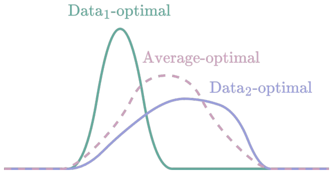
Learned Compression for Images and Point Clouds

Mateen Ulhaq

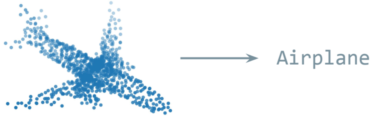
SFU
SIMON FRASER
UNIVERSITY

MASc (current)
BASc Eng. Phys. Hon. and Math minor

Topics



1. Learned compression of “encoding distributions” for compression.

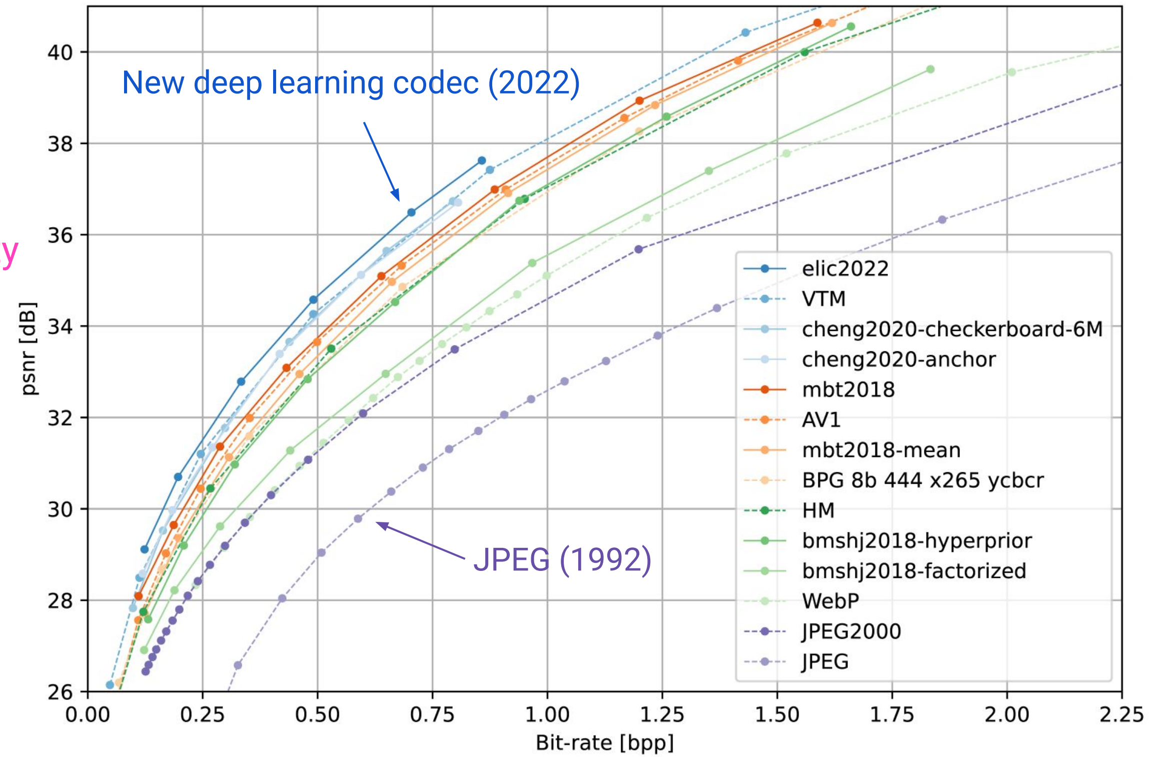


2. Learned point cloud compression for classification.

3. Analyze motion in the latent space (for p-frames).

Comparison of image compression codecs

Goal: max quality



New deep learning codec (2022)

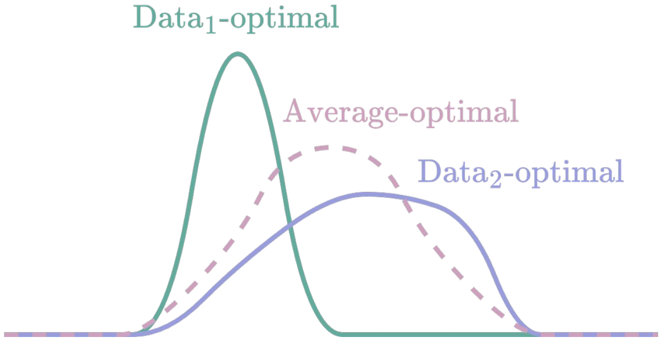
JPEG (1992)

Goal: minimum compression time

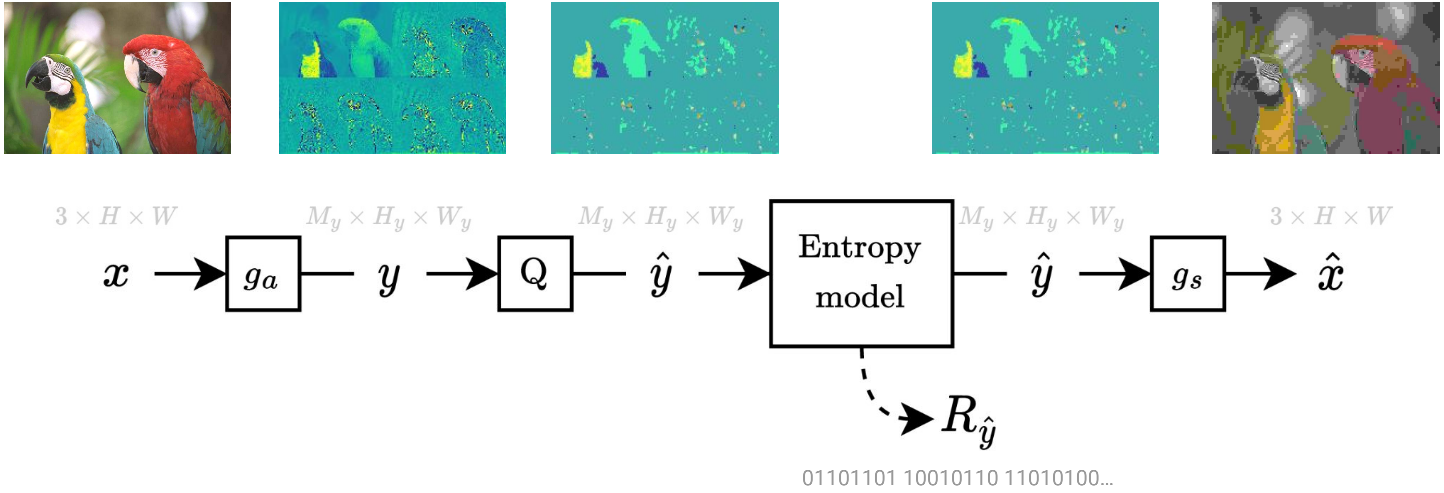
(not visualized)

Goal: minimum file size

Learned compression of “encoding distributions” for compression



Architecture (standard)



$3 \times H \times W$ $\quad$ $M_y \times H_y \times W_y$ $\quad$ $M_y \times H_y \times W_y$ $\quad$ $M_y \times H_y \times W_y$ $\quad$ $3 \times H \times W$

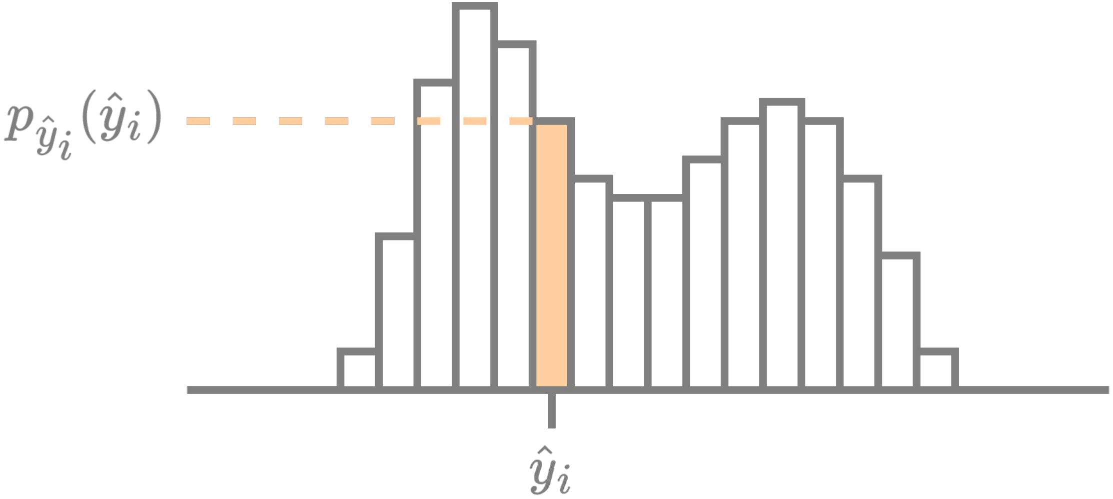
$x \rightarrow \boxed{g_a} — y \rightarrow \boxed{Q} — \hat{y} \rightarrow$ Entropy model $— \hat{y} \rightarrow \boxed{g_s} \rightarrow \hat{x}$

$R_{\hat{y}}$

01101101 10010110 11010100...

$$\mathcal{L} = R + \lambda \cdot D(\boldsymbol{x}, \hat{\boldsymbol{x}})$$

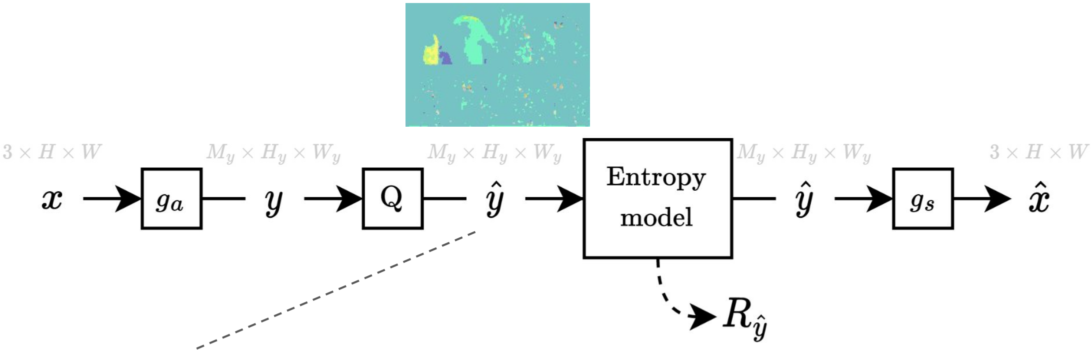
Encoding distribution

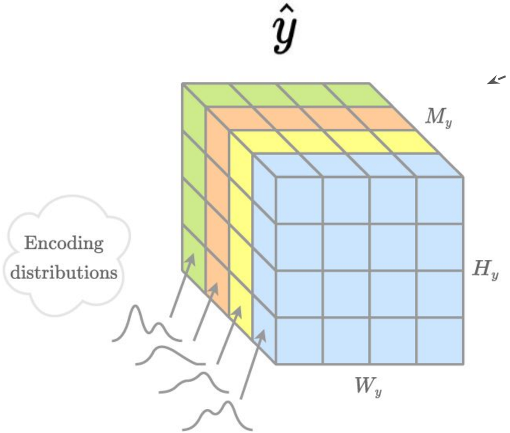


p(ŷᵢ)	Rate cost
1	0 bits
1/2	1 bit
1/4	2 bits
1/8	3 bits
1/16	4 bits
…	
0	∞ bits

Rate cost of encoding a single element: $R_{\hat{y}_i} = -\log_2 p_{\hat{y}_i}(\hat{y}_i)$

Factorized model





(Each color represents the use of a different encoding distribution.)

