Learned Image Compression

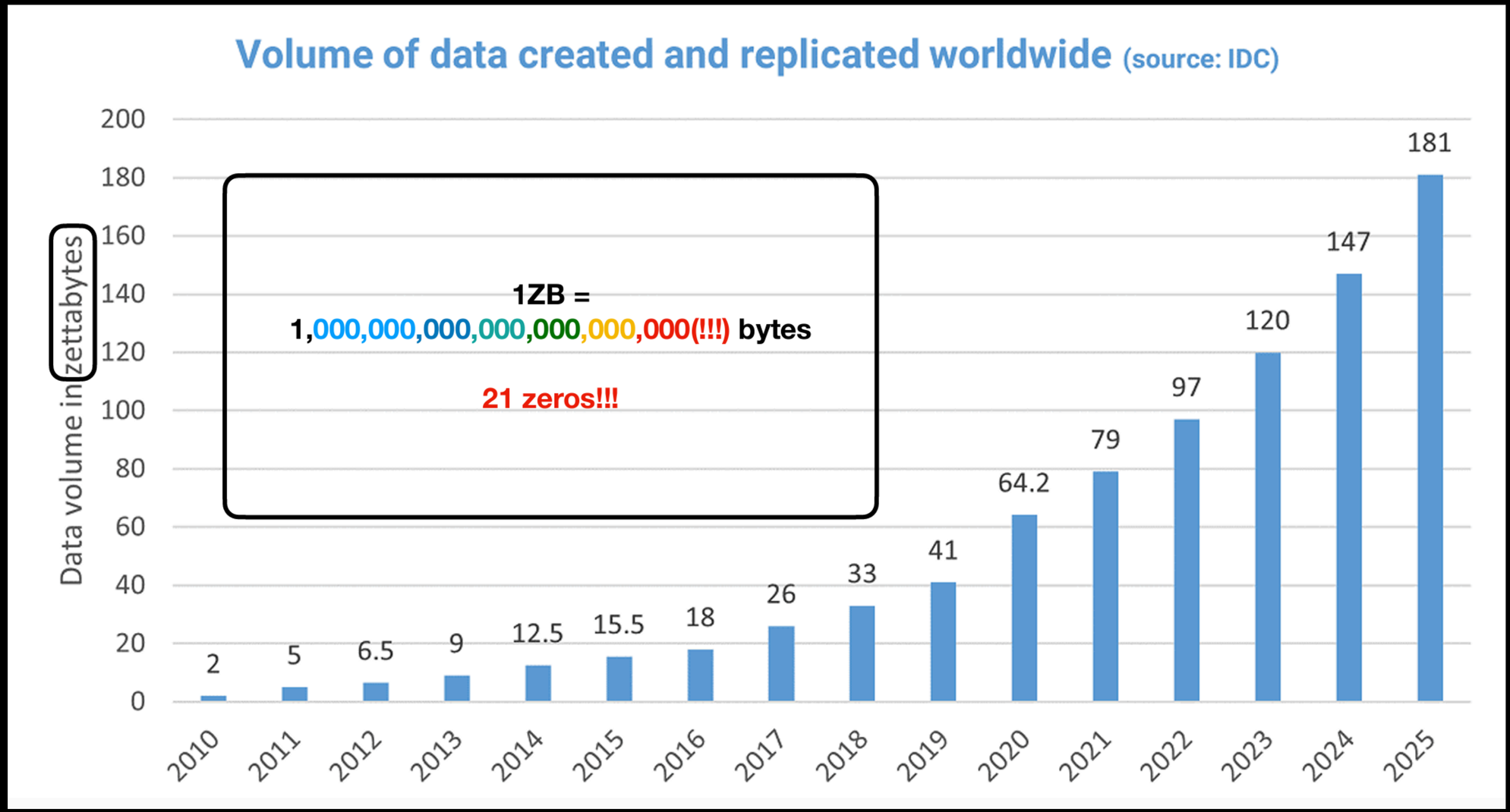
CVIP 2023 Tutorial

Shubham Chandak, Pulkit Tandon

Material based on Stanford class on "EE 274: Data Compression"

Joint with: Kedar Tatwawadi, Tsachy Weissman

Why compression?



Why multimedia compression?

Category	Proportion of Internet Data Traffic
Video	53.72%
Social	12.69%
Gaming	9.86%
Web browsing	5.67%
Messaging	5.35%
Marketplace	4.54%
File sharing	3.74%
Cloud	2.73%
VPN	1.39%
Audio	0.31%

Most of these involve images directly or indirectly

https://explodingtopics.com/blog/data-generated-per-day#category

Tutorial Outline

Part I

Compression basics
+
Traditional image compression (JPEG)

Part II

Learnt image compression basics
+
Google colab tutorial
+
Challenges and outlook

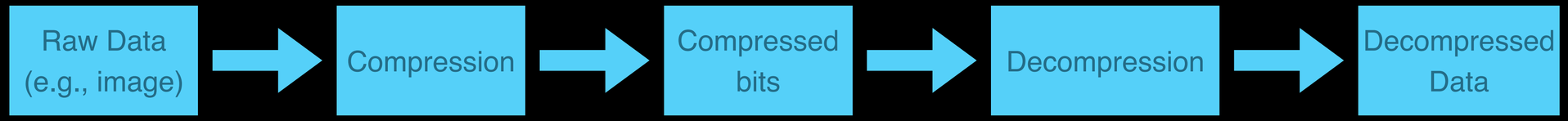
Part I

Resources

- *“The Unreasonable Effectiveness of JPEG: A Signal Processing Approach” (Youtube video -> Reducible, beautiful illustrations!)*
  https://www.youtube.com/watch?v=0me3guauqOU

- *EE398A Stanford Lecture Notes: (Bernd Girod)*
  https://web.stanford.edu/class/ee398a/handouts

What is compression?



Lossless Compression:
- decompressed data identical to raw data
- Aim: get highest compression ratio given computational constraints

Lossy Compression:
- decompressed data need not be identical to raw data
- Aim: get highest compression ratio given distortion and computational constraints

Why is compression even possible?

- Some symbols occur more often than others
  - e.g., e >> q in English
- Correlation/Predictability of symbols based on context
  - e.g., **q** followed almost always by a **u**

More predictable implies more compressible!

Why is compression even possible?

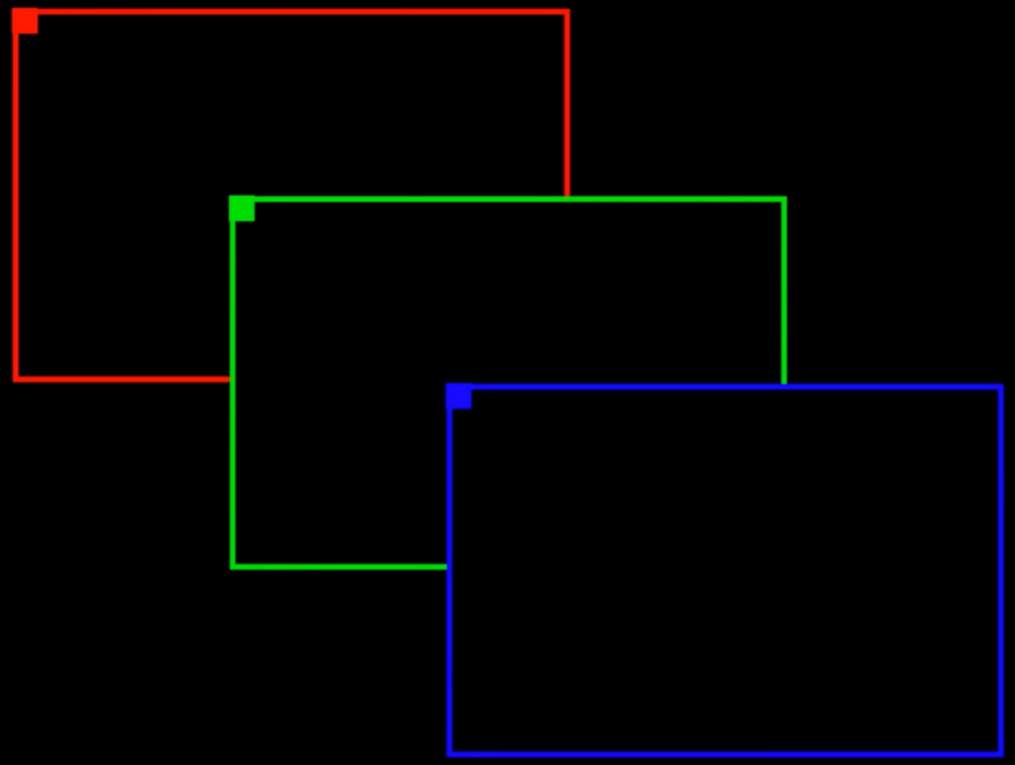
- Some symbols occur more often than others
  - e.g., e >> q in English
- Correlation/Predictability of symbols based on context
  - e.g., **q** followed almost always by a **u**

More predictable implies more compressible!

