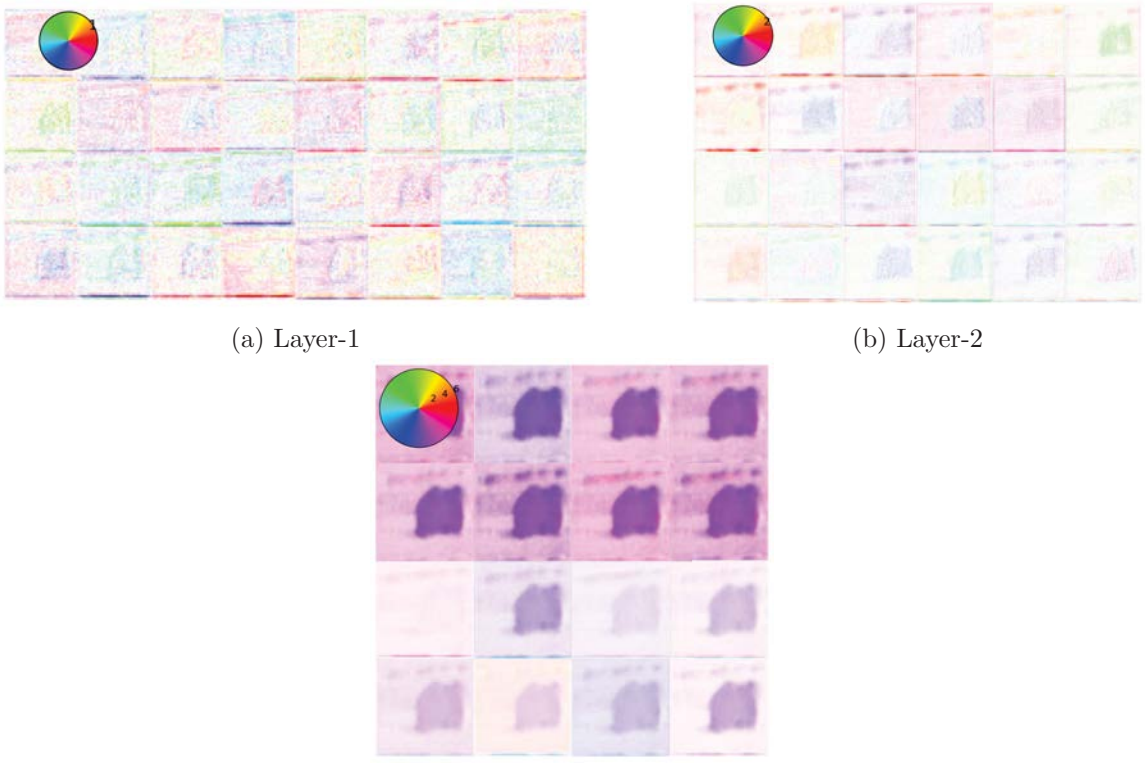
(a) Layer-1 (b) Layer-2 (c) Layer-3

Figure 3.17: Offset field visualization for Model-1



(a) Layer-1 (b) Layer-2

Figure 3.18: Offset field visualization for Model-2



(a) Layer-1 (b) Layer-2

(c) Layer-3

Figure 3.19: Offset field visualization for Model-3

(a) Model-1

(b) Model-2

(c) Model-3

(d) ground truth

Figure 3.20: Visualization of a channel in the latent space and its predicted versions generated by the DNN models

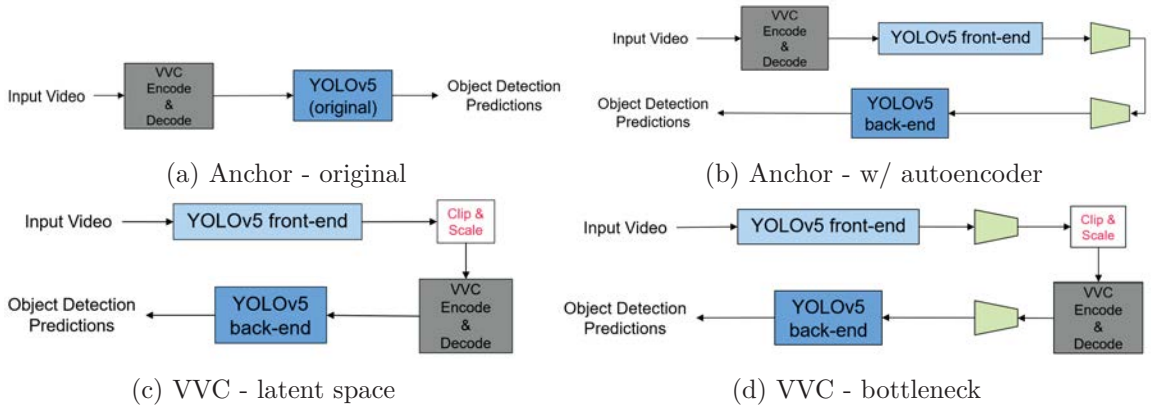
(a) Model-1

(b) Model-2

(c) Model-3

(d) ground truth

Figure 3.21: Object detection annotations on the predicted and ground truth tensors