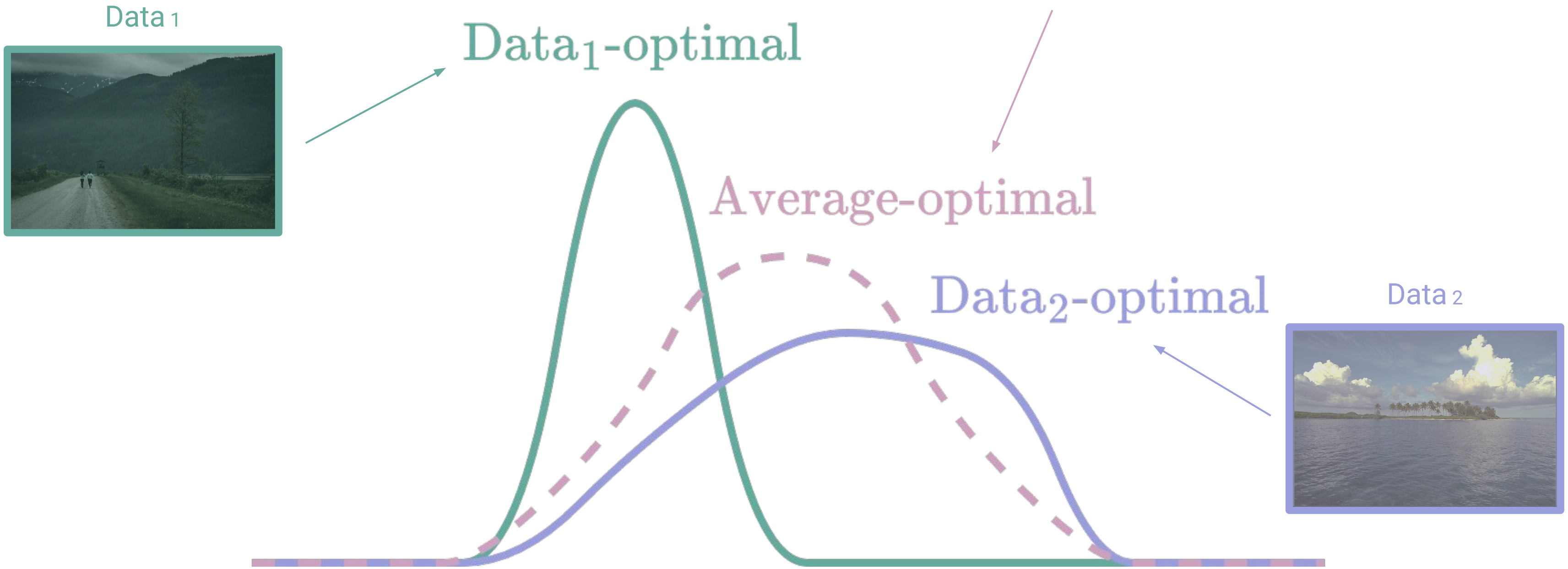
Each channel is encoded using a unique encoding distribution.

Component in SOTA models.

Static: uses same set of encoding distributions for all inputs.

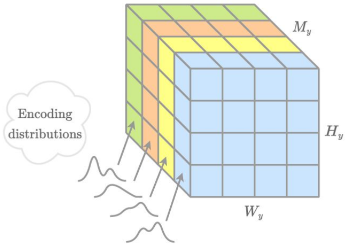
Suboptimality of static encoding distributions

The static "average-optimal" distribution tries to account for all possible data distributions.



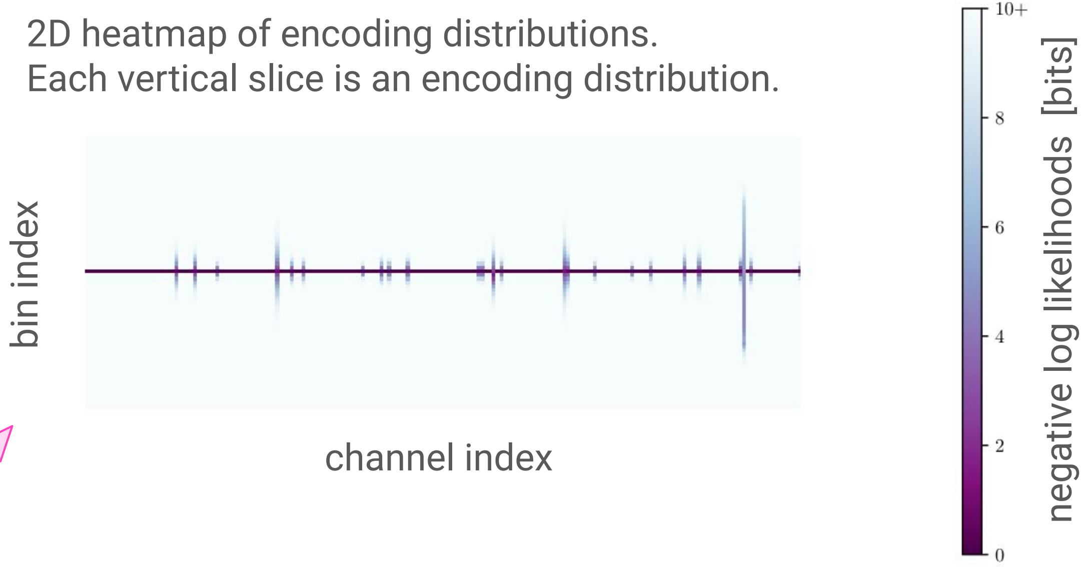
Distribution heatmap

Recall: each channel uses a unique encoding distribution.



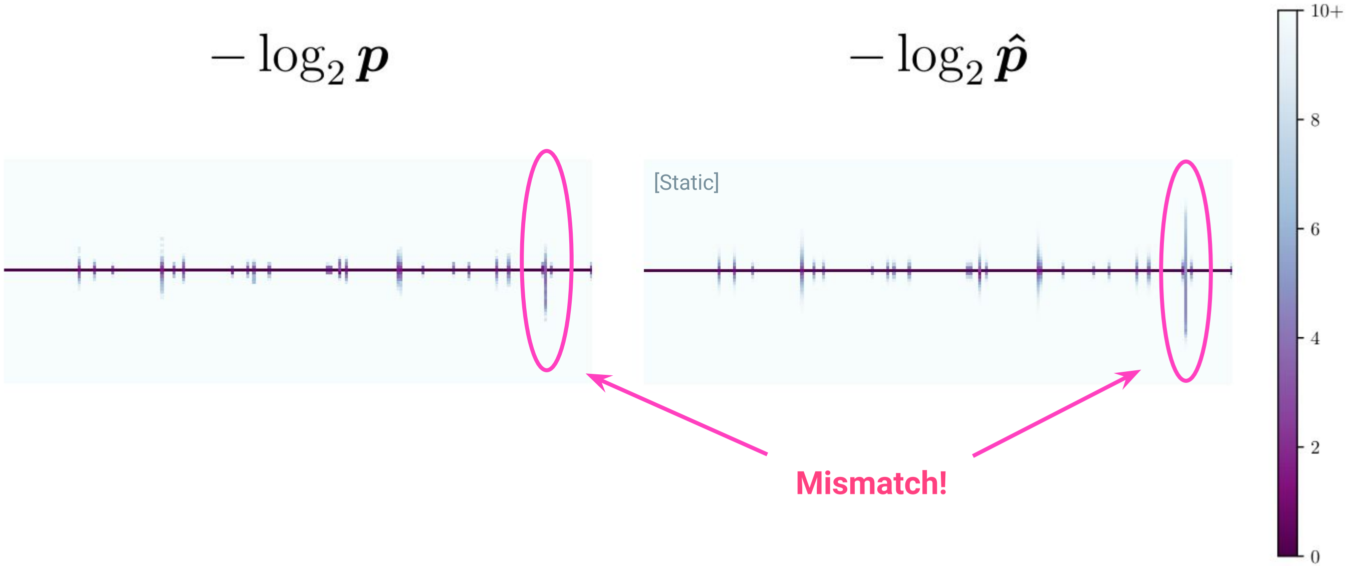
Let’s plot them as a 2D heatmap.

2D heatmap of encoding distributions.
Each vertical slice is an encoding distribution.



Encoding distributions (non-adaptive, static)

Input image



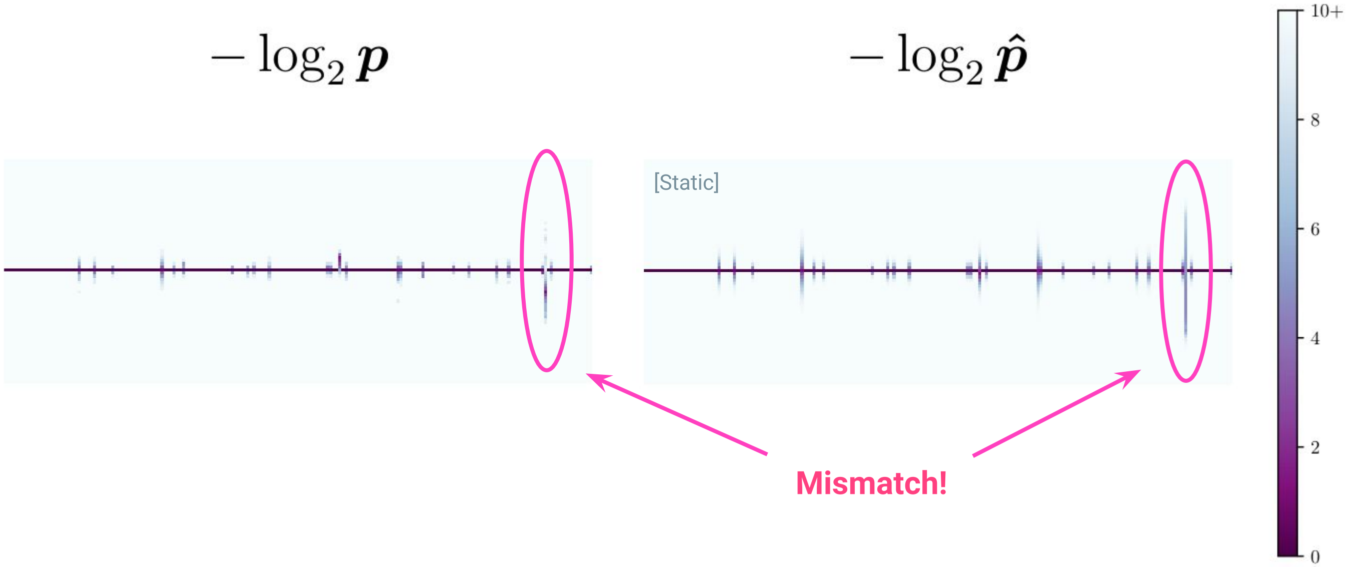
Left shows ideal.
Right is actual (static).
Mismatch leads to suboptimal rate.

(Ideal)

(Actually used for encoding)

Encoding distributions (non-adaptive, static)

Input image



Left shows ideal.
Right is actual (static).
Mismatch leads to suboptimal rate.

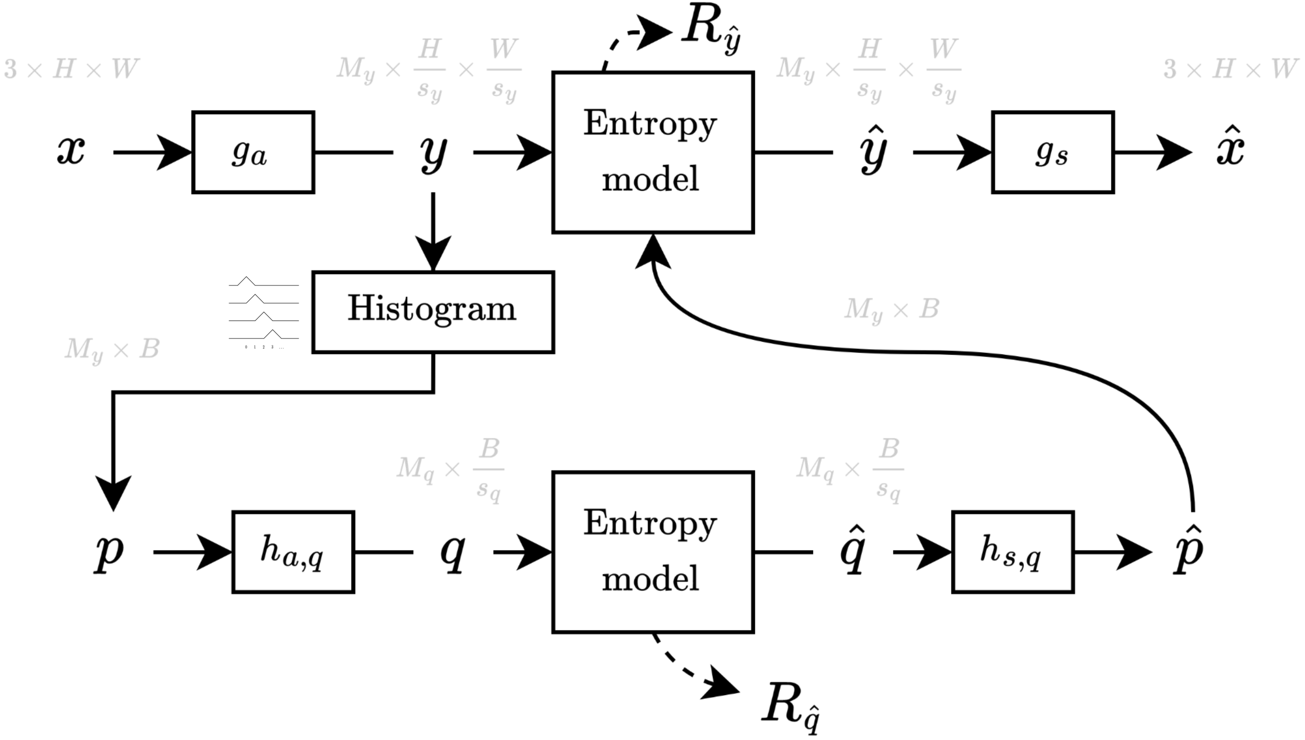
(Ideal)

