Joint motion and residual information latent representation for P-frame coding

Renam Castro da Silva[†], Nilson Donizete Guerin Jr.[⋆], Pedro Sanches[⋆], Henrique Costa Jung[⋆], Eduardo Peixoto[⋆], Bruno Macchiavello[⋆], Edson M. Hung[⋆], Vanessa Testoni[†], Pedro Garcia Freitas[†]
[⋆]Universidade de Brasília, [†]Samsung R&D Institute Brazil
renam.cs@samsung.com, nguerinjr@aluno.unb.br

Abstract

This paper proposes an inter-frame prediction frame encoding for the P-frame video compression challenge of the Workshop and Challenge on Learned Image Compression (CLIC). For this challenge, we use an uncompressed reference (previous) frame to compress the current frame. So, this is not a complete solution for learning-based video compression. The main goal is to represent a set of frames with an average of 0.075 bpp (bits per pixel), which is a very low bitrate. A restriction on the model size is also requested to avoid overfitting. Here we propose an autoencoder architecture that jointly represents the motion and residue information at the latent space. Three trained models were used to achieve the target bpp and a bit allocation algorithm is also proposed to optimize the quality performance of the encoded dataset.

1. Introduction

A conventional digital video is a set of digital images that are periodically captured over time giving rise to the set of so-called video frames. In order to efficiently compress the video frames, traditional video encoders exploit temporal, spatial, and statistical redundancies in the video sequence. Temporal redundancy is associated with inter-frame coding and considers that video frames closer in time are more similar. Spatial redundancy, associated with intra-frame coding, means that nearby pixels within a frame are often similar, and statistical redundancy is associated with a general similarity in the data.

Although a complete video decoder needs to handle these three types of redundancy, our focus in this paper is to exploit temporal redundancy to achieve inter-frame compression. Inter-frame coding can be further classified into B-frame compression and P-frame compression. P-frame compression uses only frames that should be displayed previously in the video, but B-frames may use both previous and posterior frames to achieve better compression, as video frames do not need to be compressed in the same order as they are displayed.

This paper proposes a prediction-based inter-frame coding method for the P-frame video compression challenge of the Workshop and Challenge on Learned Image Compression (CLIC). For this challenge, we use an uncompressed reference (previous) frame to compress the current frame. Hence, this is not a complete solution for learning-based video compression but is focused only on inter-frame and data redundancy. The main goal is to represent a set of frames with an average of 0.075 bpp, which is a very low bitrate. Instead of splitting the dataset into training and test sets, in this challenge, the entire dataset is released before the test phase. To discourage overfitting, the model size is added to the compressed dataset size and the sum cannot exceed a target bitrate. Therefore, participants should try to minimize both dataset and model sizes. In the validation and test phases, the CLIC's organizers evaluate the submissions on a randomly picked subset of the dataset. Details on how to achieve the target compression rate are detailed in Section 2.4.

2. Proposed P-frame codec

An overview of our solution submitted to the P-frame track of the CLIC is schematically shown in Figure 1. It is comprised of stacked convolutional layers as detailed in Tables 1 and 2. Given the frame to be coded $\mathbf{x}_i$ and its reference (previous) frame $\mathbf{x}_{i-1}$, the *Encoder* is assigned the task of producing a latent representation $\mathbf{y}$ conveying both motion information for inter-frame prediction and residual information error between the raw frame being coded $\mathbf{x}_i$ and the warped previous frame $W(\mathbf{x}_{i-1};\boldsymbol{\theta})$. The latent representation $\mathbf{y}$ undergoes a first step of joint processing in the *Common trunk decoder*, and the resulting feature maps are then fed to the *Motion decoder* network and to the *Residue decoder* network. The *Motion decoder* network aims at producing the parameters $\boldsymbol{\theta}$ of an affine transformation $W(\cdot,\boldsymbol{\theta})$ which is applied to previous frame $\mathbf{x}_{i-1}$ to produce a prediction $\mathbf{x}_{i,pred}$, whereas the *Residue decoder* targets to decode the residue $\mathbf{r}_i$ to be added to the mentioned prediction, and in turn producing the decoded frame $\mathbf{x}_{i,dec}$. This residue conveys the novelty within the frame $\mathbf{x}_i$ that could not be inferred from $\mathbf{x}_{i-1}$.

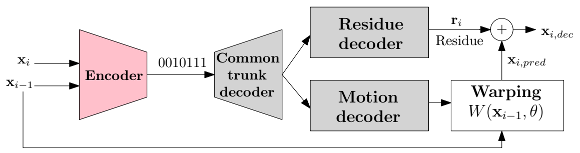

Figure 1. Overall model architecture

2.1. Encoder and common trunk decoder

The *Encoder* and the *Common trunk decoder* employ a standard autoencoder architecture, leveraged from the architecture proposed in [1], schematically shown in Figure 2), and detailed in Table 1. However, this architecture was modified to jointly encode both motion and residual information at the latent representation. The motion information is used to perform pixel-wise motion compensation with a spatial transformer network as described in Section 2.2. The decoder reconstructs the motion vector and warps the reference frame to obtain the predicted frame. The residual information is decoded and added to the predicted frame to get the reconstructed frame.