(a) Anchor - original

(b) Anchor - w/ autoencoder

(c) VVC - latent space

(d) VVC - bottleneck

Figure 3.22: The benchmark pipelines

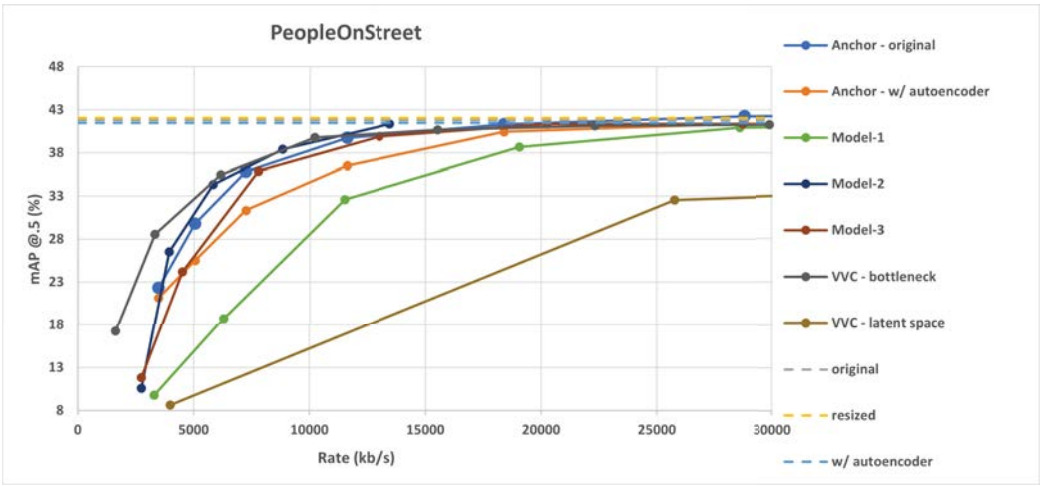


Figure 3.23: rate-accuracy curve for PeopleOnStreet

In the end, we should examine the performance of the models in a closed-loop manner and compare them with some benchmarks. These benchmarks include four pipelines depicted in Fig. 3.22. In all of these pipelines, the input video has been resized to the resolution of 1024×1024. Accordingly, the resolution of the channels in the latent space (or the bottleneck) would be 128×128. The proposed models are also examined in the pipeline shown in Fig. 3.7, and each residual frame is encoded by VVC as an I-frame.

The rate-accuracy curves for several test sequences are given in Fig. 3.23 to Fig. 3.27. In these figures, three horizontal dashed lines correspond to the cases without compression. "Original" is the mAP of the original YOLOv5x6 model on the native resolution of the input videos, "resized" is the mAP of the original YOLOv5x6 model on the resized resolution of the input videos to 1024×1024, and "w/ autoencoder" is the mAP of the retrained YOLOv5x6 (back-end) with the autoencoder inserted into the model on the input videos resized to 1024×1024.