(Actually used for encoding)

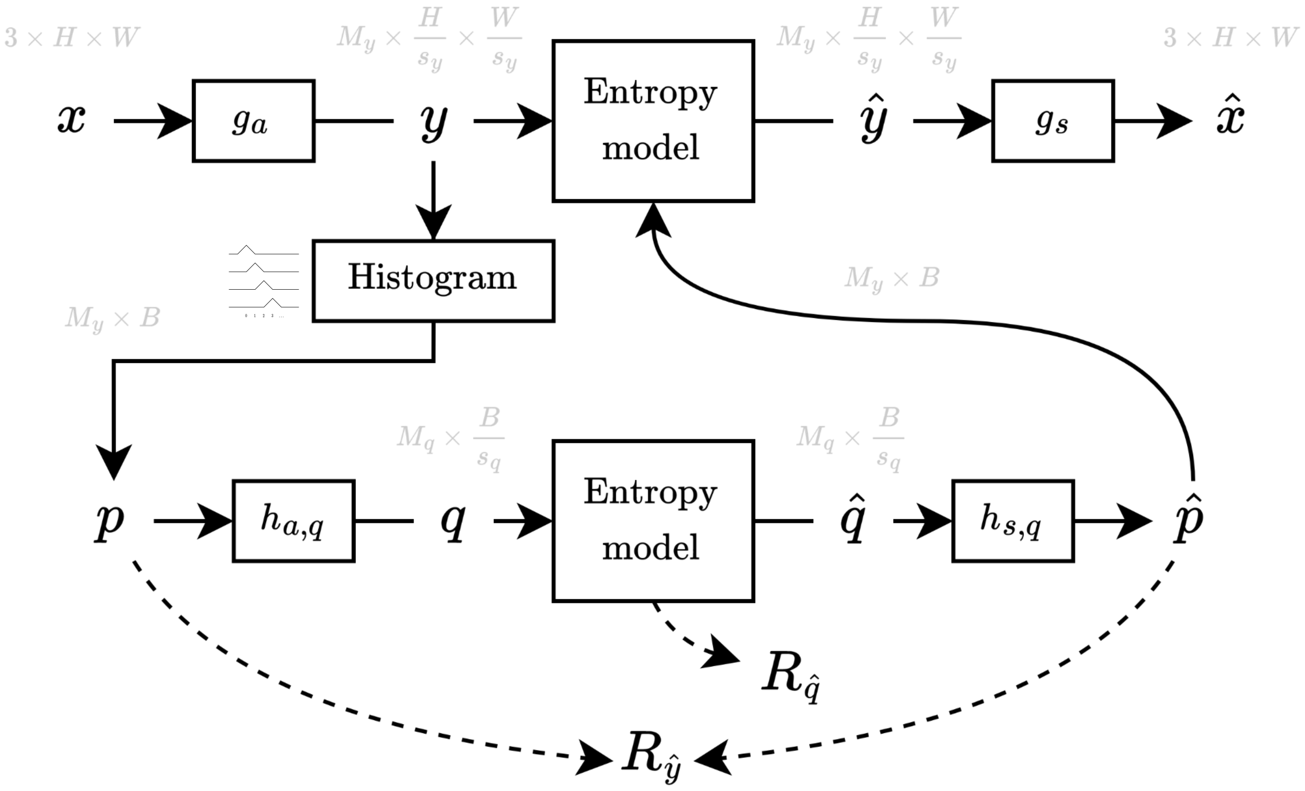
How do we address this mismatch?

Solution: transmit per-image adapted encoding distributions.

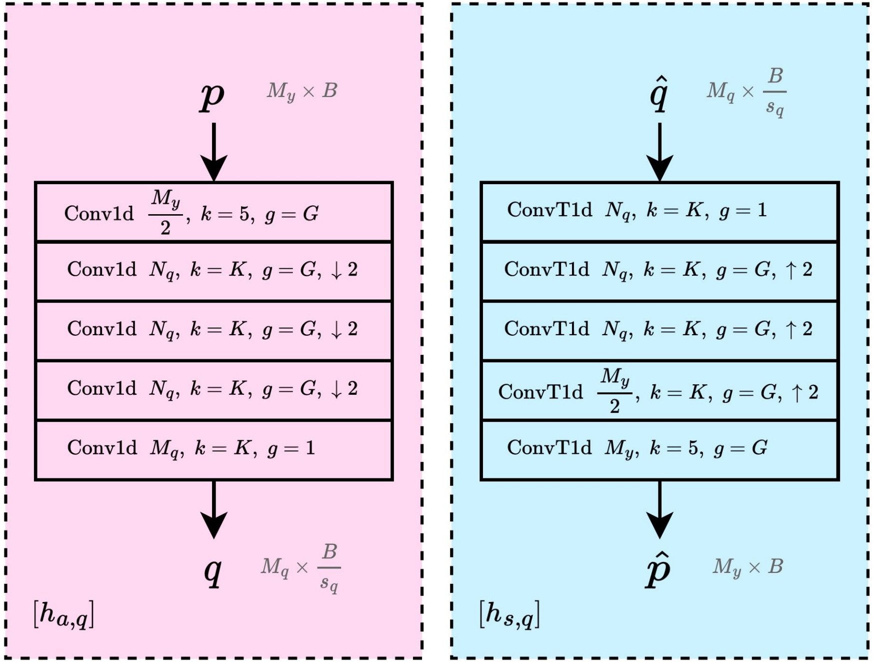
Architecture (with proposed distribution compression model)



Architecture (with proposed distribution compression model)



Architecture details (proposed transforms)



$\boldsymbol{p}$ $M_y \times B$

Conv1d $\frac{M_y}{2}$, $k = 5$, $g = G$

Conv1d N_q, $k = K$, $g = G$, $\downarrow 2$

Conv1d N_q, $k = K$, $g = G$, $\downarrow 2$

Conv1d N_q, $k = K$, $g = G$, $\downarrow 2$

Conv1d M_q, $k = K$, $g = 1$

$\boldsymbol{q}$ $M_q \times \frac{B}{s_q}$

$[h_{a,q}]$

$\hat{\boldsymbol{q}}$ $M_q \times \frac{B}{s_q}$

ConvT1d N_q, $k = K$, $g = 1$

ConvT1d N_q, $k = K$, $g = G$, $\uparrow 2$

ConvT1d N_q, $k = K$, $g = G$, $\uparrow 2$

ConvT1d $\frac{M_y}{2}$, $k = K$, $g = G$, $\uparrow 2$

ConvT1d M_y, $k = 5$, $g = G$

$\hat{\boldsymbol{p}}$ $M_y \times B$

$[h_{s,q}]$

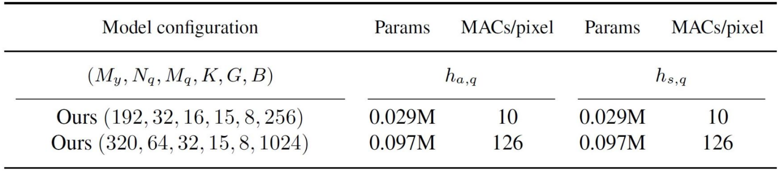
Figure 2.4: Architecture layer diagram for $h_{a,q}$ and $h_{s,q}$ transforms. k denotes kernel size, g denotes number of channel groups, and $\downarrow, \uparrow$ denote stride.

“ShuffleNet” CNN with 3 downsample strides, grouped conv, and 16—64 channels.

Increase in parameters:

3.00M → 3.06M (low rate)
7.03M → 7.22M (high rate)

Table 2.3: Trainable parameter counts and number of multiply-accumulate operations (MACs) per pixel.



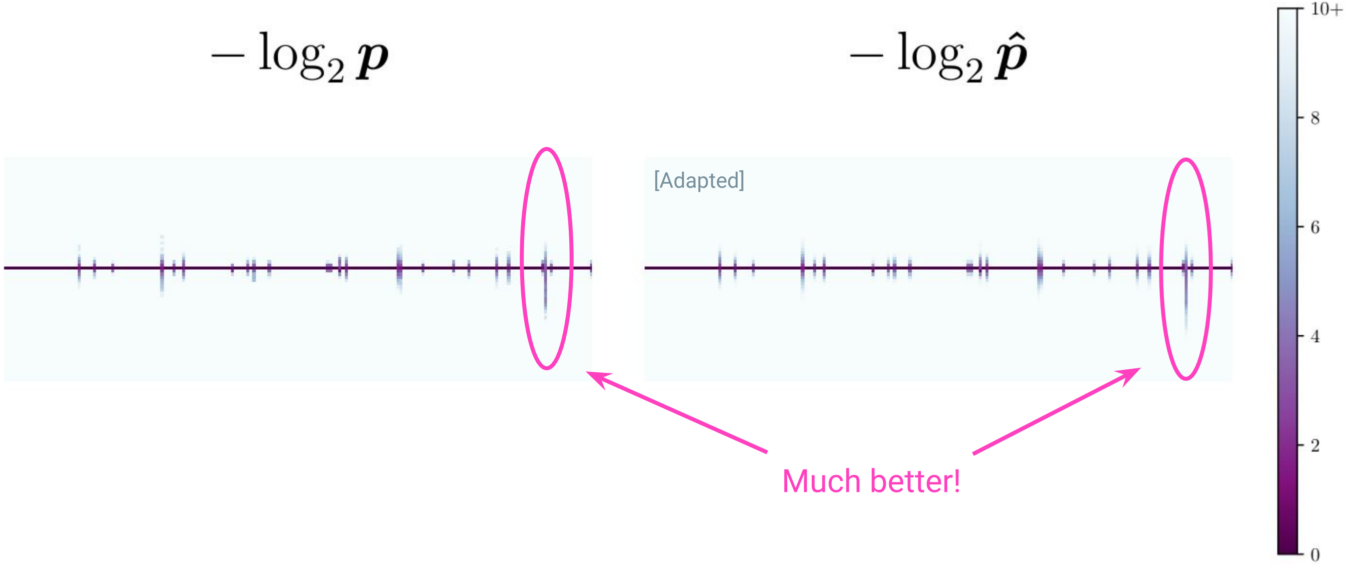
Model configuration	Params	MACs/pixel	Params	MACs/pixel
(M_y, N_q, M_q, K, G, B)	$h_{a,q}$		$h_{s,q}$	
Ours $(192, 32, 16, 15, 8, 256)$	0.029M	10	0.029M	10
Ours $(320, 64, 32, 15, 8, 1024)$	0.097M	126	0.097M	126

Encoding distributions (adaptive, dynamic)

