Efficient Learned Flexible-Rate Video Coding: Supplementary Material

A. Codelayer Details

The general strategy is to use a convolutional neural network to transform some collection of features into a quantized codelayer, and then to use entropy coding to losslessly encode those features. Another neural network is then used to decode the codelayer and produce tensors of interest. An entropy coder can efficiently compress a tensor if a well-calibrated probability for each possible value of that tensor is provided. For the I-frame, the input is the image to be compressed. For the P-frame, the input includes other features such as the previous flow and previous reconstruction. The architecture uses side information ($\mathbf{q}_1$) to encode the main codelayer ($\mathbf{q}_0$) more efficiently.

In equations, we have $\mathbf{q}_0 = \mathbf{Q}(\mathbf{E}_0(\text{inputs}))$, $\mathbf{q}_1 = \mathbf{Q}(\mathbf{E}_1(\mathbf{q}_0))$, $\mu_0, \sigma_0 = \mathbf{D}_1(\mathbf{q}_1)$, and outputs $= \mathbf{D}_0(\mathbf{q}_0)$ where $\mathbf{E}_0, \mathbf{E}_1, \mathbf{D}_1, \mathbf{D}_0$ are convolutional neural networks and $\mathbf{Q}(\mathbf{x}) = Q_w \text{round}(\mathbf{x}/(Q_w))$ is a point-wise quantization function with quantization width Q_w. $\mathbf{q}_0$ and $\mathbf{q}_1$ are tensors of the shape $H/s_i, W/s_i, C$ where H, W are the dimensions of the frame, s_i is an integer stride, and C is a number of channels. $\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0$ are tensors of the same shape as $\mathbf{q}_0$. They assign a mean and standard deviation for each element of the codelayer.

In order to use entropy coding to losslessly encode the discrete values, $q \in \{0, \pm Q_w, \pm 2Q_w, \ldots\}$, in the quantized codelayers $\mathbf{q}_0$ and $\mathbf{q}_1$, a probability for each possible discrete value needs to be assigned. A Gaussian with support on the real line can be used to provide probabilities for a discrete distribution by computing the area under the Gaussian within $\pm 1/2$ of the quantization width: $p(q|\mu, \sigma, Q_w) = \int_{q-Q_w/2}^{q+Q_w/2} \mathcal{N}(x|\mu, \sigma)\, dx = \Phi(\frac{q-\mu+Q_w/2}{\sigma}) - \Phi(\frac{q-\mu-Q_w/2}{\sigma})$ where Φ is the CDF of the standard normal distribution.

The codelength can be computed and included as part of the differentiable loss by computing the sum over all elements in the codelayers $\sum_{i,j} -\log_2 p(q_{i,j}|\mu_i, \sigma_i, Q_w)$ where j sums over the height, width, and channel axes of the codelayer tensor. This interpretation of the probabilities for each discrete quantized value results in the same equations as Balle *et al.*, but is simpler than their interpretation which involves convolving an initial density model with a uniform distribution.

Since the quantized tensors are of a shape a H' by W' by C, an estimate of the codelength as a *function of space* can be computed by summing the log probabilities over the channel axis, using nearest neighbor upsampling to align the codelength maps which have different strides.