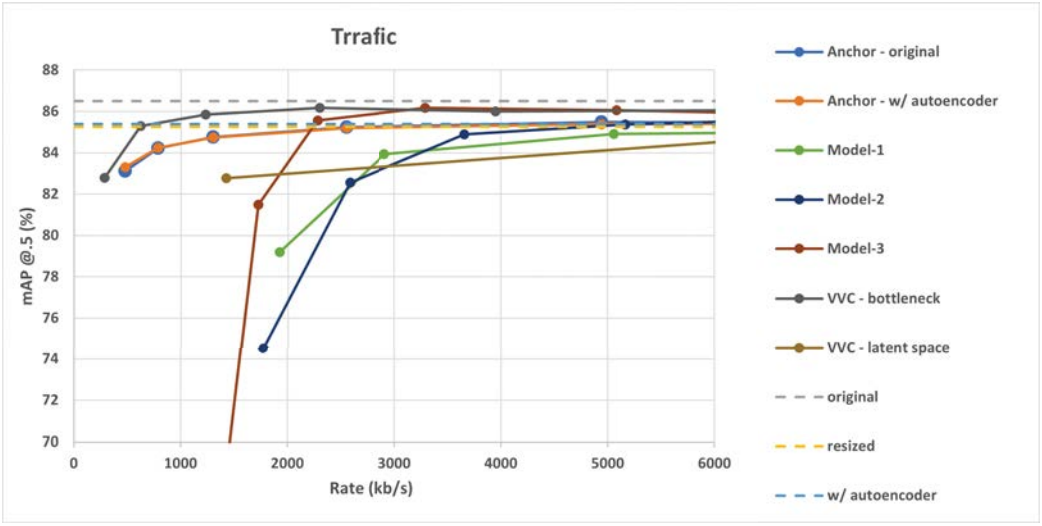


Figure 3.24: rate-accuracy curve for Traffic

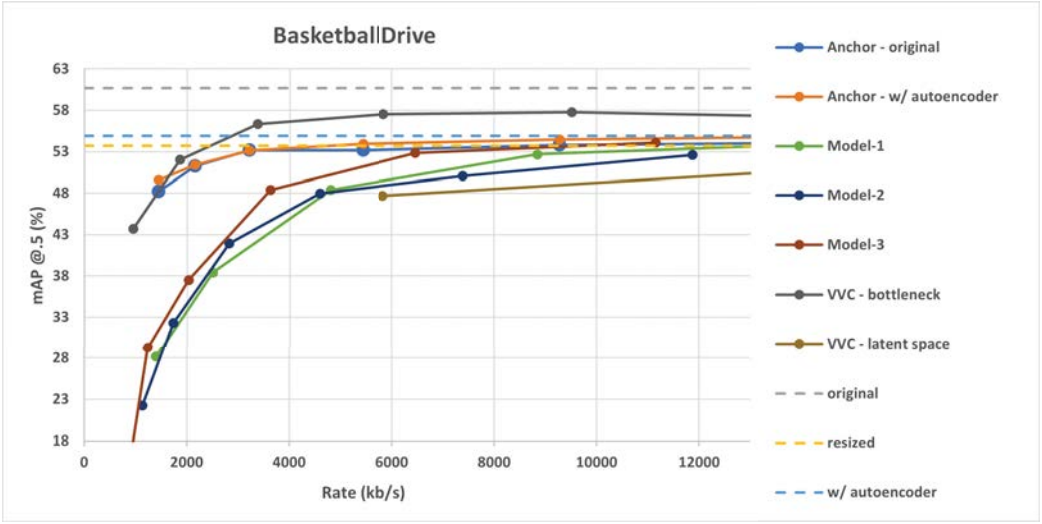


Figure 3.25: rate-accuracy curve for BasketballDrive

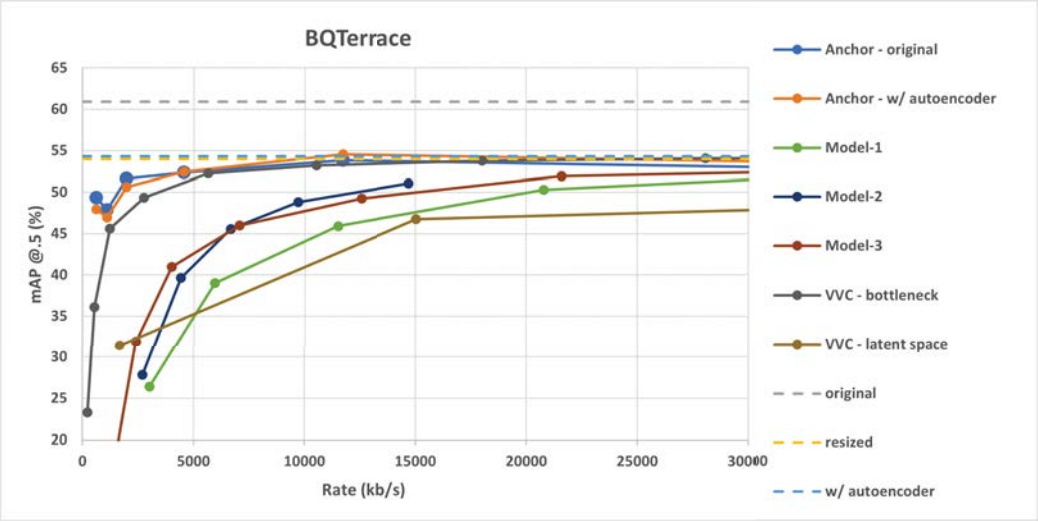


Figure 3.26: rate-accuracy curve for BQTerrace

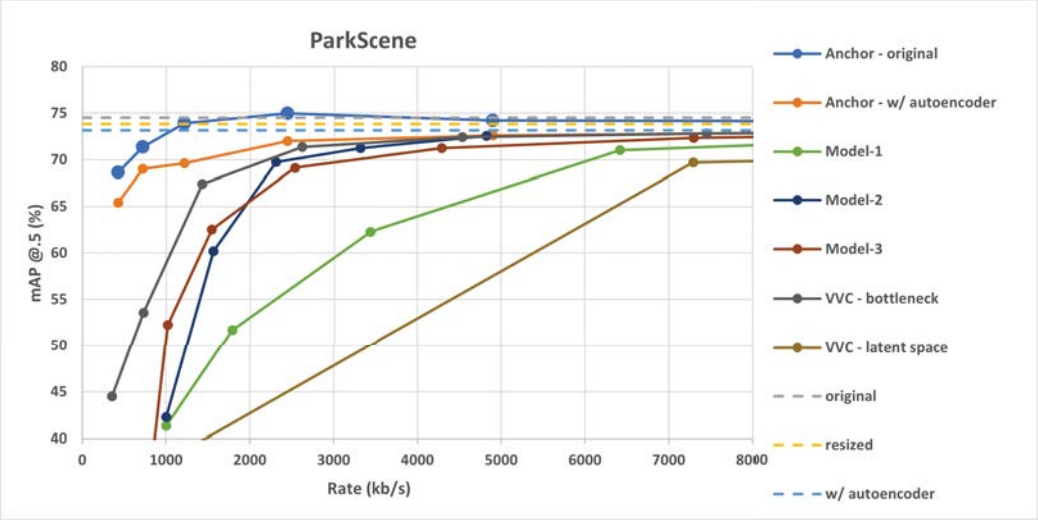


Figure 3.27: rate-accuracy curve for ParkScene

Table 3.11: The average BD-rate and BD-mAP over five test sequences for the available methods with respect to "Anchor - original"

Method	based on mAP@.5 (%)		based on mAP@.5:.95 (%)	
	BD-Rate	BD-mAP	BD-Rate	BD-mAP
Anchor - original	0.0%	0.0	0.0%	0.0
Anchor - w/ autoencoder	21.7%	-1.0	20.0%	-0.5
Model-1	690.6%	-8.2	576.2%	-4.1
Model-2	495.0%	-5.7	310.7%	-3.7
Model-3	431.8%	-4.2	306.6%	-2.0
VVC - bottleneck	64.6%	-0.1	20.0%	0.3
VVC - latent space	1607.8%	-12.6	875.7%	-6.6

The average BD-rate and BD-mAP between the presented curves for the proposed methods and the benchmarks are provided in Table 3.11. According to the figures and this table, using the autoencoder in the latent space and decreasing the number of channels in the bottleneck significantly reduces the bitrate needed for coding the feature channels. Note that "VVC-latent space," which codes the entire latent space (without the autoencoder) by VVC low-delay, has by far the worst performance. As mentioned before, excessively shrinking the latent space could increase the vulnerability of the object detection model to the compression artifacts in the decoded channels. As expected, Model-3 is the best among the proposed DNN models. However, its performance is still worse than the benchmark pipelines. "VVC-bottleneck" is the best model for compressing the feature channels but still worse than "Anchor-original." The main advantage of the "VVC-bottleneck" over the DNN-based models is the use of motion information (motion vectors) associated with the current frame. The performance of the DNN-based inter-prediction methods can be improved by incorporating the to-bo-coded motion information, which would then also need to be coded in addition to prediction residuals.