Input image

$-\log_2 \boldsymbol{p}$

$-\log_2 \hat{\boldsymbol{p}}$



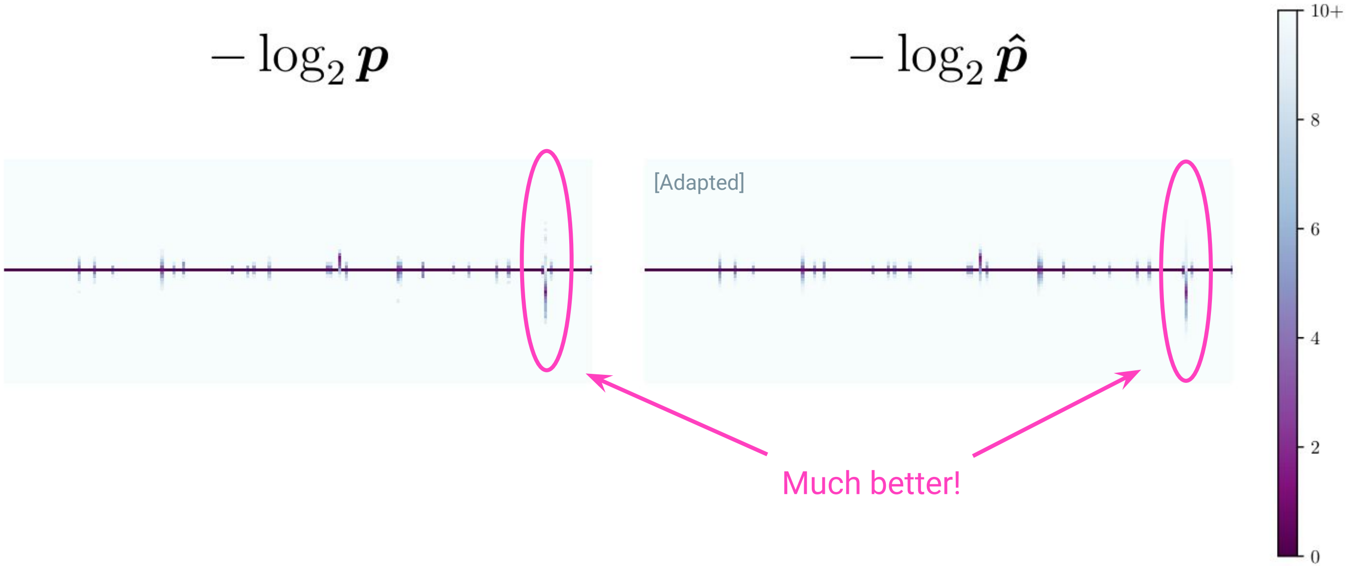
Much better!

(Ideal)

(Actually used for encoding)

Encoding distributions (adaptive, dynamic)

Input image



$-\log_2 \boldsymbol{p}$

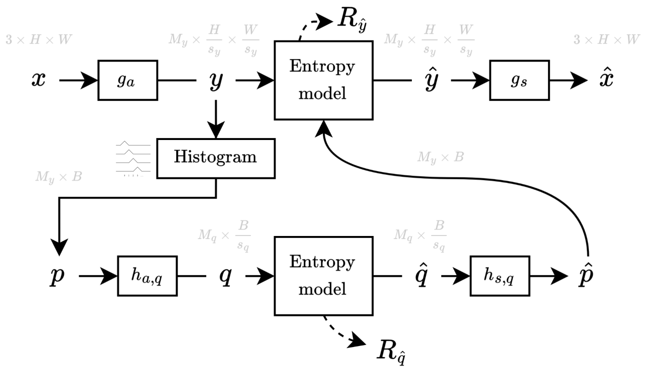
$-\log_2 \hat{\boldsymbol{p}}$

Much better!

(Ideal)

(Actually used for encoding)

Loss function



$$
\begin{aligned}
\mathcal{L} &= \mathbb{E}_{x \sim p_x(x)} \mathbb{E}_{\tilde{y}, \tilde{q} \sim q_\phi(\tilde{y}, \tilde{q} | x)} \left[-\log p_{\tilde{y}|\tilde{q}}(\tilde{y} \mid \tilde{q}) - \lambda_q \log p_{\tilde{q}}(\tilde{q}) + \lambda_x D(x, \tilde{x}) \right] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x})} \left[R_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}) + \underline{\lambda_q} R_{\tilde{\boldsymbol{q}}}(\boldsymbol{x}) + \lambda_x D(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \right]
\end{aligned}
$$

For a target that is 6x larger than the image patch we trained on, we can afford 6x more rate for the q bitstream, since the same encoding distribution is reused 6x more in the larger image.

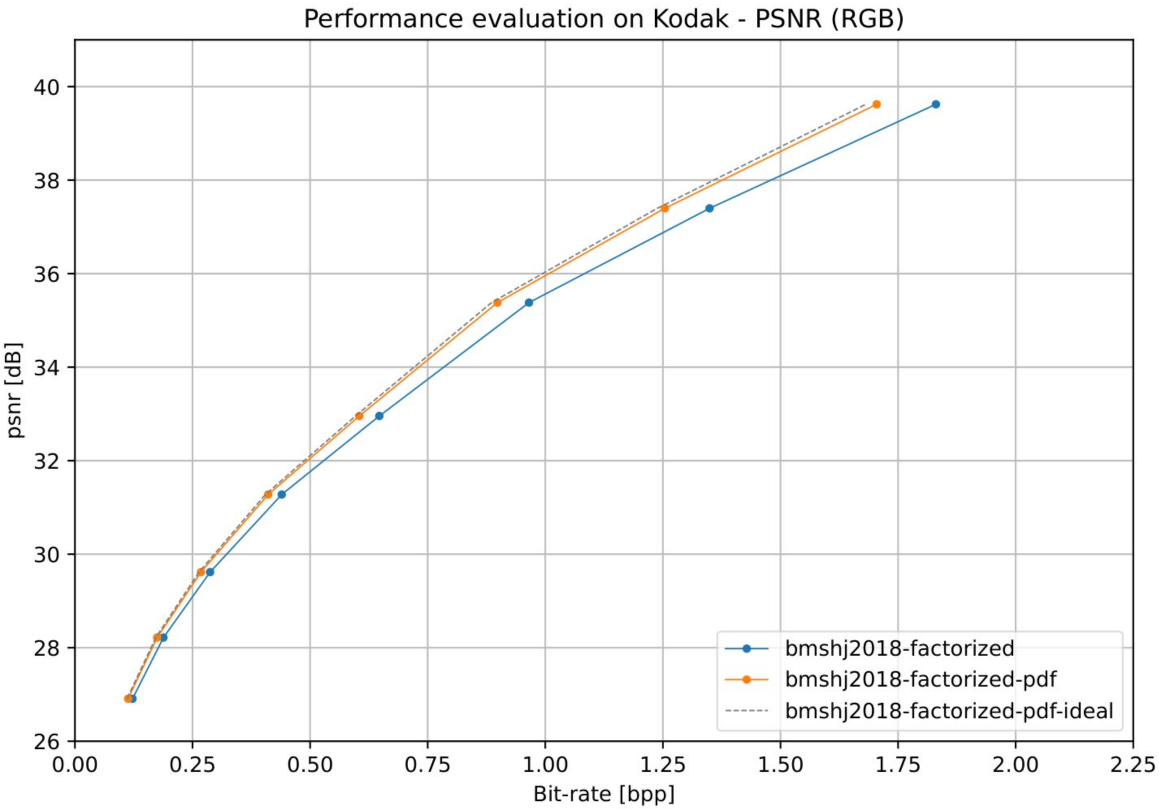
$$
\lambda_q = \frac{H_{\boldsymbol{x},\text{trained}} W_{\boldsymbol{x},\text{trained}}}{H_{\boldsymbol{x},\text{target}} W_{\boldsymbol{x},\text{target}}}
$$

256x256 “trained”
768x512 “target”

→ $\lambda_q = \frac{1}{6}$

Results

Used pretrained g_a, g□.
Froze g_a, g□ parameters.
Only trained pdf model.



Results

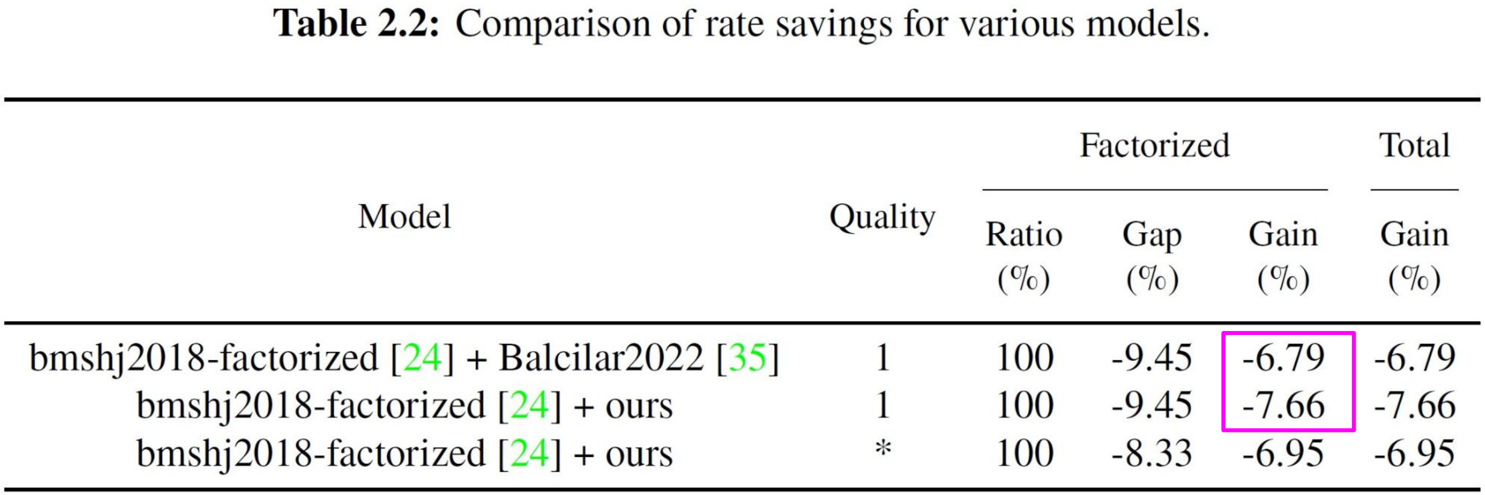


Table 2.2: Comparison of rate savings for various models.

Model	Quality	Factorized			Total
		Ratio (%)	Gap (%)	Gain (%)	Gain (%)
bmshj2018-factorized [24] + Balcilar2022 [35]	1	100	-9.45	-6.79	-6.79
bmshj2018-factorized [24] + ours	1	100	-9.45	-7.66	-7.66
bmshj2018-factorized [24] + ours	*	100	-8.33	-6.95	-6.95

Architecture comparison

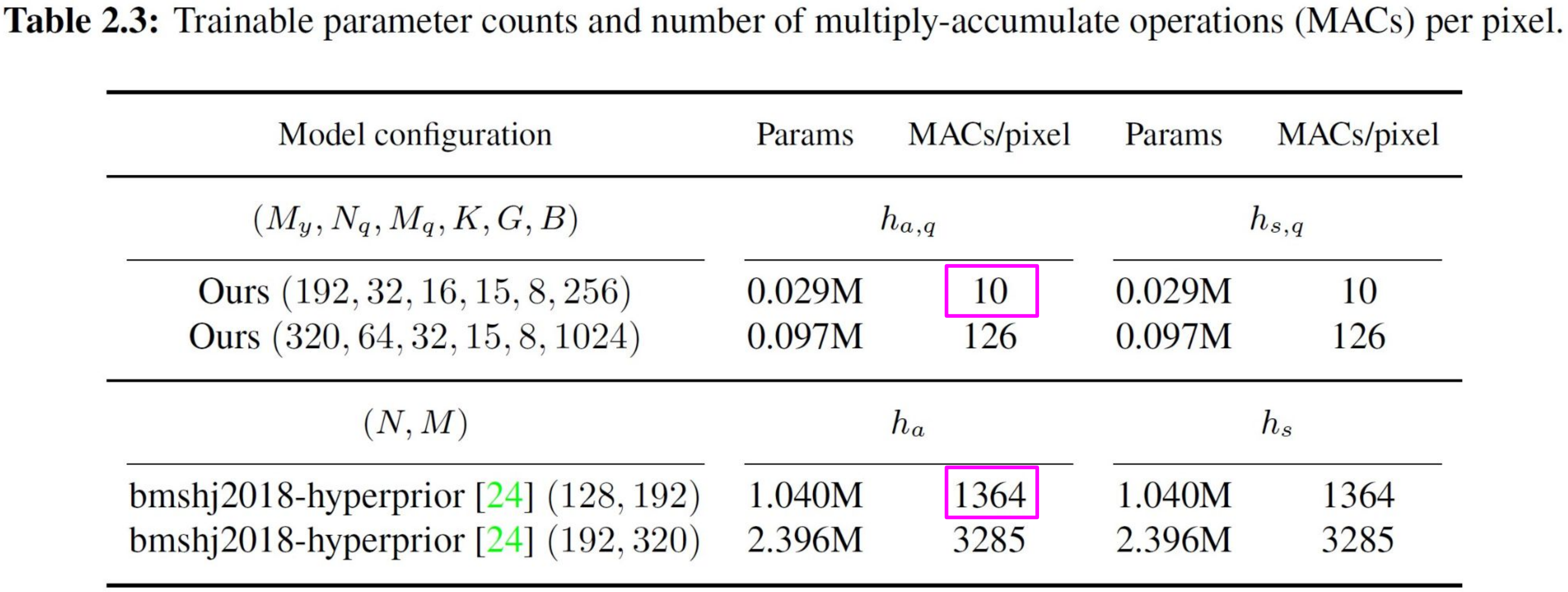


Table 2.3: Trainable parameter counts and number of multiply-accumulate operations (MACs) per pixel.