Core idea: use fewer bits for more likely symbols

What is an image?

What is an image?



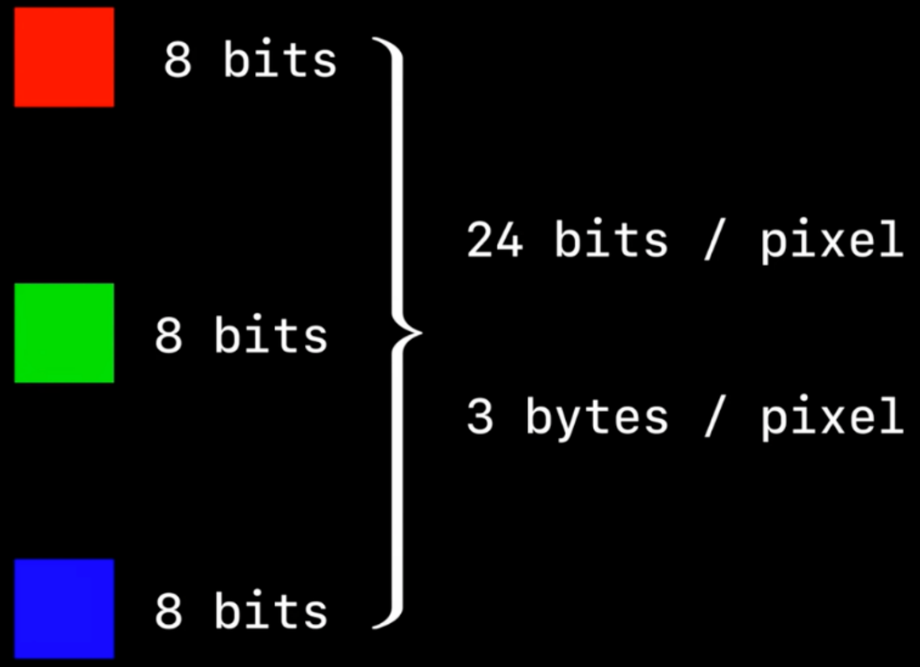


Image Compression

764x512x3 bytes
= 1.1MB!
(Uncompressed)

Image from Kodak dataset

Image Compression -> Lossless

Uncompressed -> 1.1MB
HEIC lossless -> 270KB (~4x)

Image from Kodak dataset

Image Compression -> JPEG 40x

Uncompressed -> 1.1MB
JPEG -> 27KB (~40x!)

Image from Kodak dataset

Image Compression -> JPEG 80x

Uncompressed -> 1.1MB
JPEG -> 14KB (~80x!)

Image from Kodak dataset

Image Compression -> JPEG 137x

Uncompressed -> 1.1MB
JPEG -> 8KB (~137x!)

Image from Kodak dataset

Lossy Compression

- Incredible performance gains! ~40x-137x gains without much noticeable difference (depending upon the codec)

- So ubiquitous, DSLR cameras do JPEG compression by default
  - difficult to find a “dataset” of non-compressed images!

- JPEG, JPEG2000, BPG (HEIC), AVIF, JPEG-XL, ML-based image compressors …

JPEG Image Compression

IEEE Transactions on Consumer Electronics, Vol. 38, No. 1, FEBRUARY 1992

THE JPEG STILL PICTURE COMPRESSION STANDARD

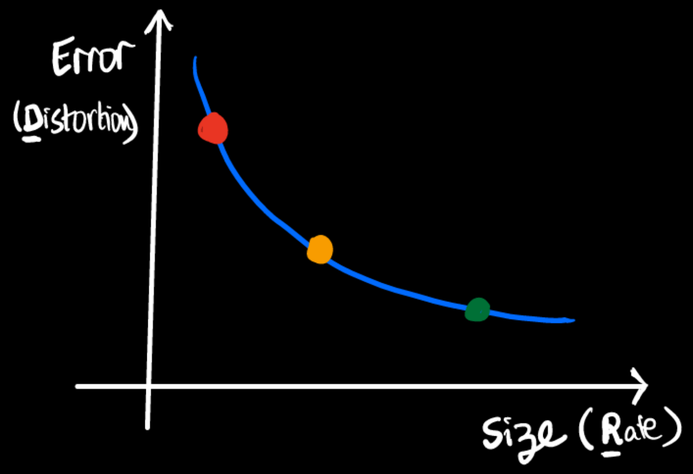
Gregory K. Wallace
Multimedia Engineering
Digital Equipment Corporation
Maynard, Massachusetts

possible exception of facsimile, digital images are not commonplace in general-purpose computing systems the way text and geometric graphics are. The majority of modern business and consumer usage of photographs and other types of images takes place through more traditional analog means.

The key obstacle for many applications is the vast amount of data required to represent a digital image directly. A digitized version of a single, color picture at TV resolution contains on the order of one million bytes; 35mm resolution requires ten times that amount. Use of digital images often is not viable due to high storage or transmission costs, even when image capture and display devices are quite affordable.

Lossy Compression -> Problem definition

Rate-Distortion Tradeoff



585 KB

280 KB

13 KB

What is the distortion metric?

Lossy Compression -> Problem definition

Distortion metric -> MSE?

MSE - basis for much of rate-distortion theory!

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Mean · Error · Squared

Lossy Compression -> Problem definition

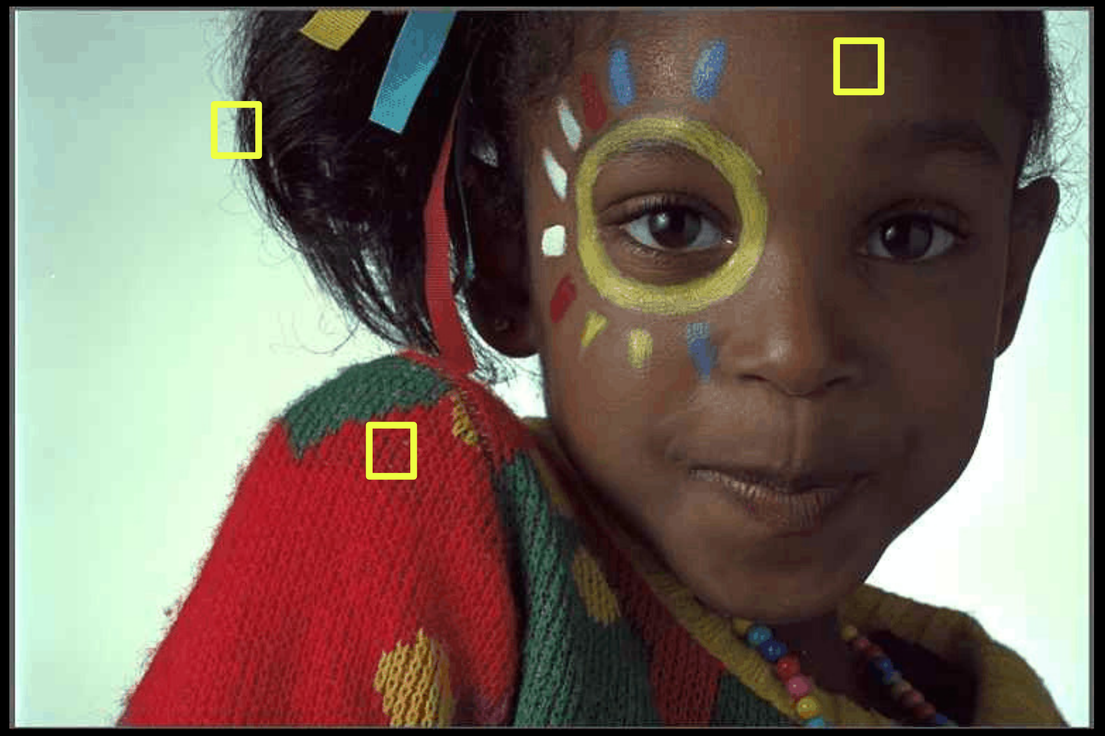
Distortion metric -> ~~MSE~~?