A Gaussian range encoder (GRE) is used to losslessly encode the quantized codelayer $\mathbf{q}_0$ and subnet codelayer $\mathbf{q}_1$. During encoding: $\mathbf{b}_1 = \text{GRE}(\mathbf{q}_1|\boldsymbol{\mu} = 0, \boldsymbol{\sigma} = \boldsymbol{\sigma}_1)$ and $\mathbf{b}_0 = \text{GRE}(\mathbf{q}_0|\boldsymbol{\mu} = \boldsymbol{\mu}_0, \boldsymbol{\sigma} = \boldsymbol{\sigma}_0)$. During decoding, a Gaussian range decoder (GRD) is used to decode the encoded bits: $\mathbf{q}_1 = \text{GRD}(\mathbf{b}_1|\boldsymbol{\mu} = 0, \boldsymbol{\sigma} = \boldsymbol{\sigma}_1)$ and $\mathbf{q}_0 = \text{GRD}(\mathbf{b}_0|\boldsymbol{\mu} = \boldsymbol{\mu}_0, \boldsymbol{\sigma} = \boldsymbol{\sigma}_0)$. $\boldsymbol{\sigma}_1$ is a parameter of the model and gives a standard deviation for each channel of $\mathbf{q}_1$. The Gaussian range coder uses the generalized entropy coder: ANS (specifically the rANS variant [1]). The input distribution to the ANS entropy coder is the discretized Gaussian distribution with the specified mean and std.

Since the Gaussian probability model has a diagonal covariance matrix, the elements of the codelayer can be encoded in parallel. Autoregressive probability models, which are common in ML-based compression, are not used because they result in prohibitively slow decoding in their current form.

B. Backbone Architecture

Full layer specification of our model along with backbone configurations and DM block parameters can be found in Figure 1.

C. Channel Normalization

In a number of locations of the network, tensors with different average magnitudes are concatenated across the channel axis and fed as input to a subsequent neural network. For instance, the flow is in units of pixels and may be on the order of 10. In contrast, the residual is normalized to roughly be between -1 and 1. In standard neural networks, normalization is handled by batch normalization.

However, one issue with batchnorm is that it effectively injects noise into the network, as the output for a particular example depends on the other randomly sampled elements in the batch. Adding noise during training in this way is not ideal for the problem of compression. Instead, we compute channel-wise moving averages of the means and variances of the input to the layer. Then those features are normalized using the computed means and variances. In order to get around the issue of the mean and variance drifting to infinity, we freeze the moving averages after 2000 training steps. In practice, failing to normalize the inputs in one way or another causes the network to either blow up or to under-