Model configuration	Params	MACs/pixel	Params	MACs/pixel
(M_y, N_q, M_q, K, G, B)	$h_{a,q}$		$h_{s,q}$	
Ours $(192, 32, 16, 15, 8, 256)$	0.029M	$\boxed{10}$	0.029M	10
Ours $(320, 64, 32, 15, 8, 1024)$	0.097M	126	0.097M	126
(N, M)	h_a		h_s	
bmshj2018-hyperprior [24] $(128, 192)$	1.040M	$\boxed{1364}$	1.040M	1364
bmshj2018-hyperprior [24] $(192, 320)$	2.396M	3285	2.396M	3285

Section summary

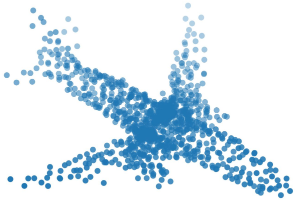
- Proposed a new method for the compression of encoding distributions.
- Our method achieves -7% rate vs -8.3% ideal for the factorized entropy model.

Future work:

- Working fully end-to-end dynamically adaptive entropy bottleneck.
- Adaptive distribution correction for Gaussian conditional.
- Non-parametric distribution modeling.

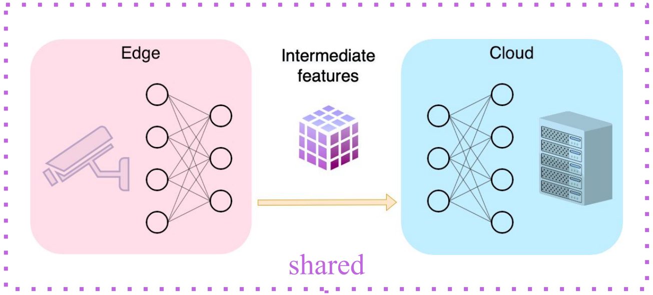
Learned Point Cloud Compression for Classification

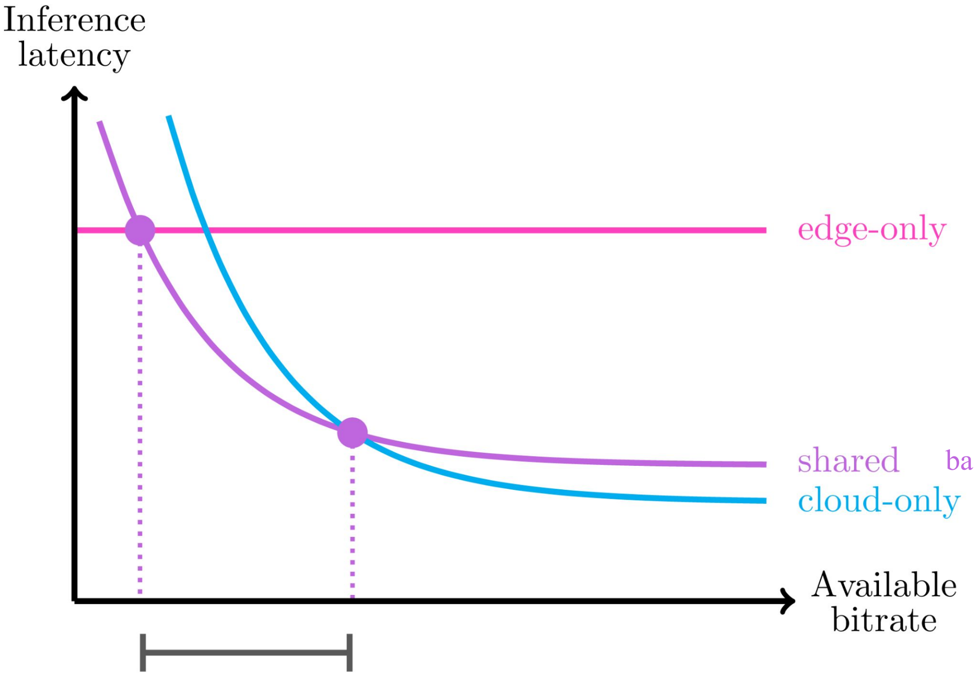
Presented at IEEE MMSP 2023   Poitiers, France



https://github.com/multimedialabsfu/learned-point-cloud-compression-for-classification

Motivation





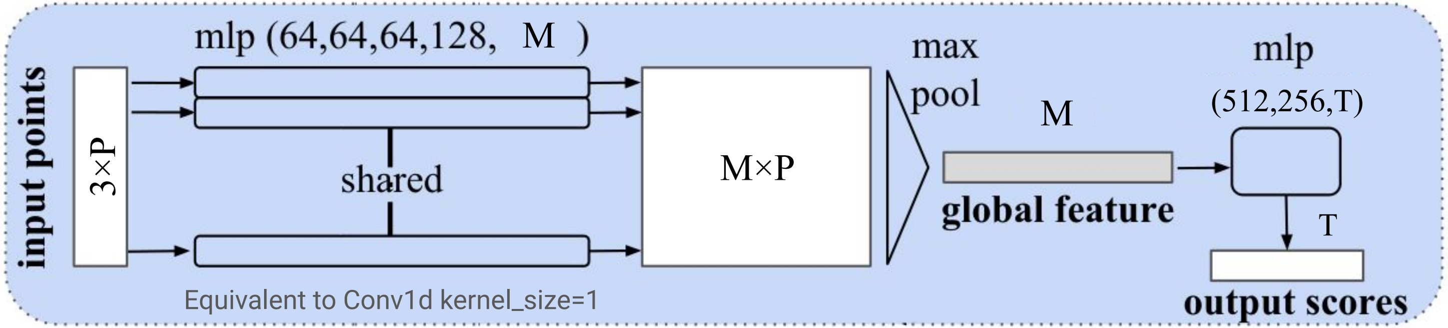
edge-only limited computational ability

shared balance of edge-cloud

cloud-only limited by available bitrate

Interval over which shared inference is fastest

PointNet



$(h(\mathrm{x}_1), \ldots, h(\mathrm{x}_{\square}))$ where $h : \mathbb{R}^3 \rightarrow \mathbb{R}^M$
shared single-point-input function

$\pi : \mathbb{R}^{M \times P} \rightarrow \mathbb{R}^M$
input permutation invariant function

$\gamma : \mathbb{R}^M \rightarrow \mathbb{R}^T$
any function

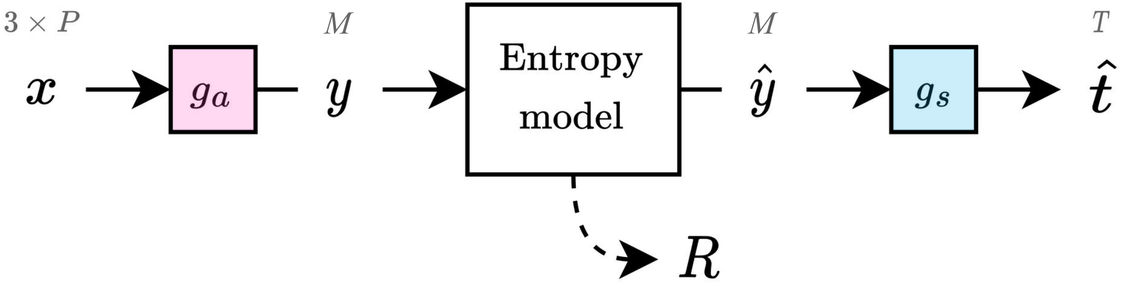
PointNet
input permutation invariant function

$$\mathrm{PointNet}(x_1, \ldots, x_P) = (\gamma \circ \pi)(h(x_1), \ldots, h(x_P))$$

Figure adapted from Qi *et al.* “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” *CVPR*, 2017.

Architecture

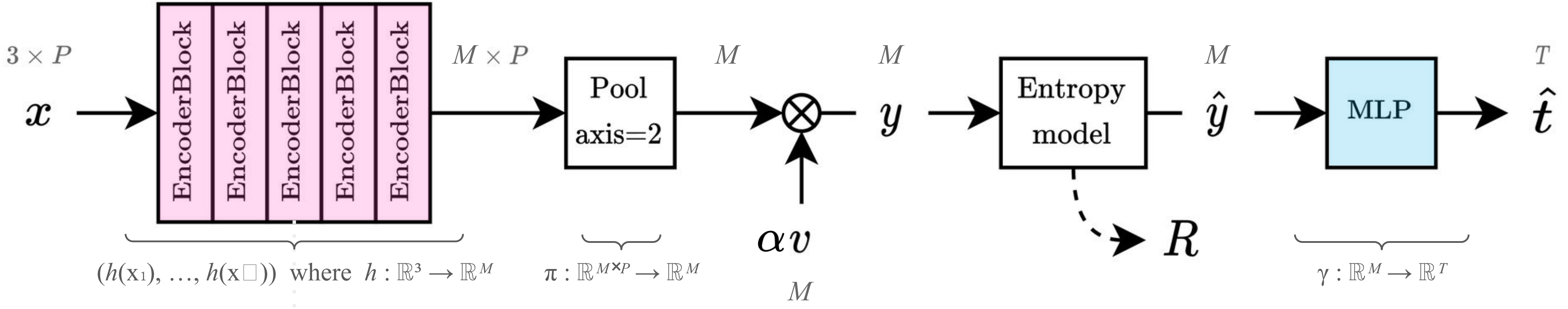
Standard compression architecture, except that the output is a classification label vector.



(for shared edge-cloud inference)

$$\mathrm{PointNet}(x_1, \ldots, x_P) = (\gamma \circ \pi)(h(x_1), \ldots, h(x_P))$$

Architecture



Multiply by reparameterized gain vector for faster convergence, where $\alpha = 10$.

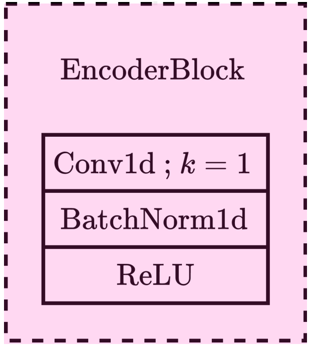


TABLE I
LAYER SIZES AND MAC COUNTS FOR VARIOUS PROPOSED CODEC TYPES

Proposed codec	Encoder layer sizes	Decoder layer sizes	Encoder MAC/pt	Decoder MAC
full	64 64 64 128 1024	512 256 40	150k	670k
lite	8 8 16 16/2 32/4	512 256 40	0.47k	160k
micro	16	512 256 40	0.048k	150k

*Format: “out channels/groups”

Experimental setup

- Dataset: sampled point clouds from ModelNet40 object meshes.
- Loss: $\mathcal{L} = R + \lambda \cdot D(\boldsymbol{t}, \hat{\boldsymbol{t}})$

ModelNet40 object meshes (before sampling).

