Compression Without Learned Encoders: Adaptive Probability Modeling on Latent Manifolds

Pablo Strasser

Wednesday 17th June, 2026

Haute école de gestion

dmml group

Introduction

Notation:

- X: data
- C: compressed representation of the data

What we need for compression:

1. A Compressor:
$$\mathbb{P}(C \mid X)$$
2. A Decompressor:
$$D(X \mid C)$$
3. A prior distribution over compressed representations:
$$\mathbb{P}(C)$$

Standard Neural Compressor

Pro and Cons of Standard Neural Compressor

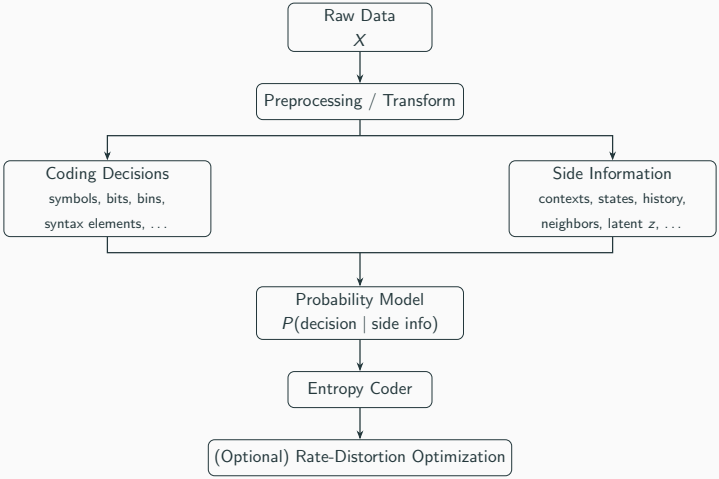
Cons:

- Training the autoencoder is long.
- The amount of input data is big.
- The model is very bad except at the end of training.
- There is no robustess on data outside the training distribution.

Pros:

- Better compression ratio than classical methods.
- Adapt to the data distribution.

Classical Method



Classical Codec Examples

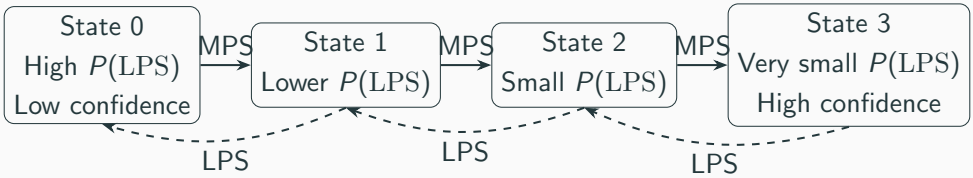
Codec	Transform / Representation	Coding decisions	Probability model	Rate optimization
JPEG	DCT (Discrete Cosine Transform) + quantization	Quantized coefficients	Huffman	Quantization controls rate
JPEG2000	Wavelet transform + bitplanes	Significance/refinement decisions	MQ arithmetic coder	Progressive truncation (Tier-2)
H.264 / AVC	Prediction + transform + quantization	Syntax elements (prediction modes, motion vectors, coefficients)	CABAC/CAVLC	Encoder RDO
HEVC / H.265	Improved prediction + transform + quantization	Syntax elements	CABAC	Encoder RDO
VVC / H.266	Advanced prediction + transform + quantization	Syntax elements	CABAC	Encoder RDO

Probability Models in Vision Codecs

- Vision codecs encode many small binary decisions: coefficient significance, signs, refinement bits, prediction modes, and syntax flags.
- The probability model estimates $P(\text{next bit} \mid \text{context})$ and gives this probability to an arithmetic entropy coder.
- A **context** is a discrete summary of local side information: neighboring coefficients, previous decisions, block type, position, or coding mode.
- Instead of one global probability, each context has its own adaptive probability estimate.
- **CABAC** (H.264/HEVC/VVC) and **MQ** (JPEG2000) use small finite-state machines per context.
- Each state stores two things: the probability level of the less probable symbol and which symbol is currently more probable.
- After each encoded bit, the context moves to a new state: confidence increases after expected bits and decreases, or flips symbol, after surprising bits.

MQ State Evolution

Repeated MPS observations
make the model more confident



LPS observations move back toward uncertainty;
near low-confidence states, the MPS may switch

- MQ uses a small discrete table of states, not a continuous learned probability.
- Each context stores its current state and its current MPS value.