Lots of research into understanding “Human Perceptual loss”...

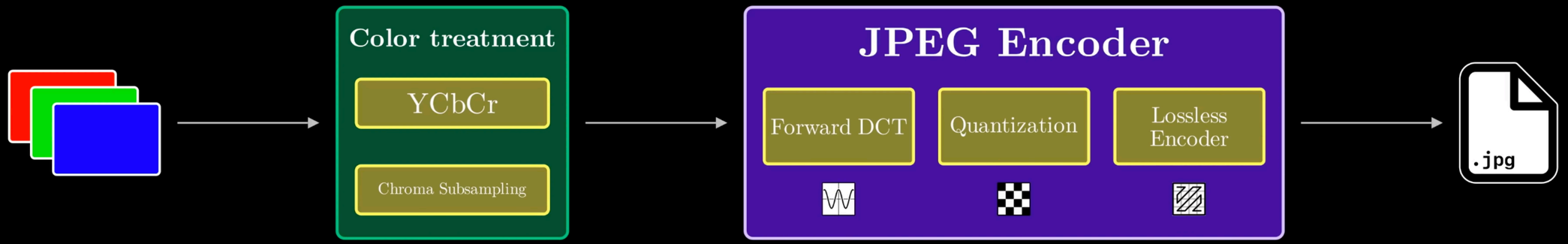
For simplicity, we will consider MSE (+heuristics)

Exploiting Spatial correlation in the data

Key Idea -> We need to somehow exploit/remove the correlation between neighboring pixels.