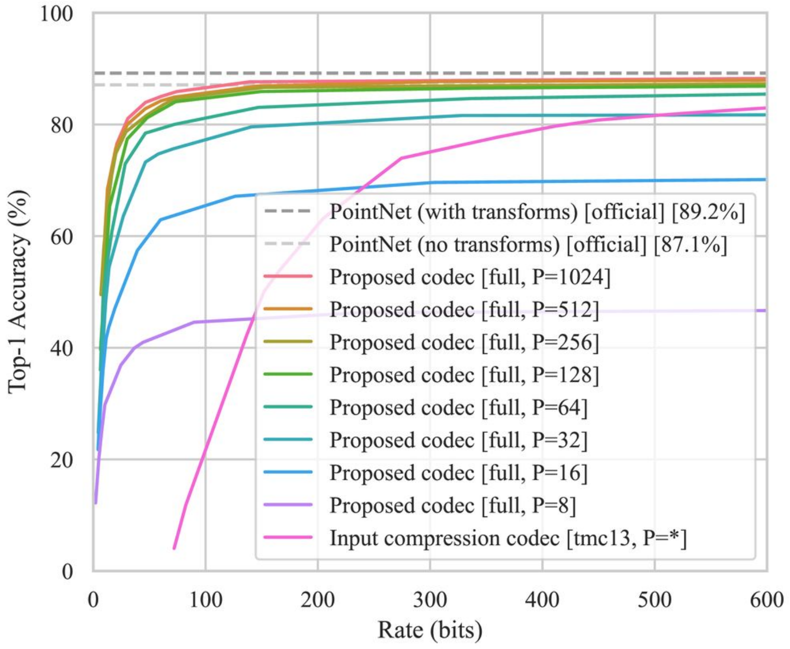
Trained separate models for various tuples $(\lambda, P, \text{ArchitectureSize})$:

- Varying R-D tradeoff $\lambda \in [10, 16000]$
- Number of input points $P \in \mathcal{P} = \{8, 16, 32, 64, 128, 256, 512, 1024\}$
- “full”, “lite”, “micro” architecture sizes

[ModelNet40]: “3D ShapeNets: A Deep Representation for Volumetric Shapes,” *CVPR*, 2015.

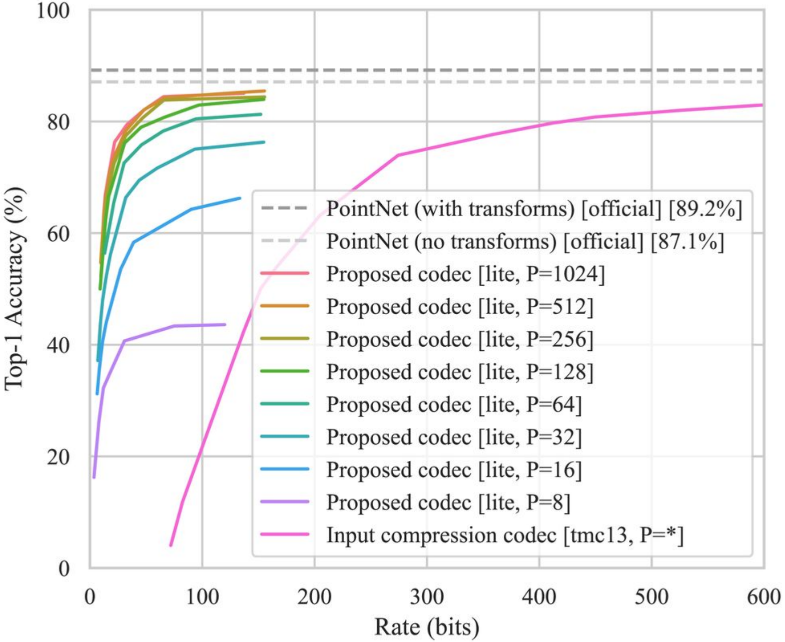
Results: rate-accuracy curves

Our codec does better than the non-specialized codec.

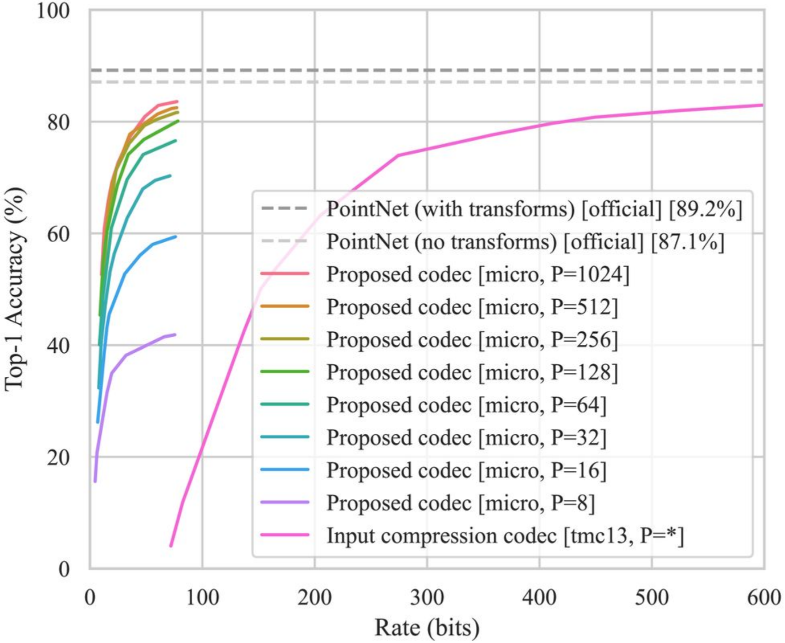


(a) “full” codec

Results: rate-accuracy curves



(b) “lite” codec



(c) “micro” codec

Results

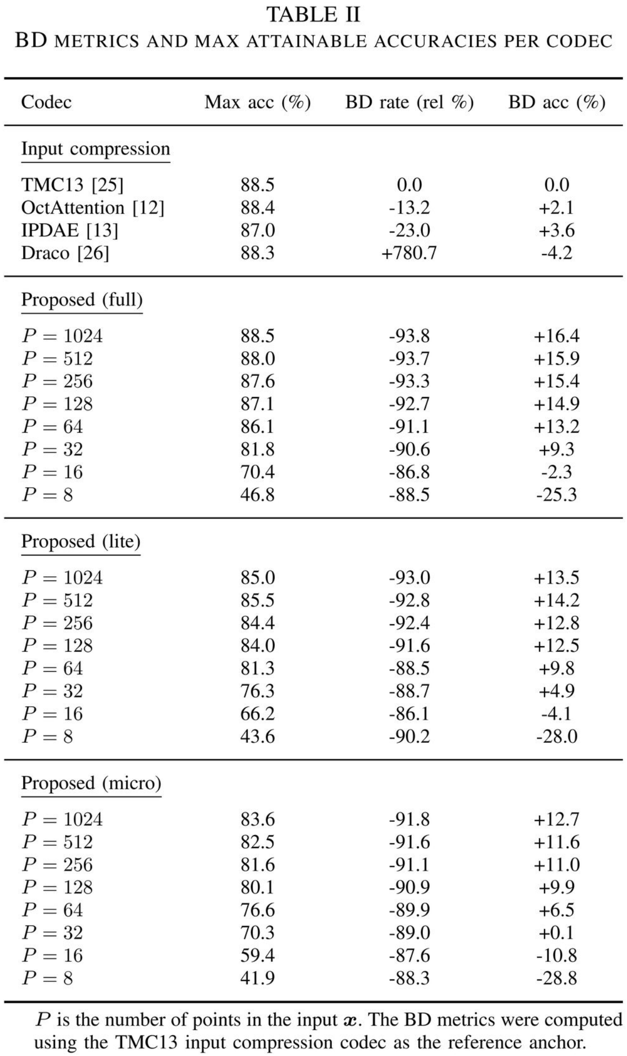


TABLE II

BD METRICS AND MAX ATTAINABLE ACCURACIES PER CODEC

Codec	Max acc (%)	BD rate (rel %)	BD acc (%)
Input compression			
TMC13 [25]	88.5	0.0	0.0
OctAttention [12]	88.4	-13.2	+2.1
IPDAE [13]	87.0	-23.0	+3.6
Draco [26]	88.3	+780.7	-4.2
Proposed (full)			
$P = 1024$	88.5	-93.8	+16.4
$P = 512$	88.0	-93.7	+15.9
$P = 256$	87.6	-93.3	+15.4
$P = 128$	87.1	-92.7	+14.9
$P = 64$	86.1	-91.1	+13.2
$P = 32$	81.8	-90.6	+9.3
$P = 16$	70.4	-86.8	-2.3
$P = 8$	46.8	-88.5	-25.3
Proposed (lite)			
$P = 1024$	85.0	-93.0	+13.5
$P = 512$	85.5	-92.8	+14.2
$P = 256$	84.4	-92.4	+12.8
$P = 128$	84.0	-91.6	+12.5
$P = 64$	81.3	-88.5	+9.8
$P = 32$	76.3	-88.7	+4.9
$P = 16$	66.2	-86.1	-4.1
$P = 8$	43.6	-90.2	-28.0
Proposed (micro)			
$P = 1024$	83.6	-91.8	+12.7
$P = 512$	82.5	-91.6	+11.6
$P = 256$	81.6	-91.1	+11.0
$P = 128$	80.1	-90.9	+9.9
$P = 64$	76.6	-89.9	+6.5
$P = 32$	70.3	-89.0	+0.1
$P = 16$	59.4	-87.6	-10.8
$P = 8$	41.9	-88.3	-28.8

P is the number of points in the input $\boldsymbol{x}$. The BD metrics were computed using the TMC13 input compression codec as the reference anchor.

Encoder:
150 kMAC/pt

Decoder:
670 kMAC

Encoder
0.47 kMAC/pt

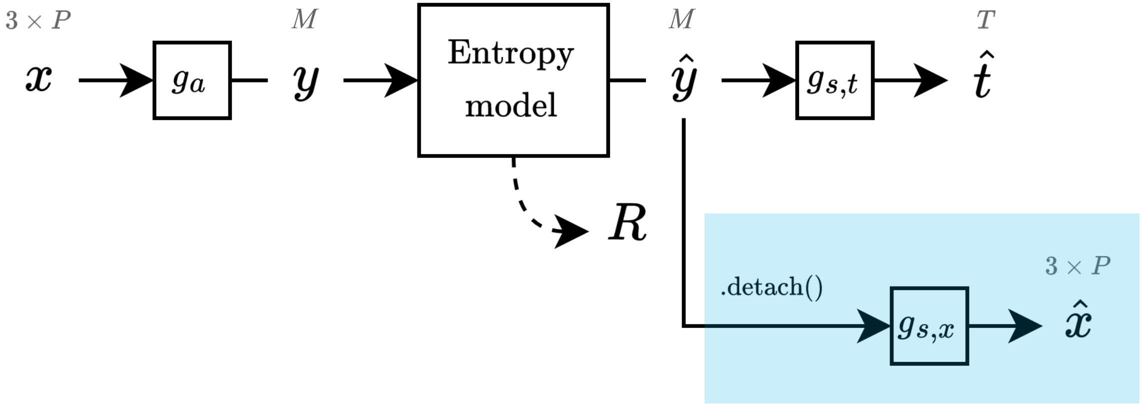
Decoder:
160 kMAC

Encoder:
0.048 kMAC/pt

Decoder:
150 kMAC

Reconstruction network (for visualization only)

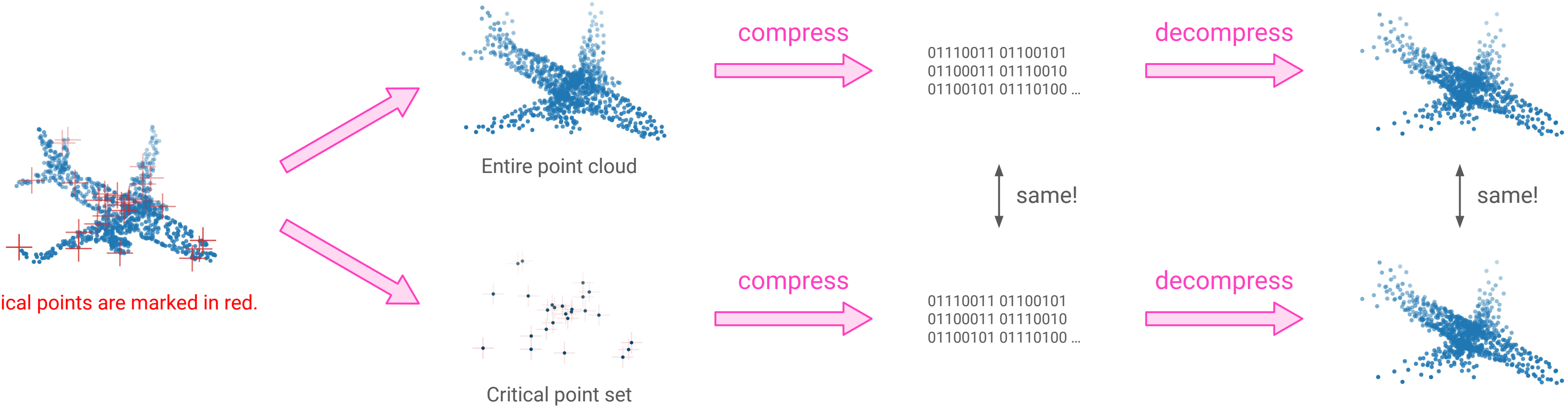
We trained an auxiliary reconstruction network on the loss $\mathcal{L} = D(\boldsymbol{x}, \hat{\boldsymbol{x}})$, where D is Chamfer distance. Detached so gradients are not propagated to $\hat{\boldsymbol{y}}$.