Compression Size and Likelihood

Bit cost
For an event with probability p, the ideal code length is:

$$-\log_2(p) \quad \text{bits}$$

(With an optimal entropy coder, e.g. arithmetic coding or range coding.)

- High probability event: small bit cost.
- Low probability event: large bit cost.
- For a sequence of coding decisions $d_1, \ldots, d_n$:

$$\begin{aligned} \text{size} &= -\sum_i \log_2 P(d_i \mid \text{context}_i) \\ &= -\log_2 P(d_1, \ldots, d_n) \end{aligned}$$

- Compression is therefore likelihood maximization: better probability models assign higher likelihood and produce fewer bits.

What Information Is Free?

Key idea: side information, contexts, and probability states do not need to be transmitted if the decoder can reproduce them exactly.

- The bitstream only pays for the actual coding decisions: symbols, bins, syntax elements, coefficients, etc.
- The **context** is computed from information already available to both sides: position, block type, neighboring decoded values, previous decisions, and syntax history.
- The **state** of CABAC/MQ is updated deterministically after every decoded bit.
- Therefore, after decoding the same past bits, the decoder has the same context and the same probability state as the encoder.
- This is why the model can use rich side information without sending it explicitly.

Not free: anything the decoder cannot derive must be encoded in the bitstream, or specified as fixed shared rules in the codec standard. For example, a learned latent representation is not free unless it can be reconstructed from already decoded data.

What About Reconstruction?

- Reconstruction accuracy decides **what information is worth keeping**.
- This enters through the transform, quantization, and coding decisions: which coefficients, bits, modes, or latents are kept or dropped.
- It does **not** directly enter the probabilistic model.
- The probability model only answers: how many bits does it cost to send this decision?
- The rate optimizer chooses how much information to keep by trading distortion against rate, e.g. distortion $+ \lambda$ rate.
- A better probability model makes decisions cheaper to encode.
- For a fixed bitrate, cheaper decisions mean more useful decisions can be kept, improving reconstruction.

Our Latent probability model

We learn a model that predicts each discrete coding decision from discrete side information and a discrete latent variable z:

$$P_\theta(d_t \mid s_t, z)$$

- d_t: discrete decision to encode.
- s_t: discrete side information / context available to encoder and decoder.
- z: discrete latent variable.
- The model gives the probability used to entropy-code d_t.
- There is no explicit position variable: the same model is reused everywhere, based only on (s_t, z).
- This makes the model behave like a learned probability table conditioned by side information and the latent variable.

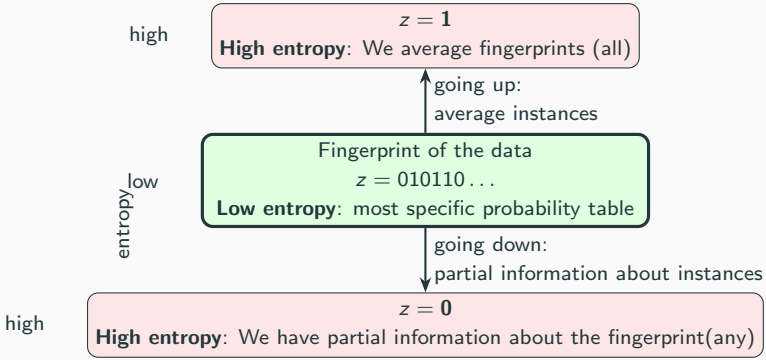
How to Get a Discrete Latent Variable?

- We could use an autoencoder + quantization to get a discrete latent $z \rightarrow$ but this is slow to train.
- What we want is something similar to hierarchical clustering:
  - With different levels of granularity from low-entropy(fine) to high-entropy(coarse).
  - The lowest level of entropy is a lookup table with optimal probability. Highest level of entropy is a single global average probability table.
  - Moving from low to high entropy is done by merging probability (averaging the probabilities of latent below).

How to Get a Discrete Latent Variable?