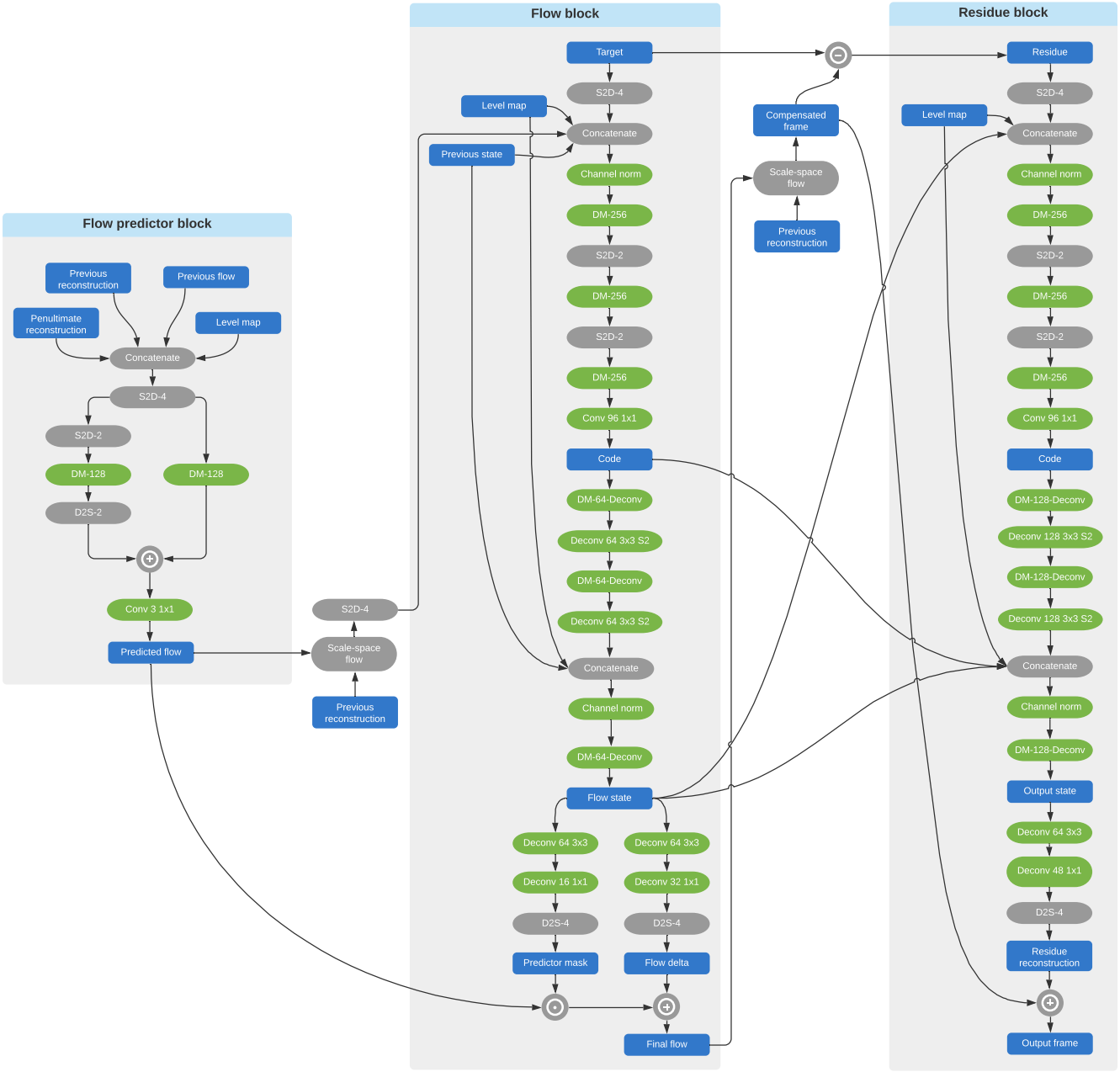

Figure 1. Full layer specification of our model along with backbone configurations and DM block parameters.

perform significantly.

D. Loss Modulator

The loss modulator multiplies the reconstruction loss for a given P-frame and level by a factor μ in order to give extra weight to modes that are under-performing during training.

Since the training uses the MSE, the difference in PSNR criterion is converted into a multiplicative factor based on the MSE using the following relationship: $\text{PSNR}_1 > \text{PSNR}_2 - \delta \iff \text{MSE}_1 < \text{MSE}_2 \cdot f(\delta)$ where $f(\delta) = 10^{\delta/10}$. Empirically, we used $f(\delta) = 1.5 \implies \delta = 1.76$. The threshold for the loss modulator was chosen by training the model for 30 epochs for a few different values of the δ between the PSNR of the I- and P-frames. μ is initialized to 1 for all frames and levels, but isn't particularly sensitive to the initialization. μ is clipped to be in the range $[1.0, 10.0]$ to improve training stability.

This method can also be seen as annealing the regularization weight to allow for larger bitrates for under-performing frames and as the performance improves, the bitrate is more aggressively regularized. We hypothesize that this method is useful because if the regularization is too strong, the

model can struggle to train. In equations, the loss can be written as:

$$\sum_{l,t} \mu_t^{(l)} \cdot \left[L_{\text{rec},t}^{(l)} + \frac{\lambda_{\text{reg}}^{(l)}}{\mu_t^{(l)}} R_t^{(l)} \right] \tag{1}$$

Thus the effective regularization weight $\frac{\lambda_{\text{reg}}^{(l)}}{\mu_t^{(l)}}$ is smaller when μ is increased.

In practice, the $\mu_t^{(l)}$ as a function of training iteration starts at 10, stays there, and then more or less smoothly transitions to 1. The lower levels (small l) and earlier frames (small t) train more efficiently so the crossover time is earlier for those frames and levels. This procedure may also mitigate error accumulation over time when unrolling the model by placing more emphasis on the later frames.