3.5 Conclusion

This chapter introduced several DNN-based models for inter-prediction in the latent space of the YOLOv5 object detection model. The last model (Model-3) was quite accurate in motion compensation. However, the rate-accuracy results showed that VVC is still a better codec, even in latent-space video coding. Part of it is due to the way that motion is handled. Our DNN-based predictors avoid coding motion but need to predict it, which may lead to lower prediction accuracy and higher residual to be coded. The other reason is that the prediction residuals are still coded using VVC-Intra, which is likely sub-optimal for coding latent-space information. Therefore, a better pipeline could be an end-to-end video codec with an object detection loss function that employs trainable motion estimation/compensation corresponding to the current frame as well.

Chapter 4

Conclusions and Future Work

In this thesis, we studied feature coding for machines in a Collaborative Intelligence framework. Our focus was on visual data and the existing challenges in coding for Deep Neural Network-based computer vision models. Some of these challenges addressed in the thesis include privacy and temporal prediction in the latent space.

In particular, an image coding scheme for an object detection model was presented in Chapter 2 that protects the transmitted data from model inversion attacks. In other words, the encoded bitstream was generated in a way to serve an object detection model on the cloud and prevent any adversary from recovering the input precisely. This was done through an adversarial training technique to modify the bottleneck feature channels such that private information would be removed. The results of our proposed method showed that the overall system is able to outperform anchors, like input compression, in terms of compression efficiency, as well as disrupt the adversary's job, which is running a model inversion attack.