- Inspiration: Bloom Filters:
  - We define a fingerprint by hashing the the data X k times and create a binary vector of size m where we set to 1 the positions given by the hash functions.
  - We define a regularizer which enforce that latent z with more bits set to 1 are given by the average of the probabilities of the latent with fewer bits set to 1.
  - To be more precise, The vector at sparsity $k' < k$ is given by the average of all the vectors at sparsity $k' - 1$ that are subsets of it.
  - For the vector of sparsity $k > k'$, we impose the reverse $(k' + 1)$.
- Going up from the fingerprint we average the probability by loosing information. The extrem is $z = \mathbf{1}$ which is the full average of all the probability tables.
- Going down from the fingerprint we loose information by having only partial fingerprint. The extrem is $z = \mathbf{0}$ which is also the full average of all the probability tables.

z and entropy



Training

- Because model doesn't have the position as input, we only need summary statistics (histograms of the tuple (d_t, s_t)) to train the model.
- We train on a random subset power of two of a block.
- Very easy to train and very fast.
- Very easy to obtain results slightly better than MQ.
- Its fast because we have no encoder to learn.

What do we do with the model?

- At inference, given a subset of a block, we want to compute z so that:
$$size = -\sum_t \log_2(P_\theta(d_t \mid s_t, z)) + C(z)$$
where $C(z)$ is the cost of encoding z in bit.
- We decompose the cost of encoding z as sending the sparsity k and all possible vector of sparsity k ($C(k) + \log_2 \binom{m}{k}$).
- For a given z, the model can be seen as a vector of logits. The size is then a linear function from this vector of logits:

$$\ell_z = \left(-\log_2 P_\theta(d \mid s, z)\right)_{(s,d)}, \qquad \mathbf{h} = \left(\#\{t : (s_t, d_t) = (s, d)\}\right)_{(s,d)},$$
$$\text{size}(z) = \mathbf{h}^\top \ell_z + C(k) + \log_2 \binom{m}{k}, \qquad k = \|z\|_0.$$

$$z^* = \underset{z \in \{0,1\}^m}{\arg\min} \left(\mathbf{h}^\top \ell_z + C(\|z\|_0) + \log_2 \binom{m}{\|z\|_0}\right)$$

Solving for z

- Solving for every block is too expensive.
- We learn an encoder that predict good z from the histogram $\mathbf{h}$ for a test dataset.

High-level view: each candidate z defines one linear cost in the histogram, and the optimal cost is the lower envelope of these lines:

$$L_z(\mathbf{h}) = \mathbf{h}^\top \ell_z + C(z), \qquad L^*(\mathbf{h}) = \min_z L_z(\mathbf{h}).$$

- Learning the encoder means learning which part of this min-envelope is active for a given $\mathbf{h}$.
- At test time, the encoder proposes a good z (or a small candidate set), then we evaluate the exact linear size.

Using the encoder

- A whole block does not need to use one single z.
- We test candidate segments whose lengths are powers of two: $1, 2, 4, 8, \ldots$.
- For each segment $[i, j)$, build $\mathbf{h}_{i:j}$, ask the encoder for $z_{i:j}$, and compute:

$$\mathrm{cost}(i, j) = \mathbf{h}_{i:j}^{\top} \boldsymbol{\ell}_{z_{i:j}} + C(z_{i:j}) \qquad D[j] = \min_{q:\, 2^q \le j} \left(D[j - 2^q] + \mathrm{cost}(j - 2^q, j) \right)$$

- $D[j]$ is the best cost for the prefix ending at position j.
- This tests many segment sizes while keeping the search cheap.
- The decoder can reproduce the same segmentation and z choices from the transmitted decisions.

Final Accounting

For each selected segment, the bitstream must contain:

- the segment length;
- the sparsity $k = \|z\|_0$;
- the chosen fingerprint mask z among masks of sparsity k;
- the arithmetic-coded decisions using $P_\theta(d_t \mid s_t, z)$.

$$\text{bits} \approx C(\text{length}, k) + \log_2 \binom{m}{k} - \sum_t \log_2 P_\theta(d_t \mid s_t, z)$$

- We compute a histogram of (length, k) pairs on the dataset.
- This gives a cheaper entropy code for sending segment length and sparsity.

Results:

$bpp = 0.074973$ $psnr = 37.5122$ $msssim = 0.979887$ $bytes = 28924$

Remarks:

- Preliminary results are good even if final results is slightly worse than JPEG2000.
- Doing inference is a bit tricky because the model need to be perfectly deterministic. We use quantization and fix batch size to be identical between encoding and decoding.
- Size budget has a direct influence on quality. The rate optimizer will use the budget for more quality.
- Presented here for JPEG2000, but the method work on other codecs.
- Removing position information make it more tabular, but it make the model maybe weaker.
- No encoder during training make the training faster. The encoder only computed after training.