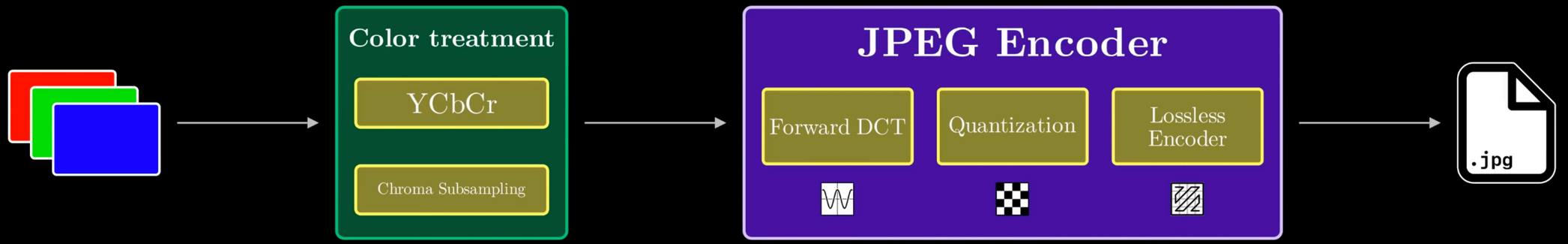


JPEG Image Compression



Optional color transform
+ color sub-sampling

JPEG Image Compression

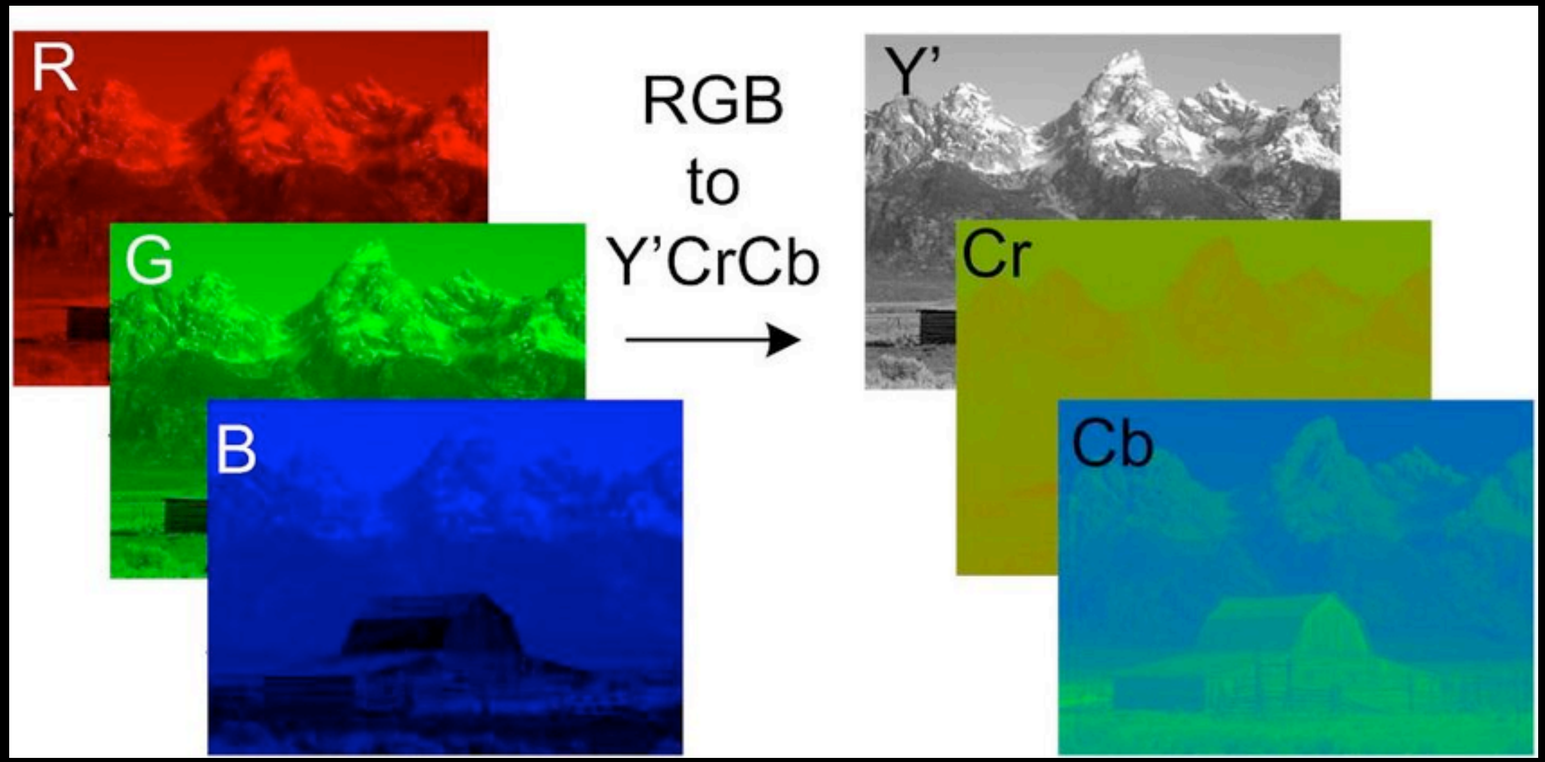


Optional color transform + color sub-sampling

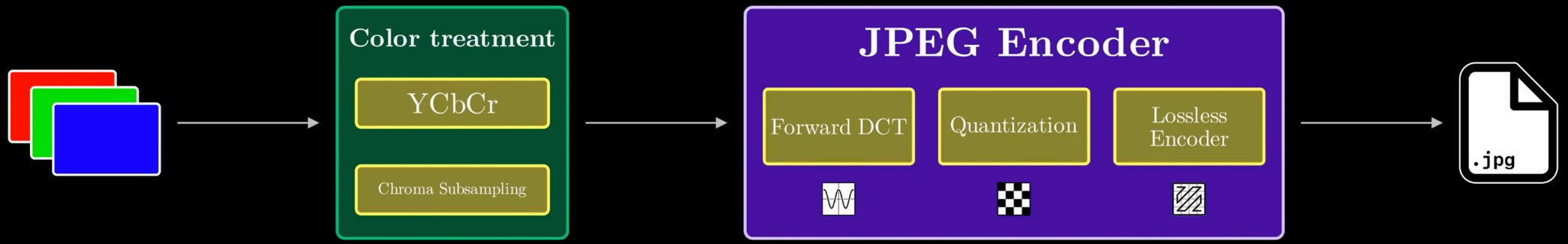
Exploit correlation across color channels and subsample according to human perception