Reconstruction network

Critical point set

For a specific codec, the **critical point set** is a minimal subset of the input point cloud that generates the exact same compressed bitstream as the input point cloud.



Critical points are marked in red.

Entire point cloud

compress

01110011 01100101
01100011 01110010
01100101 01110100 ...

decompress

same!

same!

Critical point set

compress

01110011 01100101
01100011 01110010
01100101 01110100 ...

decompress

Reconstructions

80% classification accuracy achieved at:

Codec	Rate
ideal	3.2 bits
full	30 bits
lite	40 bits
micro	50 bits

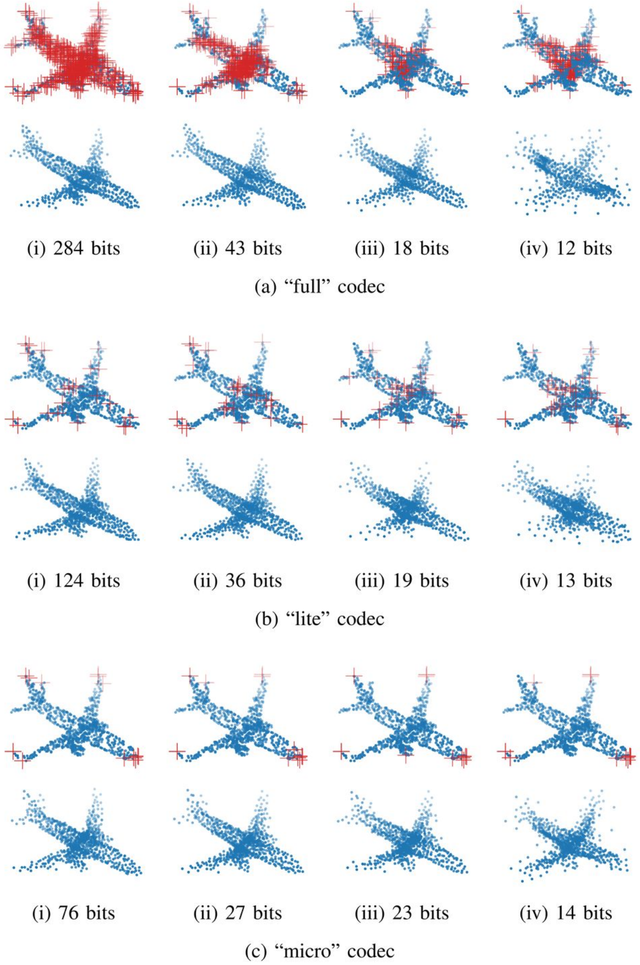
computed via Blahut-Arimoto

100% accuracy lower bound on rate for 40 balanced classes:

$$\log_2(40) \approx 5.3 \text{ bits}$$

Recall: $h(x)$ is applied to each point independently. No information mixing, except for the max pooling operation!

Contrast with traditional MLP classifier that mixes information to achieve low rate.



(i) 284 bits (ii) 43 bits (iii) 18 bits (iv) 12 bits

(a) “full” codec

(i) 124 bits (ii) 36 bits (iii) 19 bits (iv) 13 bits

(b) “lite” codec

(i) 76 bits (ii) 27 bits (iii) 23 bits (iv) 14 bits

(c) “micro” codec

Fig. 4. Reconstructions of a sample airplane 3D model from the ModelNet40 test set for various codecs and bitrates. For each reconstruction, its corresponding reference point cloud is marked with *critical points* in red.

up to 1024 critical points

up to 32 critical points

up to 16 critical points

Section summary

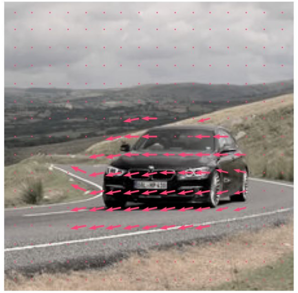
- New codec for point cloud classification.
- Our codec improves in rate-accuracy vs traditional methods.
- Fast "lite" and "micro" encoders.

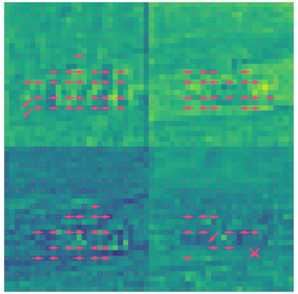
Future work:

- Real-world datasets.
- Point cloud segmentation and object detection.
- Scalable and multi-task point cloud compression.

Analysis of Latent Space Motion

Presented at IEEE ICASSP 2021   Toronto, Canada





Problem statement

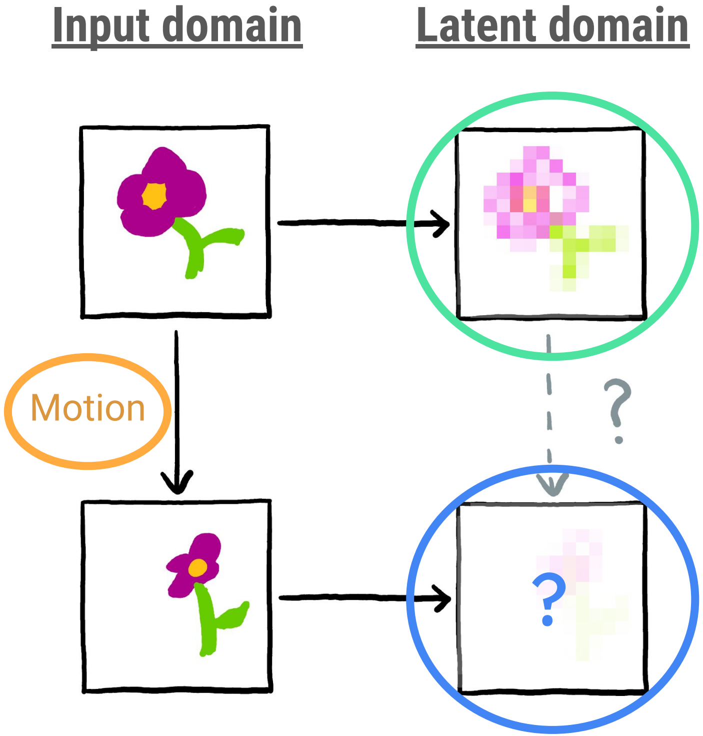
Motivation:

Video compression in the latent domain.

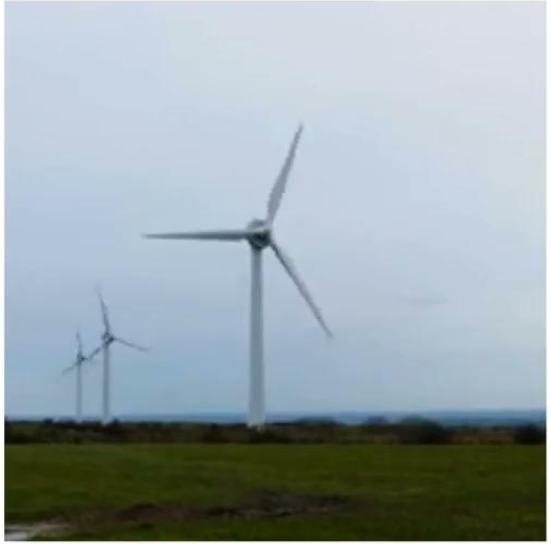
Question:

Given a reference tensor and the motion between consecutive input frames, can we determine:

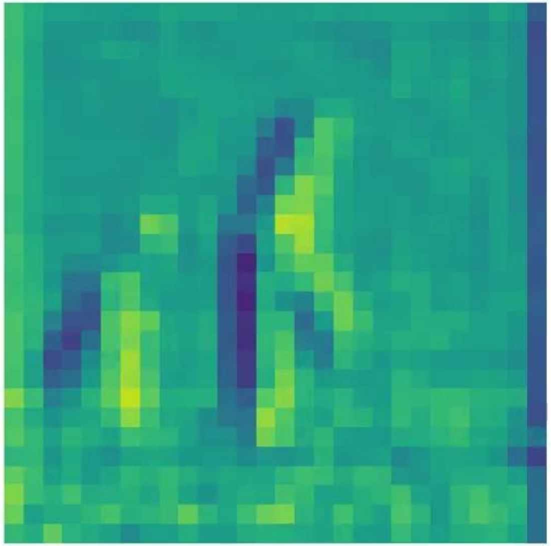
- …the motion between tensors?
- …the next tensor?



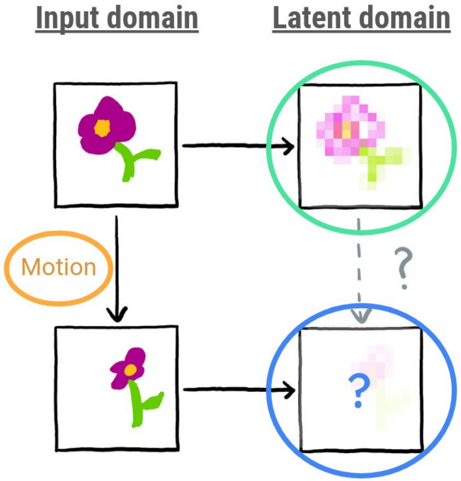
Input domain



Latent domain



Tensor reconstruction experiments



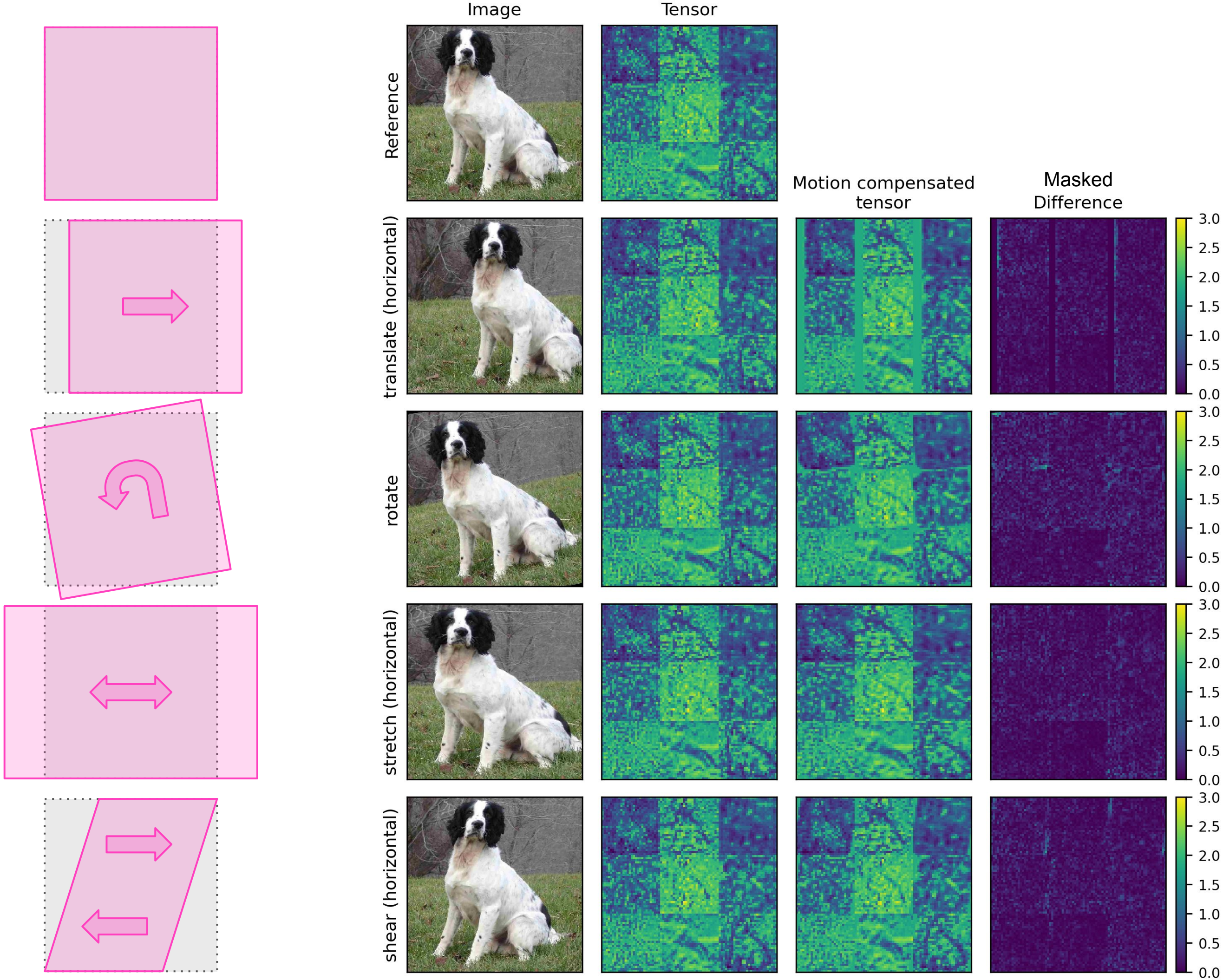
- Assume: latent motion is rescaled input motion.

$$\tilde{v}(x, y) \approx v(s^k x, s^k y)/s^k \longleftarrow \text{\# of strides}$$

- Using latent motion, warp reference tensor to predict next tensor.