In our experiments, the quality degradation of the recovered input image from the bottleneck data was considered a metric for privacy. However, privacy could be defined in a more effective way, as it is essentially related to concepts like anonymity. Therefore, devising a new metric for measuring privacy and applying it to privacy-preserving coding methods could be a possible direction for future research. For example, the privacy of a system could be measured by the accuracy of a face recognition model on the reconstructed images.

In our proposed method, we used a loss term encouraging the auxiliary model (RecNet) to reconstruct the input image with an emphasis on the edges, which consequently imposes more distortion on the edges of the recovered input. That was done because we believed private information is closely interwoven with the edges. Nevertheless, all the edges are not equally important. Thus, more effective loss functions targeting privacy directly could be studied in future works.

Temporal prediction in the latent space is another important challenge in visual coding for machines. Learning-based video coding is a hot research topic these days. Despite the great advances of end-to-end video codecs, they are still not successful in competition with

the state-of-the-art conventional video coding standard, H.266/VVC. Since inter-prediction is the most contributing factor in video compression efficiency, trainable motion estimation and compensation models have the potential to become more optimal.

Studying the motion relationship between the input and latent space of an object detection model in Chapter 3, we realized that the expected motion is not mostly captured by traditional video codecs. Then, we proposed several trainable temporal prediction models based on deformable convolutional layers. The visual results demonstrated that at least one of these models is satisfying in motion compensation in spite of the quantitative results. Our model's compression efficiency was not good enough to outperform the corresponding anchors utilizing VVC as a video encoder.

One potential problem could be the inputs to our proposed models. The only inputs are two previous tensor frames based on which the current tensors should be predicted. In that case, no information from the current frame is used for making predictions. However, both conventional and most emerging differentiable video codecs employ such information as motion vectors that also must be coded. Accordingly, considering additional inputs to the model could be helpful and might improve the overall performance of the models in the future.

In both projects studied in this thesis, we employed an autoencoder mainly to reduce the volume of the bottleneck data. We avoided training the autoencoder together with the main object detection model for lower complexity. However, joint end-to-end training might be able to improve the overall performance of the proposed methods and their corresponding computer vision models.

Studying more efficient ways of coding the bottleneck feature channels could be another future direction of this research. In this thesis, we coded the bottleneck features by tiling their channels into a gray-scale image on which an H.266/VVC encoder was executed. Putting the feature channels into the tiled tensor image in a systematic order can help better compression. Also, adopting strategies other than tiling can reduce the compression non-optimality, especially near the channels' boundary. One of these strategies is Multi-view video/image coding [95], which treats each channel as an individual view of the same sequence. In this method, inter-view prediction alongside the intra and inter-prediction is utilized to remove the redundancies existing between different views (or channels) as well.

Also, applying the proposed methodologies on an end-to-end trainable pipeline could be another promising direction for future research, since the end-to-end neural networks are remarkably flexible and can be optimized for different purposes, such as compression efficiency or privacy.

Bibliography

[1] Eirikur Agustsson, David Minnen, Nick Johnston, Johannes Balle, Sung Jin Hwang, and George Toderici. Scale-space flow for end-to-end optimized video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[2] Sangsoo Ahn, Bumshik Lee, and Munchurl Kim. A novel fast cu encoding scheme based on spatiotemporal encoding parameters for hevc inter coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(3):422–435, 2015.

[3] Md Mushfiqul Alam, Tuan D. Nguyen, Martin T. Hagan, and Damon M. Chandler. A perceptual quantization strategy for HEVC based on a convolutional neural network trained on natural images. In Andrew G. Tescher, editor, *Applications of Digital Image Processing XXXVIII*, volume 9599, pages 395 – 408. International Society for Optics and Photonics, SPIE, 2015.

[4] Saeed Ranjbar Alvar and Ivan V. Bajić. Multi-task learning with compressible features for collaborative intelligence. In *Proc. IEEE ICIP*, pages 1705–1709, 2019.

[5] Saeed Ranjbar Alvar and Ivan V. Bajić. Scalable privacy in multi-task image compression. In *Proc. IEEE VCIP*, pages 1–5, 2021.

[6] Bardia Azizian and Ivan V. Bajić. Privacy-preserving feature coding for machines. *arXiv:2210.00727*, 2022.

[7] Ivan V. Bajić, Weisi Lin, and Yonghong Tian. Collaborative intelligence: Challenges and opportunities. In *Proc. IEEE ICASSP*, pages 8493–8497, 2021.

[8] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. *arXiv:1611.01704*, 2016.

[9] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *Proc. ICLR*, 2018.

[10] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud ai. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 2543–2553, New York, NY, USA, 2021. Association for Computing Machinery.

[11] Wenbo Bao, Wei-Sheng Lai, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Memc-net: Motion estimation and motion compensation driven neural network for

video interpolation and enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(3):933–948, 2021.