JPEG Image Compression

Optional color transform
+ color sub-sampling

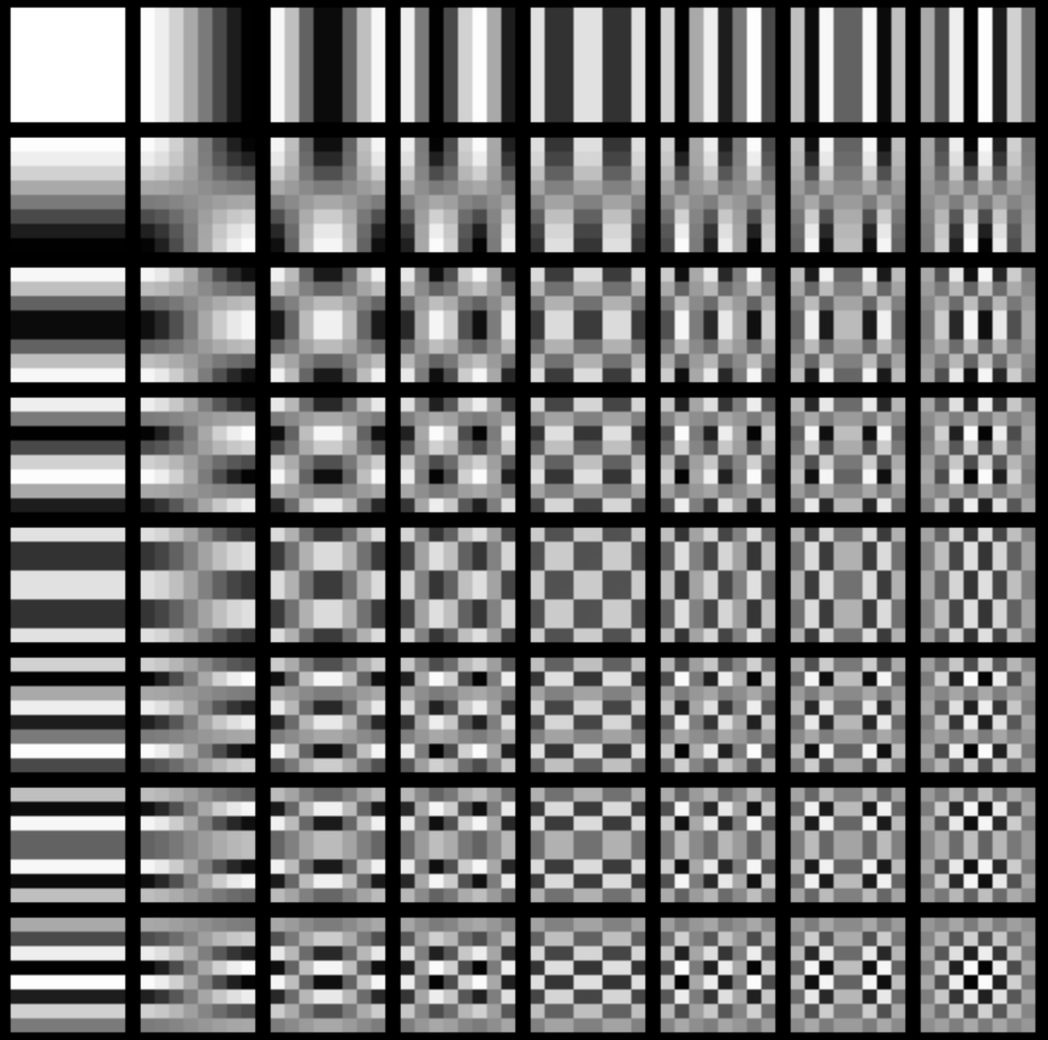


JPEG Image Compression



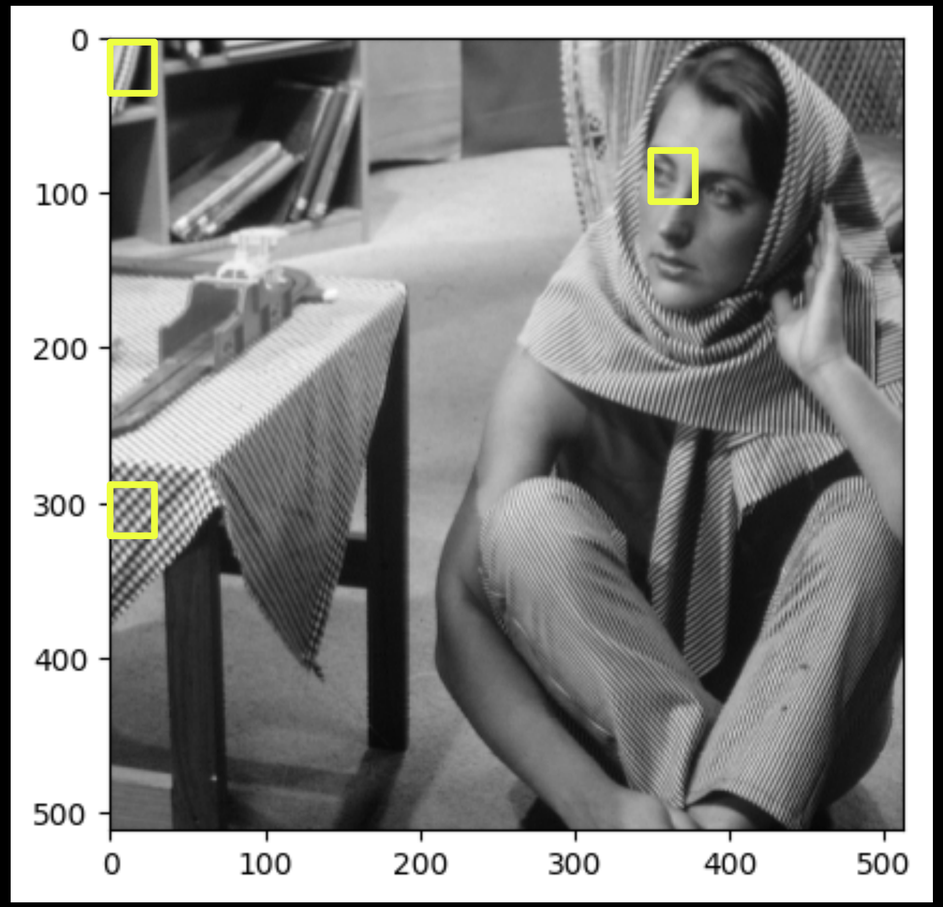
Apply DCT for spatial decorrelation

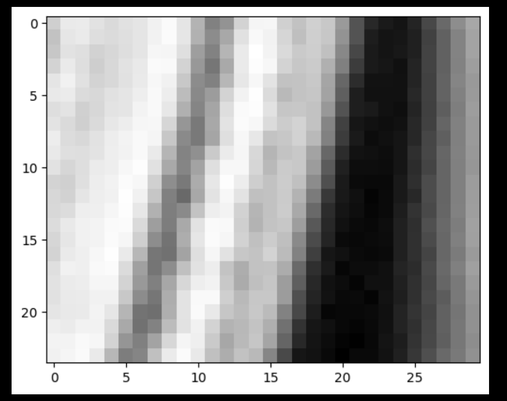
Transform Coding -> 2D-DCT vectors

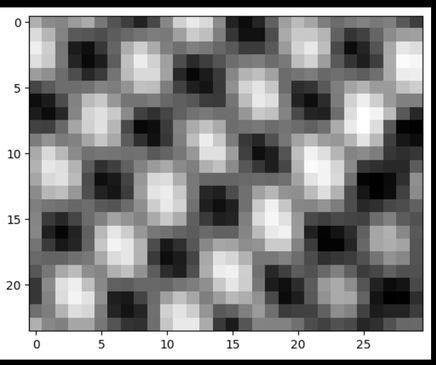


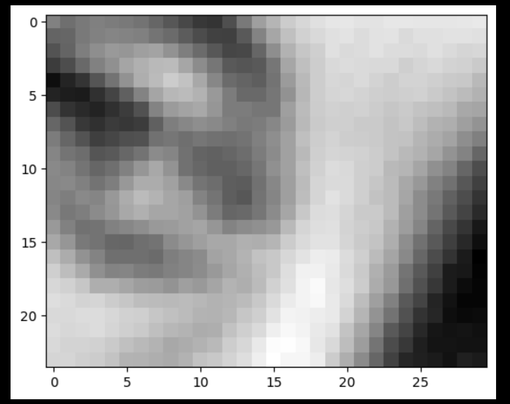
2D-DCT basis vectors for 8x8 blocks

Transform Coding -> DCT

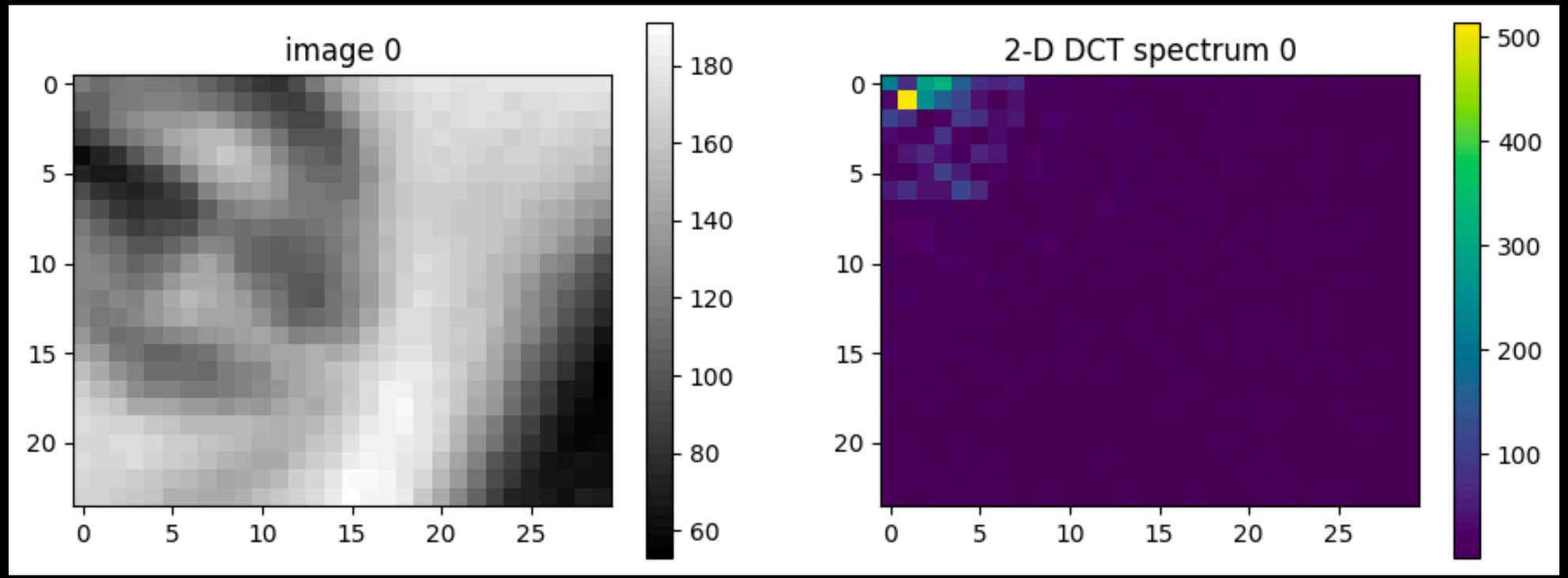