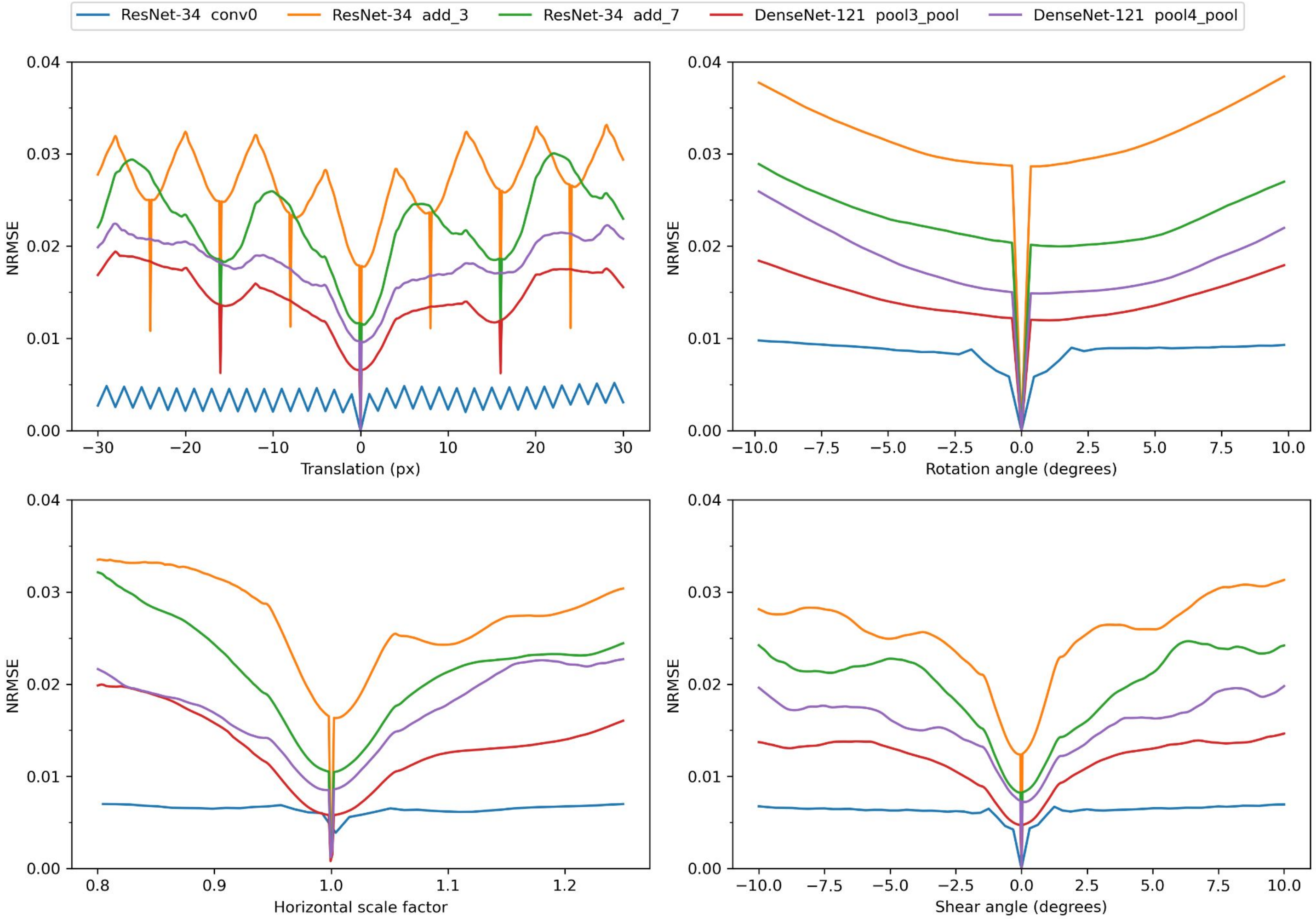
- Calculate normalized root mean square error (NRMSE) between predicted next tensor and actual next tensor.

$$\text{NRMSE} = \frac{1}{R}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}$$



Residuals for predicted motion compensated tensors under various input domain transformations.

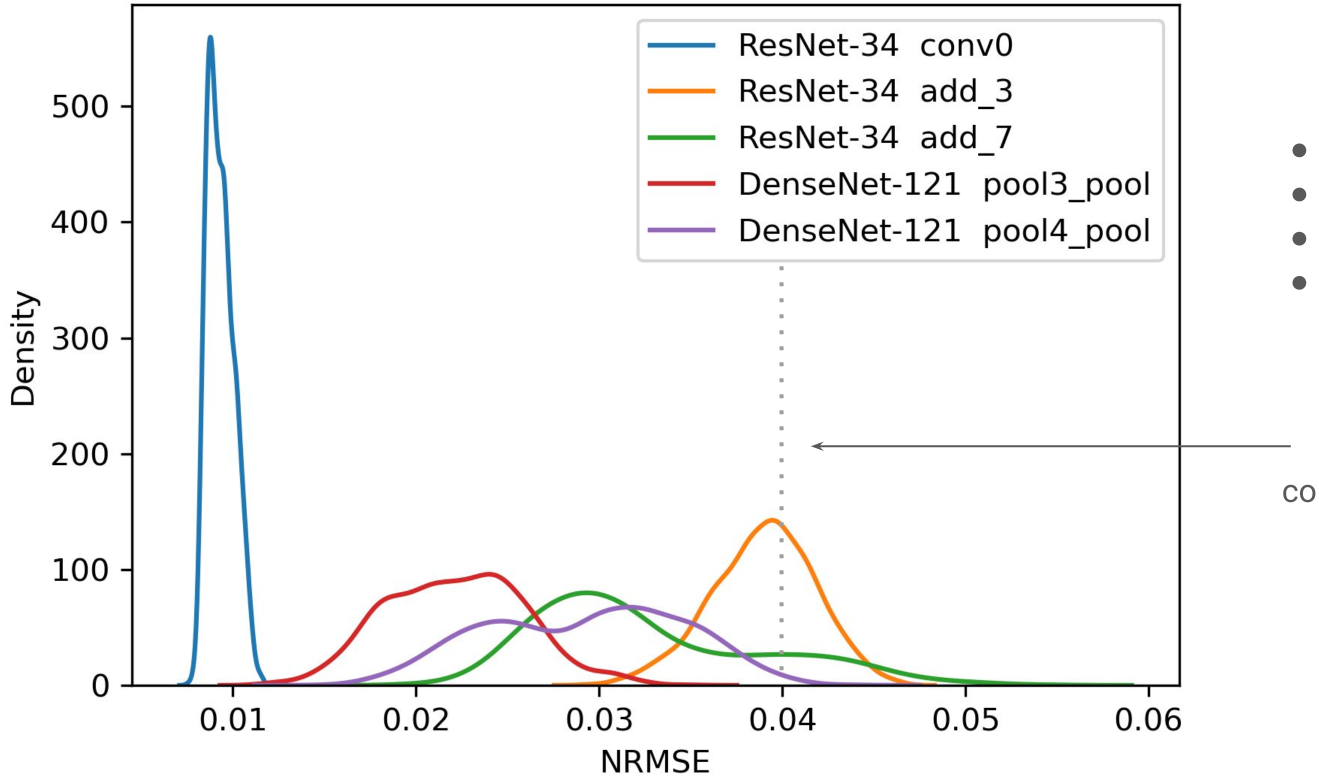
We computed the NRMSE using these masked residuals.



NRMSE in “p-frame” residual for primitive transformations for different models/layers.

Large rotation angles cause increase in error.

Prediction error during random motion



Random transformation composed of:

- xy translation (± 32 px)
- xy scaling (0.95 – 1.05x)
- xy shearing (± 5°)
- xy rotation (± 10°)

NRMSE of 0.04 roughly corresponds to 28 dB PSNR in the image domain.

Section summary

- Validated simple relationship of motion between consecutive tensors.

$$\tilde{v}(x,y) \approx v(s^k x, s^k y)/s^k$$

- Prediction error for small transformations within input is 4% NRMSE.

Applications:

- Learned codecs that motion-warp the latent directly.

In other words: input domain → latent domain → warp → input domain
“Scale-space flow”: input domain → latent domain → input domain → warp

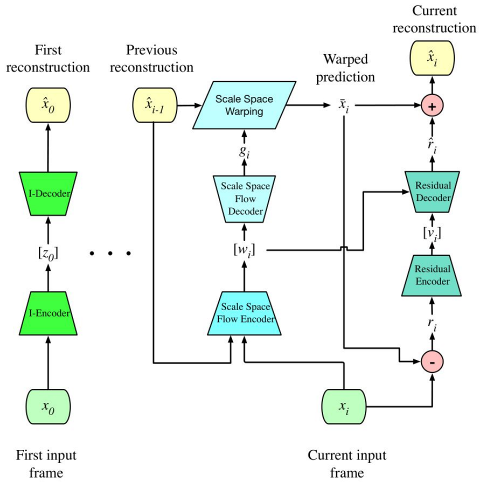


Figure from Agustsson *et al.* “Scale-space flow for end-to-end optimized video compression,” *CVPR*, 2020.

Concluding remarks

- Proposed learned compression of the “encoding distribution” itself.
- Introduced shared client-server inference for point clouds using a classification-specialized codec.
- Investigated error in motion models of the latent space.
  Useful for designing learned video compression codecs.

Learned compression shows promise, though we must reduce its complexity for it to become practical. Some of the work presented takes steps in this direction.

Publications and other work

1. **M. Ulhaq** and I. V. Bajić, “Learned point cloud compression for classification,” in *Proc. IEEE MMSP*, 2023. Available: https://arxiv.org/abs/2308.05959
2. E. Özyılkan, **M. Ulhaq**, H. Choi, and F. Racapé, “Learned disentangled latent representations for scalable image coding for humans and machines,” in *Proc. IEEE DCC*, 2023, pp. 42–51. Available: https://arxiv.org/abs/2301.04183
3. H. Choi, F. Racapé, S. Hamidi-Rad, **M. Ulhaq**, and S. Feltman, “Frequency-aware learned image compression for quality scalability,” in *Proc. IEEE VCIP*, 2022, pp. 1–5. doi: 10.1109/VCIP56404.2022.10008818.
4. S. R. Alvar, **M. Ulhaq**, H. Choi, and I. V. Bajić, “Joint image compression and denoising via latent-space scalability,” *Frontiers in Signal Processing*, vol. 2, 2022, doi: 10.3389/frsip.2022.932873.
5. **M. Ulhaq** and I. V. Bajić, “Latent space motion analysis for collaborative intelligence,” in *Proc. IEEE ICASSP*, 2021, pp. 8498–8502. doi: 10.1109/ICASSP39728.2021.9413603.
6. **M. Ulhaq** and I. V. Bajić, “ColliFlow: A library for executing collaborative intelligence graphs,” demoed at *NeurIPS*, 2020. Available: https://yodaembedding.github.io/neurips-2020-demo/
7. **M. Ulhaq** and F. Racapé, “CompressAI Trainer.” GitHub, 2022. Available: https://github.com/InterDigitalInc/CompressAI-Trainer

Thank you