E. Level interpolation

Suppose that one wants to embed L levels in a L_e-dimensional space in a way that smoothly interpolates between levels and reduces to a one-hot representation when $L_e = L$. The embedding of a level $l \in \{0, 1, \dots L-1\}$ is computed as follows:

$$s_l = \frac{l(L_e - 1)}{L - 1} \qquad u_l = \lfloor s_l \rfloor, \qquad v_l = u_l + 1 \tag{2}$$

$$l_e = (1 - \{s_l\}) \cdot \text{onehot}(u_l | L_e) + \{s_l\} \cdot \text{onehot}(v_l | L_e) \tag{3}$$

where $\{s\}$ denotes the fractional part of s and $\text{onehot}(a|b)$ gives the b-dimensional one-hot representation of the integer a. If $a < 0$ or $a \geq b$, it returns the zero vector. It can be verified that if $L_e = L$, this reduces back to the original one-hot representation. When $L > L_e$, this method smoothly interpolates between one-hot vectors. This embedding allows us to arbitrarily increase the number of levels without additional training (Figure 2). The level-interpolated model is used in the rate control figures in the main text. As an example, if 5 levels are embedded in a 3 dimensional space, the resulting vectors are $(1,0,0), (0.5,0.5,0), (0,1,0), (0,0.5,0.5), (0,0,1)$.

In order to derive these equations, consider a number line marked at $0, 1, \dots L_e - 1$. Each of these marks corresponds to a one-hot vector in the L_e-dimensional space. We want to evenly space L values on this number line where the minimum value is zero and maximum value is $L_e - 1$. It can be seen that s_l produces evenly spaced points on this line. The nearest one-hot vectors corresponding to s_l are marked at u_l and v_l. The distance from s_l to u is $\{s\}$ and the distance from s_l to v is $1 - \{s\}$. The final embedding vector is computed by weighting the nearby L_e-dimensional one-hot vectors by the distance of s_l to u_l and v_l.

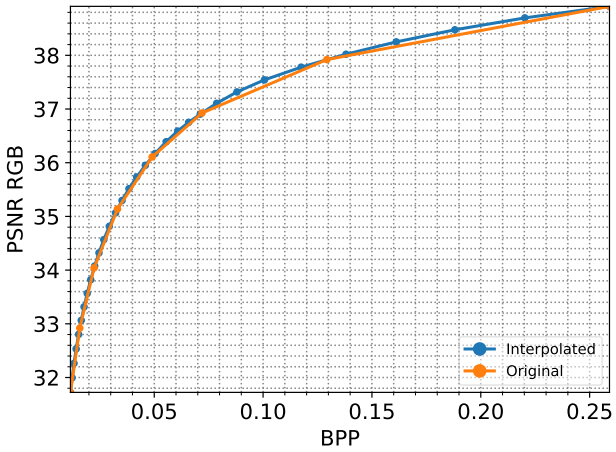

Figure 2. R-D curves of the baseline and level-interpolated model match on the UVG dataset. The baseline model has 8 levels and the interpolated model supports 32 levels.

F. Dataset

The Vimeo90k dataset [3] consists of 91,701 7-frame sequences with fixed resolution 448×256. For finetuning with a larger crop size, we generate a Vimeo90k-like dataset, which consists of 100k 32-frame clips of resolution 352×352. The clips were generated from 2363 full length original videos which were used to generate the Vimeo90k dataset (http://data.csail.mit.edu/tofu/dataset/original_video_list.txt). The original videos were pre-processed into distinct segments with a basic threshold-based scene-cut detector. The 32-frame clips were then extracted from the segments at a random downscale factors to ensure a wide range of motion in the dataset.