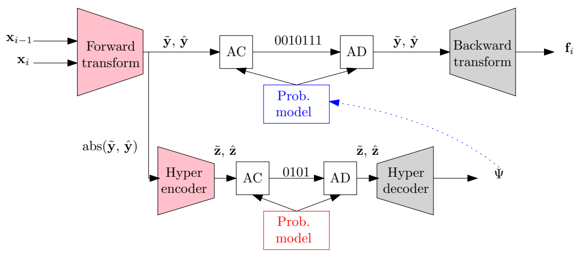
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
Forward transform
$\tilde{\mathbf{y}}, \hat{\mathbf{y}}$
AC
0010111
AD
$\tilde{\mathbf{y}}, \hat{\mathbf{y}}$
Backward transform
$\mathbf{f}_i$
Prob. model
abs($\tilde{\mathbf{y}}, \hat{\mathbf{y}}$)
$\tilde{\mathbf{z}}, \hat{\mathbf{z}}$
Hyper encoder
AC
0101
AD
$\tilde{\mathbf{z}}, \hat{\mathbf{z}}$
Hyper decoder
Ψ
Prob. model

Figure 2. Encoder/Common trunk decoder architecture. AC and AD stand for Arithmetic encoder and decoder, respectively.

2.2. Spatial transformer network

The spatial transformer network [2] is comprised of the blocks *Motion decoder* and *Warping* as shown in Figure 1. The *Motion decoder* is required to produce the parameters $\boldsymbol{\theta}$ of the warping $W(\cdot;\boldsymbol{\theta})$ applied to the previous frame. The warping function could be a more general affine transformation, but for this challenge, it was constrained to be a simple translation, thus $\boldsymbol{\theta} \in \mathbb{R}^{2\cdot w \cdot h}$ conveys the horizontal and vertical displacement of each pixel from the previous frame to the current frame, the same warping is applied to each color component. Following [2], each pixel in the warped output $\mathbf{x}_{i,pred}$ is computed by applying a sampling kernel centered at a particular location (x_{i-1}, y_{i-1}) in $\mathbf{x}_{i-1}$:

$$\begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} = W_\theta \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}, W_\theta = \begin{pmatrix} 1 & 0 & t_x \\ 1 & 0 & t_y \end{pmatrix} \quad (1)$$

Encoder	Common trunk decoder
Forward transform	**Backward transform**
C-5 × 5, ↓2, GDN, 192	TC-5 × 5, ↑2, IGDN, 192
C-5 × 5, ↓2, GDN, 192	TC-5 × 5, ↑2, IGDN, 192
C-5 × 5, ↓2, GDN, 192	TC-5 × 5, ↑2, IGDN, 192
C-5 × 5, ↓2, 192	TC-5 × 5, ↑2, 192
Hyper encoder	**Hyper decoder**
C-3 × 3, ReLU, 192	TC-5 × 5, ↑2, ReLU, 192
C-5 × 5, ↓2, ReLU, 192	TC-5 × 5, ↑2, ReLU, 192
C-5 × 5, ↓2, 192	TC-3 × 3, 192

Table 1. Encoder and decoder networks. In our notation, C-5×5, ↓2,GDN, 192 indicates a convolutional layer with 192 filters, each with spatial support 5 × 5, stride 2, and the Generalized Divisive Normalization (GDN) as activation function. Similarly, TC refers to transposed convolutional layer, IGDN to inverse GDN, and ReLU to the rectified linear unit.

Motion decoder	Residue decoder
C-5 × 5, ReLU, 32	C-5 × 5, ReLU, 32
C-1 × 1, 2	C-1 × 1, 3

Table 2. Motion decoder and residue decoder networks

In our notation, $\boldsymbol{\theta}$ gathers the displacement parameters t_x and t_y for the whole frame domain. For sampling, we adopt the differentiable sampling method described in [2] (bilinear interpolation).

2.3. Training procedure

Training a neural network involves using a dataset to update the model parameters to create a good mapping from inputs to outputs. When using autoencoders, this objective is translated to compress data and reconstruct it as good as possible. In this track of the challenge, it is required to compress a video frame conditioned on a previous frame.

The dataset used is the one CLIC organizers provided, where the training data is a subset of UGC Dataset, composed of 739 videos, with a total of $\sim$ 466684 frames. Each of these frames is represented by 3 PNG images, one for each channel of the YUV encoding, where the Y (luma) channel has twice the weight and twice the height of the others U and V color channels.