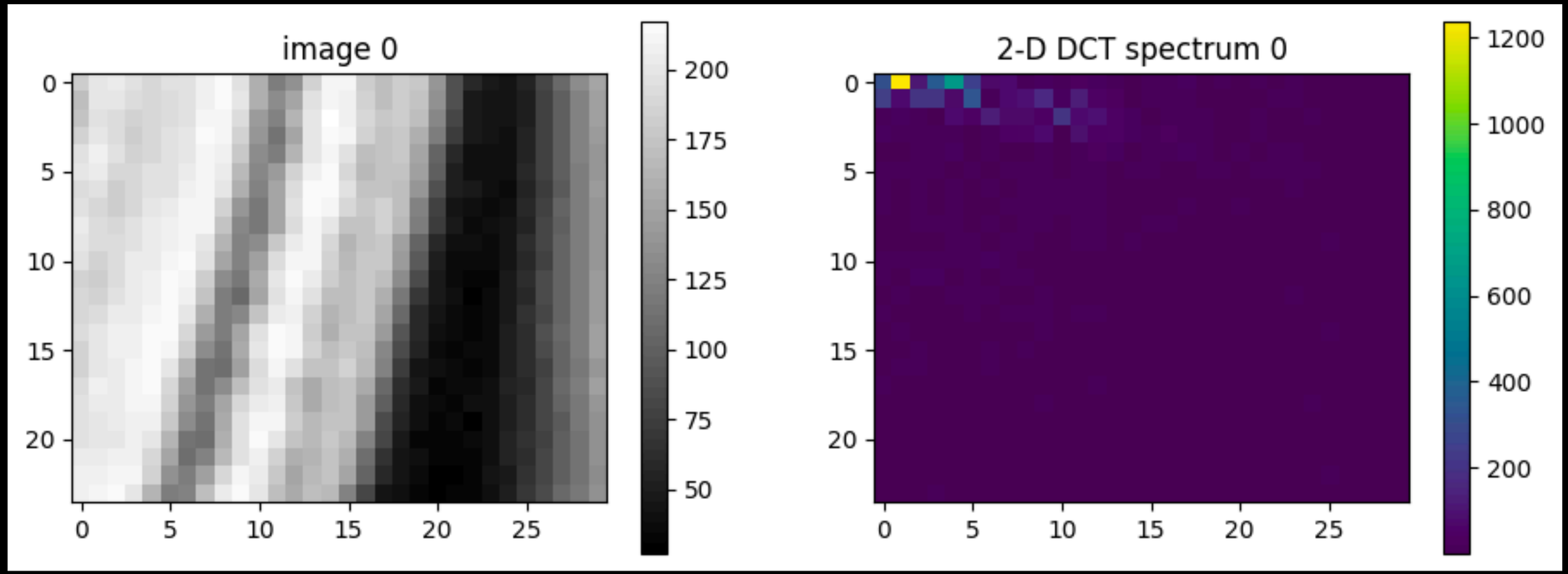


Transform Coding -> DCT



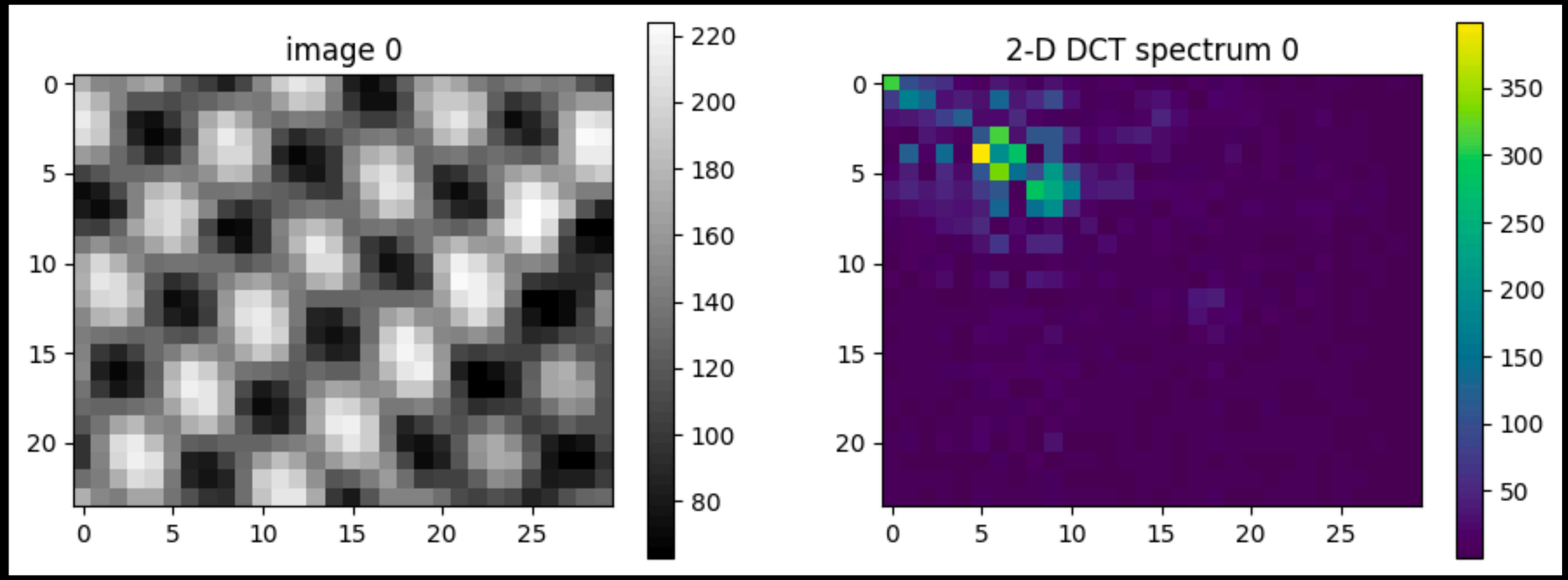
DCT -> Sparse

Transform Coding -> DCT



DCT -> Sparse

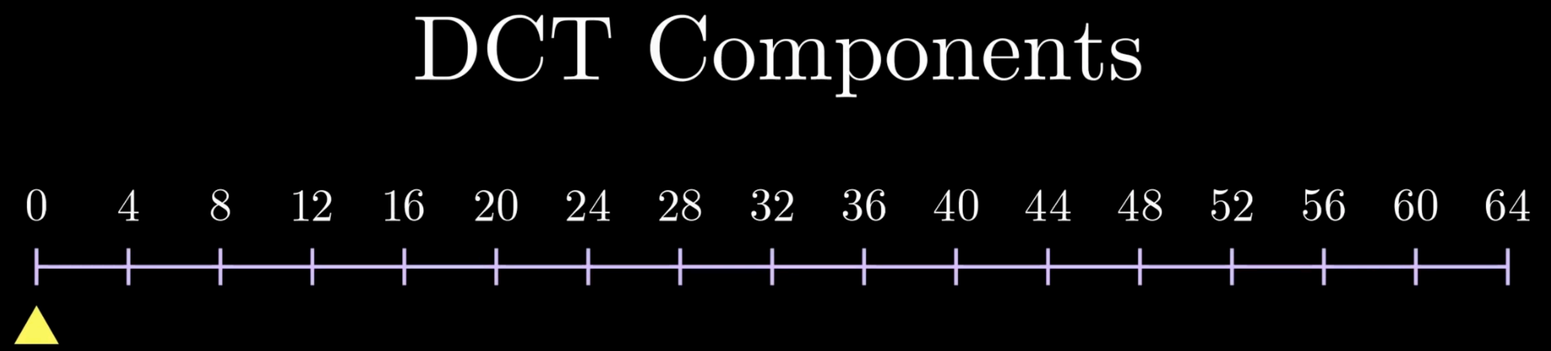
Transform Coding -> DCT



DCT -> Sparse (but higher frequencies)

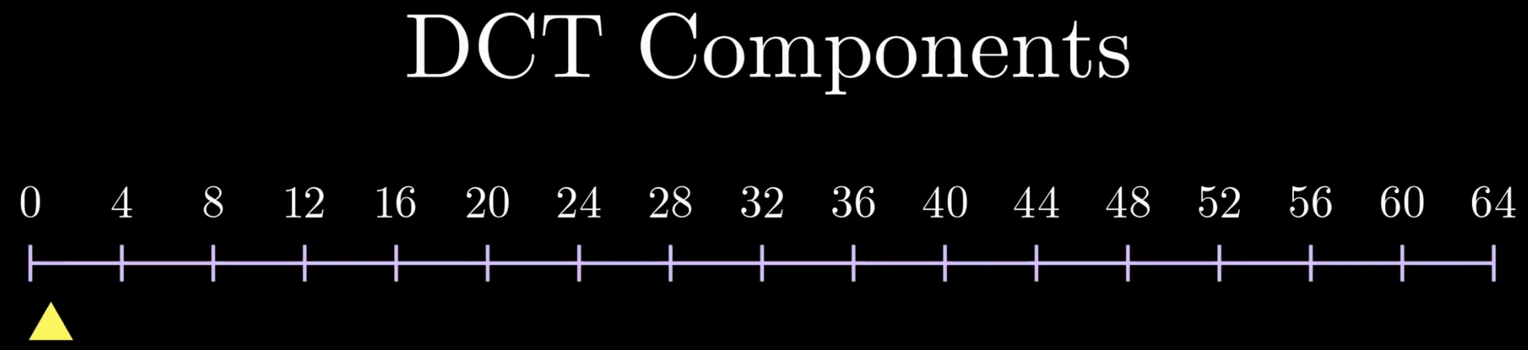
Transform Coding -> DCT

- Observation: For most of the “natural” image blocks, the DCT is sparse, and concentrated in the lower frequencies