G. Results by Video Type

As mentioned in the main paper, we found that our codec suffers on non-photorealistic videos. In Figure 4 we plot the performance of ELF on the four non-photorealistic videos in the MCL-JCV dataset (video IDs 18, 20, 24, 25 within the set), and in Figure 3 plot on all other videos.

H. Results by Loss Type

Figures 5, 6, and 7 compare and contrast the reconstructions generated by the models optimized for different distortion losses.

I. Commands Used for Standards-based Video Compression

H.264 We use the following command to encode all H.264 videos in the paper:

```
ffmpeg -i [SRC] \
    -preset medium \
    -codec:v libx264 \
    -crf [RATE] \
    -x264-params bframes=0 [DST]
```

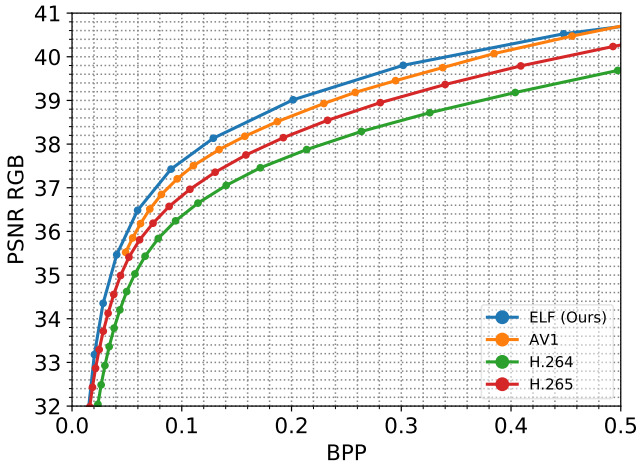

Figure 3. ELF R-D curves on all natural videos in the MCL-JCV dataset.

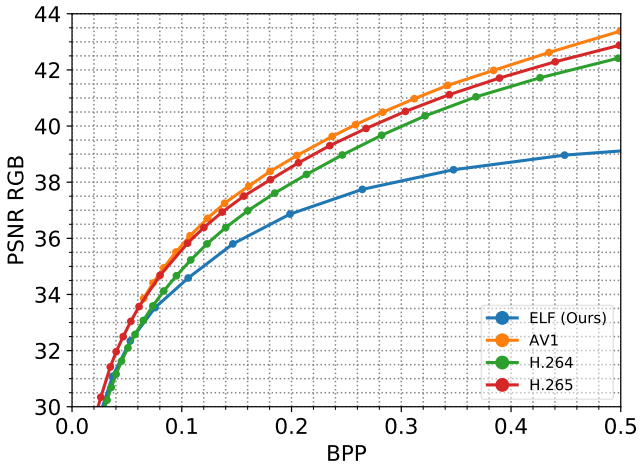

Figure 4. ELF R-D curves on the four non-photorealistic videos in the MCL-JCV dataset.

H.265 We use the following command to encode all H.265 videos in the paper:

```
ffmpeg -i [SRC] \
    -preset medium \
    -codec:v libx265 \
    -crf [RATE] \
    -x265-params bframes=0 [DST]
```

AV1 We use the SVT-AV1 Encoder [2] (version v0.8.5-72-gd210088) for comparison. Encoding using only I– and P-frame types is disabled by default in the SVT-AV1 encoder, but can be enabled as mentioned in the github issue https://github.com/AOMediaCodec/SVT-AV1/issues/973. The encoding command used is:

```
SvtAv1EncApp -i [SRC] \
    -b [DST] \
    --rc 0 -q [RATE] \
    --hierarchical-levels 0 \
    --lookahead 0
```

Figure 5. Qualitative Differences Between PSNR and MS-SSIM-optimized models at Low Bitrates. Reconstructions and reconstruction error of the first frame of ntia spectrum1 from the CDVL SD dataset. The first column shows a PSNR model encoded using level $= 1$. The second and third columns show the MS-SSIM model with level $= 1, 2$. The PSNR model has less detail in the textures of the face, but is more accurate in reconstructing the edges of the text in the top left of the frame. The residual is boosted by a factor of 3 for visualization purposes.