[12] Raz Birman, Yoram Segal, and Ofer Hadar. Overview of research in the field of video compression using deep neural networks. *Multimedia Tools and Applications*, pages 1–24, 2020.

[13] Gisle Bjøntegaard. Calculation of average psnr differences between rd-curves. In *VCEG Meeting (ITU-T SG16 Q.6)*, 2001. VCEG-M33.

[14] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv:2004.10934*, 2020.

[15] Benjamin Bross, Jianle Chen, Jens-Rainer Ohm, Gary J. Sullivan, and Ye-Kui Wang. Developments in international video coding standardization after avc, with an overview of versatile video coding (vvc). *Proceedings of the IEEE*, pages 1–31, 2021.

[16] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J. Sullivan, and Jens-Rainer Ohm. Overview of the versatile video coding (vvc) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021.

[17] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proc. IEEE/CVF CVPR*, June 2020.

[18] Hyomin Choi and Ivan V. Bajić. Deep feature compression for collaborative object detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 3743–3747, 2018.

[19] Hyomin Choi and Ivan V. Bajić. Deep frame prediction for video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(7):1843–1855, 2020.

[20] Hyomin Choi and Ivan V. Bajić. Affine transformation-based deep frame prediction. *IEEE Transactions on Image Processing*, 30:3321–3334, 2021.

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

[22] Hyomin Choi, Elahe Hosseini, Saeed Ranjbar Alvar, Robert Cohen, and Ivan Bajić. Sfu-hw-objects-v1: Object labelled dataset on raw video sequences. 10.25314/7d8efc0a-3943-4738-b7a5-72badb04d765, 2020.

[23] Cisco. Cisco annual Internet report (2018–2023), Mar. 2020.

[24] Robert A. Cohen, Hyomin Choi, and Ivan V. Bajić. Lightweight compression of intermediate neural network features for collaborative intelligence. *IEEE Open J. Circuits Syst.*, 2:350–362, 2021.

[25] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006.

[26] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[27] Abdelaziz Djelouah, Joaquim Campos, Simone Schaub-Meyer, and Christopher Schroers. Neural inter-frame compression for video coding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[28] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[29] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 20(2):565–576, 2021.

[30] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *Proc. IEEE/ACM ISLPED*, pages 1–6, 2019.

[31] Amir Erfan Eshratifar and Massoud Pedram. Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, GLSVLSI '18, page 111–116, New York, NY, USA, 2018. Association for Computing Machinery.

[32] Omobayode Fagbohungbe, Sheikh Rufsan Reza, Xishuang Dong, and Lijun Qian. Efficient privacy preserving edge intelligent computing framework for image classification in iot. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(4):941–956, 2022.

[33] Wen Gao, Shan Liu, Xiaozhong Xu, Manouchehr Rafie, Yuan Zhang, and Igor Curcio. Recent standard development activities on video coding for machines. *arXiv:2105.12653*, 2021.

[34] Xuan Gong, Liangchen Song, Rishi Vedula, Abhishek Sharma, Meng Zheng, Benjamin Planche, Arun Innanje, Terrence Chen, Junsong Yuan, David Doermann, and Ziyan Wu. Federated learning with privacy-preserving ensemble attention distillation. *IEEE Transactions on Medical Imaging*, pages 1–1, 2022.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. NeurIPS*, 2014.

[36] Onur G. Guleryuz, Philip A. Chou, Hugues Hoppe, Danhang Tang, Ruofei Du, Philip Davidson, and Sean Fanello. Sandwiched image compression: Wrapping neural networks around a standard codec. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3757–3761, 2021.

[37] Onur G. Guleryuz, Philip A. Chou, Hugues Hoppe, Danhang Tang, Ruofei Du, Philip Davidson, and Sean Fanello. Sandwiched image compression: Increasing the resolution

and dynamic range of standard codecs. In *2022 Picture Coding Symposium (PCS)*, 2022.

[38] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In *Proc. 35th Annual Computer Security Applications Conference*, page 148–162, 2019.

[40] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet of Things Journal*, 8(12):9706–9716, 2021.

[41] C Hollmann, S Liu, W Gao, and X Xu. [VCM] on VCM reporting template. ISO/IEC JTC 1/SC 29/WG2 M56185, Jan. 2021.

[42] Zhihao Hu, Guo Lu, and Dong Xu. Fvc: A new framework towards deep video compression in feature space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1502–1511, June 2021.

[43] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[44] ISO/IEC JTC 1/SC29/WG1. Final call for proposals for JPEG AI, Jan. 2022. N100095.