In the reported approach, based on autoencoders, a Lagrangian cost function is optimized. It takes into account not only the reconstruction distortion D of the frame being coded but also the rate R of the latent representation. Therefore, the loss is one in the form:

$$L = R + \lambda \times D \quad (2)$$

Adam was used as the default optimizer, and the training data was structured in batches of size 8. During training, patches (256×256) were extracted from the data.

A couple of models were trained by varying the Lagrangian multiplier λ, and all of them were optimized using the mean squared error (MSE) as the reconstruction metric. These models were tuned from two hundred thousand to 1 million iterations, depending on the observed convergence of the model in the course of training.

In order to discourage overfitting, one of the rules of the challenge imposes a maximum file size restriction on submission. This restriction considers both the trained models and the compressed dataset. Therefore, to fulfill task requirements, three models with λ values of 0.002, 0.01, and 0.05 were empirically chosen out of all that were trained.

2.4. Bit allocation

In order to allocate the bit budget to the frames in the validation/test dataset, it is employed a couple of trained models targeted at different rate-distortion trade-offs and a simple procedure described in Algorithm 1. For the P-frame track of the challenge, contestants are asked to allocate the given bit budget to encode the validation/test dataset and to represent the model according to the following rule:

$$ModelSize + 100 \cdot \sum_{j}^{|Dataset|} FileSize_j < K \qquad (3)$$

where $K = 3,900,000,000$ bytes and $|\cdot|$ means cardinality.

Assuming one decides to allocate the bit budget evenly over the frames of the validation/test set, the $FileSize$ would be required to be $< (K - ModelSize)/(100 \cdot |Dataset|)$.

Let $p = [q, b]$ be the tuple of arrays q (quality) and b (bitstream) associated to a given frame $\mathbf{x}_i$. The lengths of q and b are defined by the number of encoders used. Also, let $\mathcal{X}$ be a set of frames to be coded and its (reference) previous frame $\{\mathbf{x}_i, \mathbf{x}_{i-1}\}$. The tuples p associated to each frame $\mathbf{x}_i$ is initialized as zero.

The bit allocation procedure described in pseudocode in Algorithm 1 accomplishes its task in three steps. In the first step, each frame is encoded in *Compress* with all available encoders resulting in the arrays of quality and bitstream, each entry associated with an encoder. Those arrays are input to the *ConvexHullTarget* which outputs the index of the encoder lying in the upper convex hull with the highest bitstream length below the target $Filesize$. If the returned index is valid, the bitstream is written to file. Otherwise, the frame is let to be encoded in the second step. At the end of this first step, the easiest frames would be encoded, remaining only the frames demanding higher rates. The second step, therefore, is targeted at encoding the high rate frames by allowing bitstream lengths higher than the target $FileSize$ (*ConvexHull*). Also, in case there is budget, frames with low quality will have the chance to be ameliorated. The third step just to check whether $Budget$ has not changed, meaning that the available $Budget$ could not be used to improve any frame.

Algorithm 1: Bit allocation procedure

```
Data: A set $\mathcal{X}$ of frames pair $\{\mathbf{x}_i, \mathbf{x}_{i-1}\}$, $Budget$
      and $FileSize$
Result: A set of compressed bitstream
1  STEP 1 ;
2  for $\{\mathbf{x}_i, \mathbf{x}_{i-1}\}$ in $\mathcal{X}$ do
3      $p \leftarrow Compress(\mathbf{x}_i, \mathbf{x}_{i-1})$ ;
4      $i \leftarrow ConvexHullTarget(p, FileSize)$ ;
5      if $i$ is valid then
6          $WriteBitstream(p.b(i))$ ;
7          $Budget \leftarrow Budget - length(p.b(i))$
8  STEP 2 ;
9  $\mathcal{X} \leftarrow Sort(\mathcal{X})$ by increasing order of quality ;
10 for $\{\mathbf{x}_i, \mathbf{x}_{i-1}\}$ in $\mathcal{X}$ do
11     $i_t \leftarrow i$ ;
12     $i \leftarrow ConvexHull(p)$ ;
13     $Budget \leftarrow Budget + length(p.b(i_t))$ ;
14     if $length(p.b(i)) < Budget$ then
15         $WriteBitstream(p.b(i))$ ;
16         $Budget \leftarrow Budget - length(p.b(i))$
17     else
18         $Budget \leftarrow Budget - length(p.b(i_t))$
19 STEP 3 ;
20 if Budget has not changed in STEP 2 then
21     Finish
22 else
23     Do STEP 2
```

2.5. Results

Our team name is UnB-SRBR. We obtained at the Validation dataset an average PSNR of 29.349 dB, an MS-SSIM of 0.9599. To compress all the data, we spent 18,665,233 bytes, the decoder size was 70,677,284 bytes and we spent 15,921 seconds to decode this dataset.

References

[1] Johannes Ball, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior, 2018. International Conference on Learning Representations. 2

[2] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2016. Computer Vision and Pattern Recognition. 2