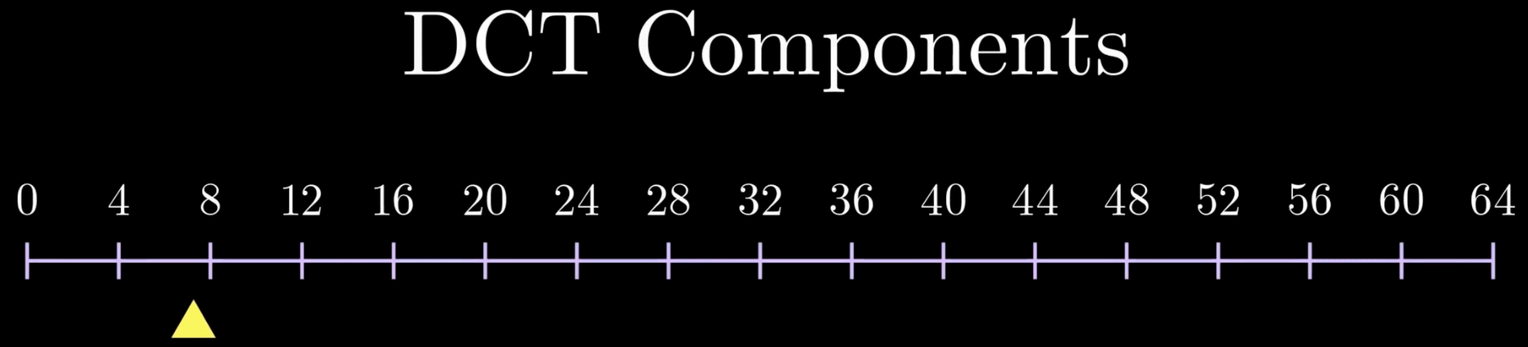
Transform Coding -> DCT

- Observation: For most of the “natural” image blocks, the DCT is sparse, and concentrated in the lower frequencies



Transform Coding -> DCT

- Observation: For most of the “natural” image blocks, the DCT is sparse, and concentrated in the lower frequencies



Transform Coding -> DCT

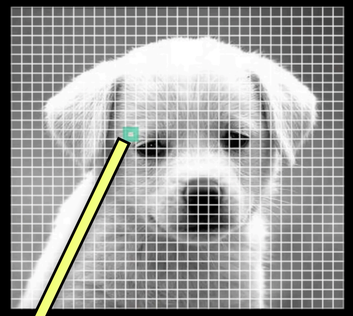
- Observation: For most of the “natural” image blocks, the DCT is sparse, and concentrated in the lower frequencies

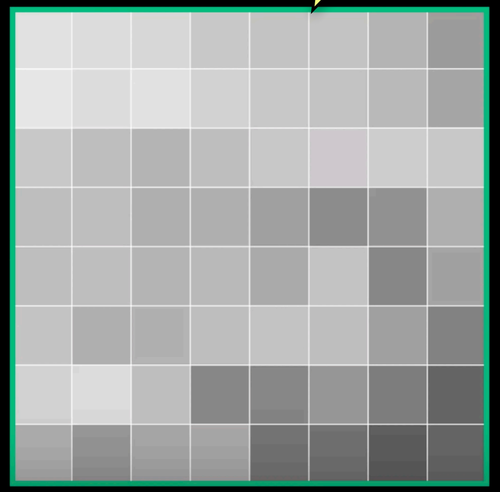
- Energy Compaction: Most of the high-frequency DCT coefficients have low magnitude, so can be ignored during lossy-compression (i.e. perform low-pass filtering)

This key observation forms the basis of JPEG image compression

JPEG Image Compression -> 2D-Block DCT

- **STEP-1:** Cut the image into blocks of size 8x8



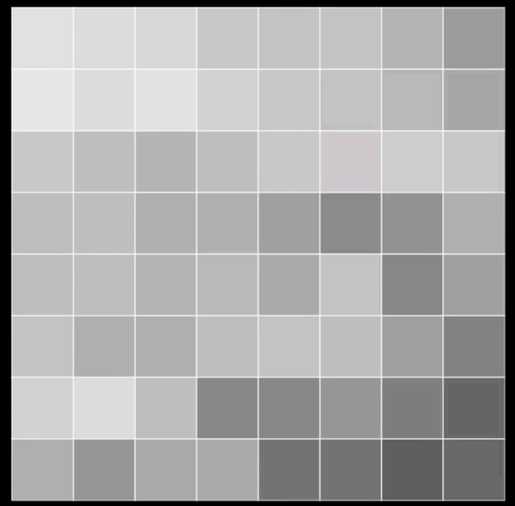


Input 8x8 block

221	218	211	196	189	189	174	149
230	220	223	205	193	188	177	160
197	183	175	185	193	198	200	193
183	183	168	170	151	129	139	166
183	185	175	181	163	187	126	154
192	169	170	183	188	185	153	120
205	215	186	126	123	142	118	93
166	142	161	161	107	105	85	94

Transform Coding -> DCT

- **STEP-1:** Cut the image into blocks of size 8x8
- **STEP-1.5:** subtract 128, to center the pixels
- **STEP-2:** 2D-DCT of each 8x8 block



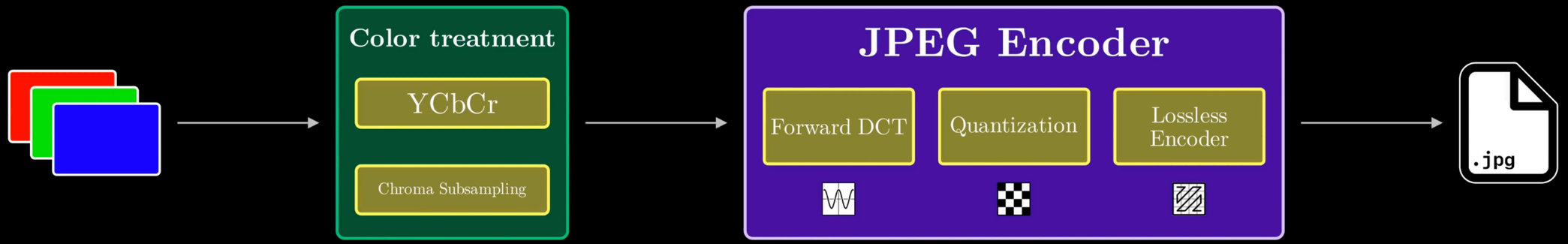
93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	−2	26
64	41	42	55	60	57	25	−8
77	87	58	−2	−5	14	−10	−35
38	14	33	33	−21	−23	−43	−34

Input 8x8 block
(zero centered)

338	145	−8	18	−7	4	16	−14
162	−41	3	2	3	−1	−13	9
−17	57	−2	−2	−20	16	2	10
41	19	−24	31	−19	−8	4	−1
−59	7	−2	−32	21	−1	6	−15
−19	12	32	0	−16	−9	−15	12
7	−55	−24	17	20	15	−4	0
15	−11	10	11	−18	−13	10	−10