[45] Ismat Jarin and Birhanu Eshete. Pricure: Privacy-preserving collaborative inference in a multi-party setting. In *Proceedings of the 2021 ACM Workshop on Security and Privacy Analytics*, IWSPA '21, page 25–35, New York, NY, USA, 2021. Association for Computing Machinery.

[46] HEVC reference software. `http://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware`.

[47] HEVC reference software (HM 16.20). `http://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.20+SCM-8.8`. Accessed: 2019-12-12.

[48] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, February 2022.

[49] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGARCH Comput. Archit. News*, 45(1):615–629, apr 2017.

[50] Jaehwan Kim, Jungyoup Yang, Kwanghyun Won, and Byeungwoo Jeon. Early determination of mode decision for hevc. In *2012 Picture Coding Symposium*, pages 449–452, 2012.

[51] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Phys. Rev. E*, 69, Jun 2004.

[52] Nam Le, Honglei Zhang, Francesco Cricri, Ramin Ghaznavi-Youvalari, and Esa Rahtu. Image coding for machines: an end-to-end learned approach. In *Proc. IEEE ICASSP*, pages 1590–1594, 2021.

[53] Jie Leng, Lei Sun, Takeshi Ikenaga, and Shinichi Sakaida. Content based hierarchical fast coding unit decision algorithm for hevc. In *2011 International Conference on Multimedia and Signal Processing*, volume 1, pages 56–59, 2011.

[54] Jiahao Li, Bin Li, Jizheng Xu, Ruiqin Xiong, and Wen Gao. Fully connected network-based intra prediction for image coding. *IEEE Transactions on Image Processing*, 27(7):3236–3247, 2018.

[55] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conf. on Computer Vision (ECCV)*, Sept. 2014.

[56] Weisi Lin and C.-C. Jay Kuo. Perceptual visual quality metrics: A survey. *J. Visual Commun. Image Represent.*, 22(4):297–312, 2011.

[57] Haojie Liu, Ming Lu, Zhan Ma, Fan Wang, Zhihuang Xie, Xun Cao, and Yao Wang. Neural video coding using multiscale motion compensation and spatiotemporal context model. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(8):3182–3196, 2021.

[58] Ximeng Liu, Lehui Xie, Yaopeng Wang, Jian Zou, Jinbo Xiong, Zuobin Ying, and Athanasios V. Vasilakos. Privacy and security issues in deep learning: A survey. *IEEE Access*, 9:4566–4593, 2021.

[59] Zhijian Liu, Zhanghao Wu, Chuang Gan, Ligeng Zhu, and Song Han. Datamix: Efficient privacy-preserving edge-cloud inference. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 578–595, Cham, 2020. Springer International Publishing.

[60] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[61] Guo Lu, Xiaoyun Zhang, Wanli Ouyang, Li Chen, Zhiyong Gao, and Dong Xu. An end-to-end learning framework for video compression. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3292—3308, October 2021.

[62] Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2020.

[63] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[64] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *NeurIPS*, 2018.

[65] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Praneeth Vepakomma, Abhishek Singh, Ramesh Raskar, and Hadi Esmaeilzadeh. Privacy in deep learning: A survey. *arXiv:2004.12254*, 2020.

[66] MPEG-CDVA. Compact descriptors for video analysis, 2019. ISO/IEC JTC 1 15938-15.

[67] MPEG-CDVS. Compact descriptors for visual search, 2015. ISO/IEC JTC 1 15938-13.

[68] Luis Muñoz-González, Bjarne Pfitzner, Matteo Russo, Javier Carnerero-Cano, and Emil C. Lupu. Poisoning attacks with generative adversarial nets. *arXiv:1906.07773*, 2019.

[69] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[70] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[71] Zhaoqing Pan, Sam Kwong, Ming-Ting Sun, and Jianjun Lei. Early merge mode decision based on motion estimation and hierarchical depth correlation for hevc. *IEEE Transactions on Broadcasting*, 60(2):405–412, 2014.

[72] Pallab Kanti Podder, Manoranjan Paul, Manzur Murshed, and Subrata Chakraborty. Fast intermode selection for hevc video coding using phase correlation. In *2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, 2014.

[73] Anurag Ranjan and Michael J. Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[74] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018.

[75] Chae Eun Rhee, Kyujoong Lee, Tae Sung Kim, and Hyuk-Jae Lee. A survey of fast mode decision algorithms for inter-prediction and their applications to high efficiency video coding. *IEEE Transactions on Consumer Electronics*, 58(4):1375–1383, 2012.

[76] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G. Anderson, and Lubomir Bourdev. Learned video compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[77] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[78] Jihyeon Ryu, Yifeng Zheng, Yansong Gao, Sharif Abuadbba, Junyaup Kim, Dongho Won, Surya Nepal, Hyoungshick Kim, and Cong Wang. Can differential privacy practically protect collaborative deep learning inference for the internet of things? *arXiv:2104.03813*, 2021.