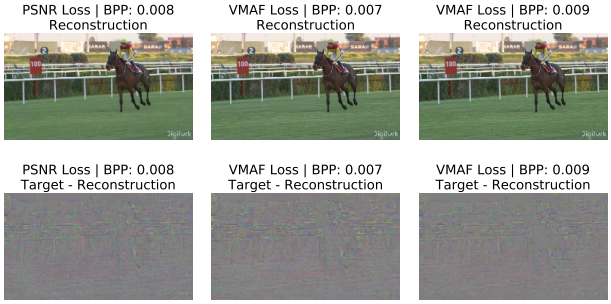

Figure 6. Visual Comparison between PSNR and VMAF-optimized models at Low Bitrates. Reconstructions and reconstruction error of the fourth frame of Jockey from the UVG dataset. The first column shows the YCbCr PSNR model encoded using level $= 1$. The second and third columns show the VMAF-finetuned model encoded using level $= 1, 2$. The reconstructions look roughly the same despite the fact that the VMAF-finetuned model performs significantly better on the VMAF metric.

J. Model Consolidation

We start with two separate R-D curves, one for each model. Given the selection of regularization weights, the two curves cover different BPP ranges, but still overlap.

These two R-D curves are consolidated together into a single one. This is simply done by computing the upper convex hull of all R-D points, and keeping the points which are used to construct this hull.

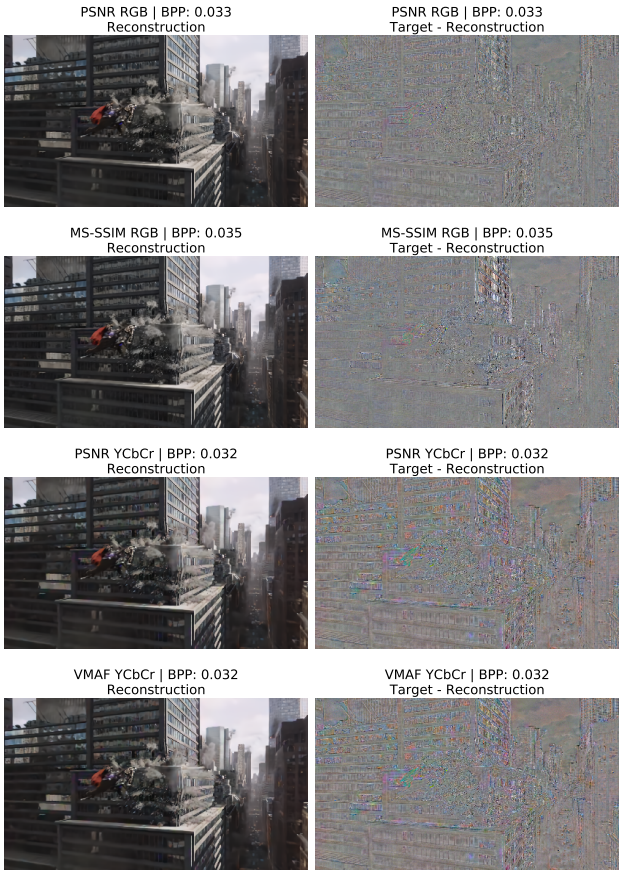

Figure 7. Side-by-Side Comparisons of All Models at Low Bitrates. Reconstructions and reconstruction error of the 11th frame of video 22 from the MCL JCV dataset (encoded using level $= 1$). Note that MS-SSIM model has larger error in the windows in the middle of the frame and the YCbCr models have a larger amount of color distortion.

References

[1] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013. 1

[2] Github. Scalable video technology for av1 (svt-av1 encoder and decoder). 4

[3] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision (IJCV)*, 127(8):1106–1125, 2019. 3