[79] Amir Said, Manish K. Singh, and Reza Pourreza. Differentiable bit-rate estimation for neural-based video codec enhancement. In *2022 Picture Coding Symposium (PCS)*, 2022.

[80] Felipe Sampaio, Sergio Bampi, Mateus Grellert, Luciano Agostini, and Júlio Mattos. Motion vectors merging: Low complexity prediction unit decision heuristic for the inter-prediction of hevc encoders. In *2012 IEEE International Conference on Multimedia and Expo*, pages 657–662, 2012.

[81] Liquan Shen, Zhi Liu, Xinpeng Zhang, Wenqiang Zhao, and Zhaoyang Zhang. An effective cu size decision method for hevc encoders. *IEEE Transactions on Multimedia*, 15(2):465–470, 2013.

[82] Zhihao Shi, Xiaohong Liu, Kangdi Shi, Linhui Dai, and Jun Chen. Video frame interpolation via generalized deformable convolution. *IEEE Transactions on Multimedia*, 24:426–439, 2022.

[83] Kyung-Ah Shim. A survey of public-key cryptographic primitives in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 18(1):577–601, 2016.

[84] Nir Shlezinger and Ivan V. Bajic. Collaborative inference for ai-empowered iot devices. *arXiv:2207.11664*, 2022.

[85] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1310–1321, New York, NY, USA, 2015. Association for Computing Machinery.

[86] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.

[87] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[88] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. *arXiv:1709.02371*, 2017.

[89] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.

[90] Hui Li Tan, Fengjiao Liu, Yih Han Tan, and Chuohao Yeo. On fast coding tree block and mode decision for high-efficiency video coding (hevc). In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 825–828, 2012.

[91] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 402–419, Cham, 2020. Springer International Publishing.

[92] Mateen Ulhaq and Ivan V. Bajić. Latent space motion analysis for collaborative intelligence. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8498–8502, 2021.

[93] Unknown. darknet. [Online]: `https://github.com/AlexeyAB/darknet`.

[94] Jarno Vanne, Marko Viitanen, and Timo D. Hämäläinen. Efficient mode decision schemes for hevc inter prediction. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(9):1579–1593, 2014.

[95] Anthony Vetro, Thomas Wiegand, and Gary J. Sullivan. Overview of the stereo and multiview video coding extensions of the h.264/mpeg-4 avc standard. *Proceedings of the IEEE*, 99(4):626–642, 2011.

[96] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S. Yu. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2407–2416, New York, NY, USA, 2018. Association for Computing Machinery.

[97] Adam Wieckowski, Jens Brandenburg, Tobias Hinz, Christian Bartnik, Valeri George, Gabriel Hege, Christian Helmrich, Anastasia Henkel, Christian Lehmann, Christian Stoffers, Ivan Zupancic, Benjamin Bross, and Detlev Marpe. VVenC: an open and optimized VVC encoder implementation. In *Proc. IEEE ICME Workshops*, pages 1–2, 2021.

[98] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[99] Mathias Wien. High efficiency video coding. *Coding Tools and specification*, 24, 2015.

[100] Chao-Yuan Wu, Nayan Singhal, and Philipp Krahenbuhl. Video compression through image interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[101] Jian Xiong, Hongliang Li, Fanman Meng, Qingbo Wu, and King Ngi Ngan. Fast hevc inter cu decision based on latent sad estimation. *IEEE Transactions on Multimedia*, 17(12):2147–2159, 2015.

[102] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127(8):1106–1125, 2019.

[103] Ning Yan, Dong Liu, Houqiang Li, Bin Li, Li Li, and Feng Wu. Convolutional neural network-based fractional-pixel motion compensation. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(3):840–853, 2019.

[104] Zhongzheng Yuan, Samyak Rawlekar, Siddharth Garg, Elza Erkip, and Yao Wang. Feature compression for rate constrained object detection on the edge. *arXiv:2204.07314*, 2022.

[105] Jinlei Zhang, Bin Li, and Houqiang Li. An efficient fast mode decision method for inter prediction in hevc. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(8):1502–1515, 2016.

[106] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2017.

[107] Lei Zhao, Shiqi Wang, Xinfeng Zhang, Shanshe Wang, Siwei Ma, and Wen Gao. Enhanced ctu-level inter prediction with deep frame rate up-conversion for high efficiency video coding. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 206–210, 2018.

[108] Lei Zhao, Shiqi Wang, Xinfeng Zhang, Shanshe Wang, Siwei Ma, and Wen Gao. Enhanced motion-compensated video coding with deep virtual reference frame generation. *IEEE Transactions on Image Processing*, 28(10):4832–4844, 2019.

[109] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.