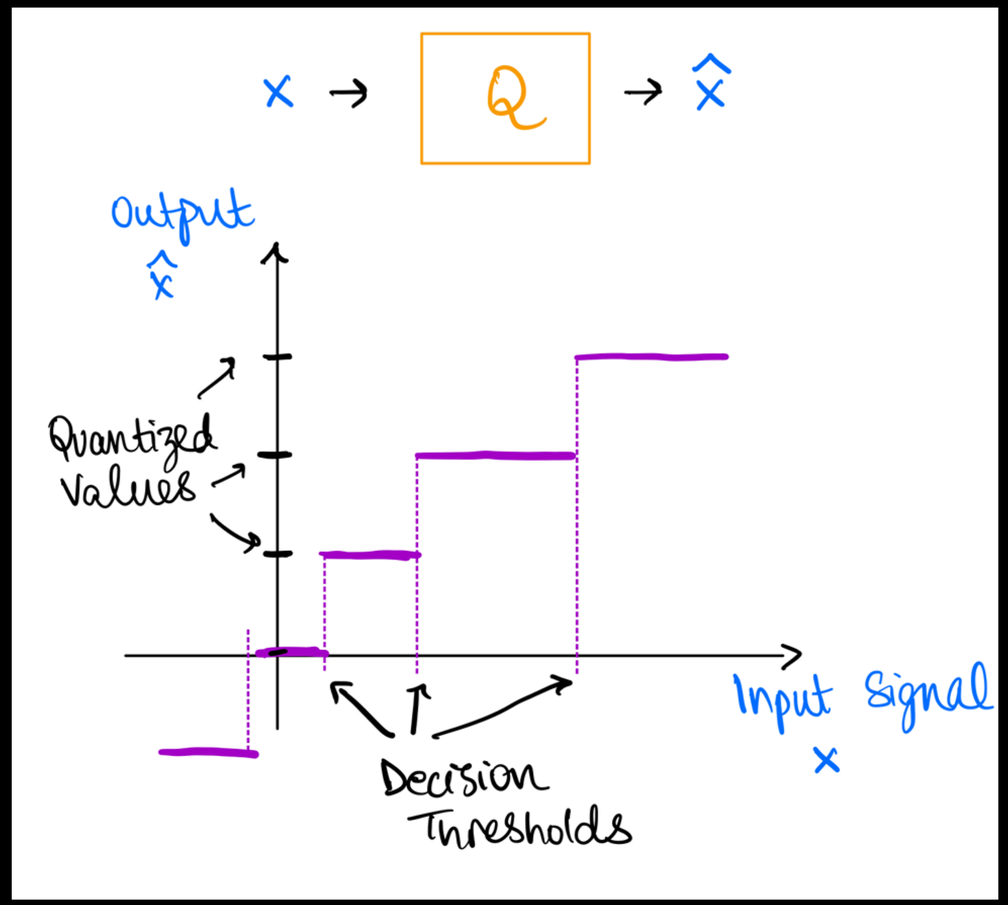
2D DCT

JPEG Image Compression



Apply lossy compression on DCT blocks

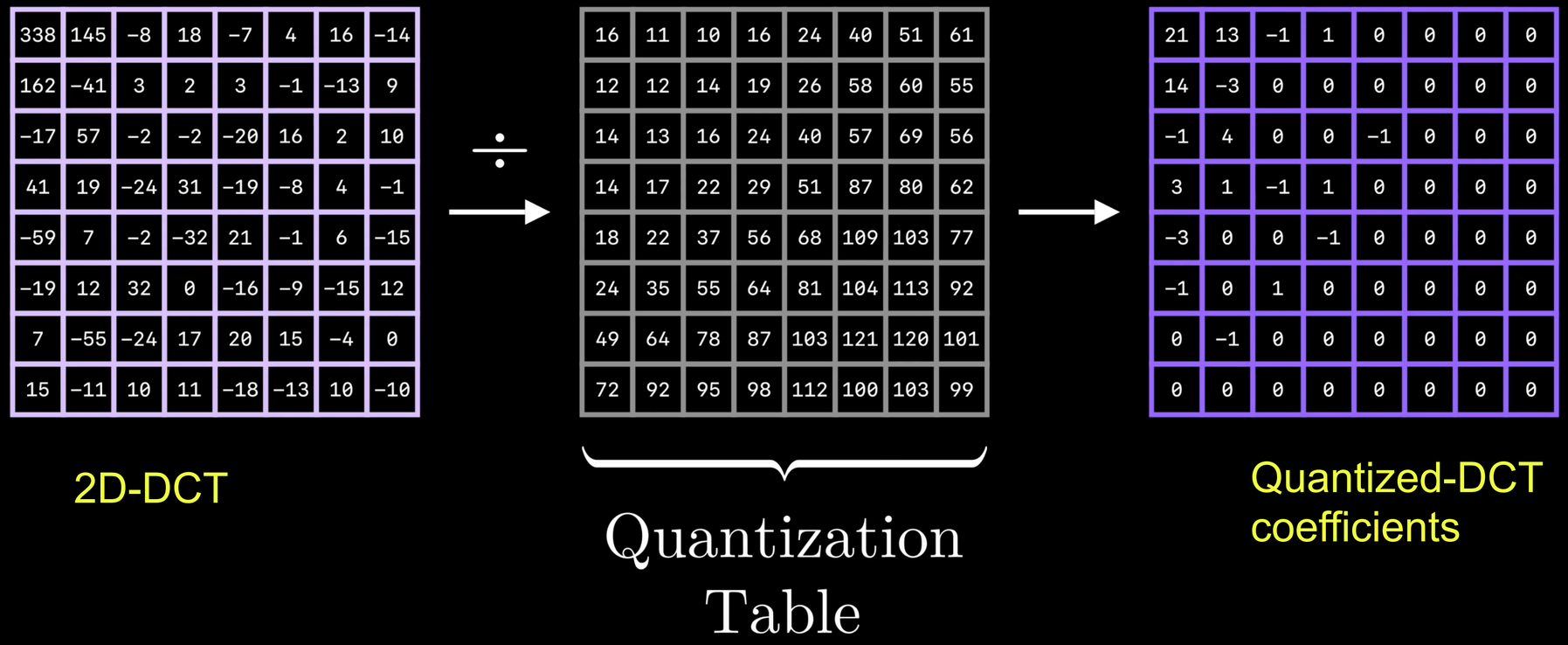
Uniform scalar quantization



Fewer quantization levels
=
Bigger quantization step size
=
Higher compression ratio
=
Higher distortion

JPEG Image Compression -> Quantization

- **STEP-1:** Cut the image into blocks of size 8x8
- **STEP-2:** 2D-DCT of each 8x8 block
- **STEP-3:** uniform scalar quantize DCT coefficients based on the quantization table.



Determines quantization step size (higher = more distortion)

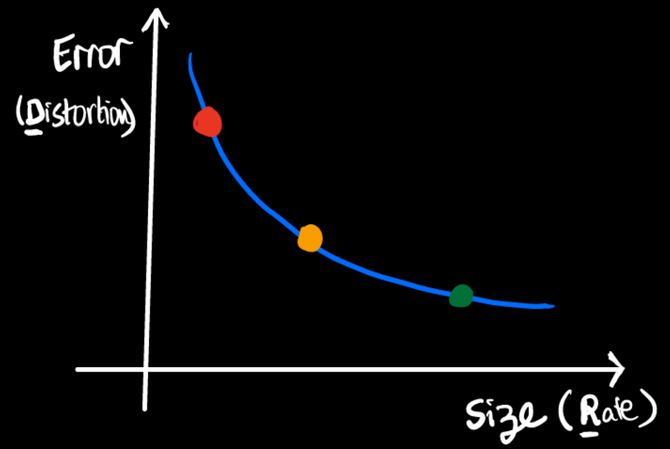
JPEG Image Compression -> Quantization

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Quality factor: 50

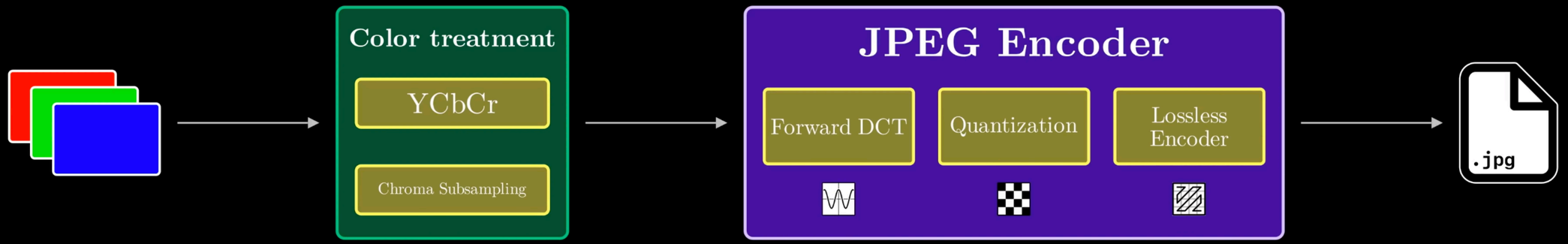
6	4	4	6	10	16	20	24
5	5	6	8	10	23	24	22
6	5	6	10	16	23	28	22
6	7	9	12	20	35	32	25
7	9	15	22	27	44	41	31
10	14	22	26	32	42	45	37
20	26	31	35	41	48	48	40
29	37	38	39	45	40	41	40

Quality factor: 80



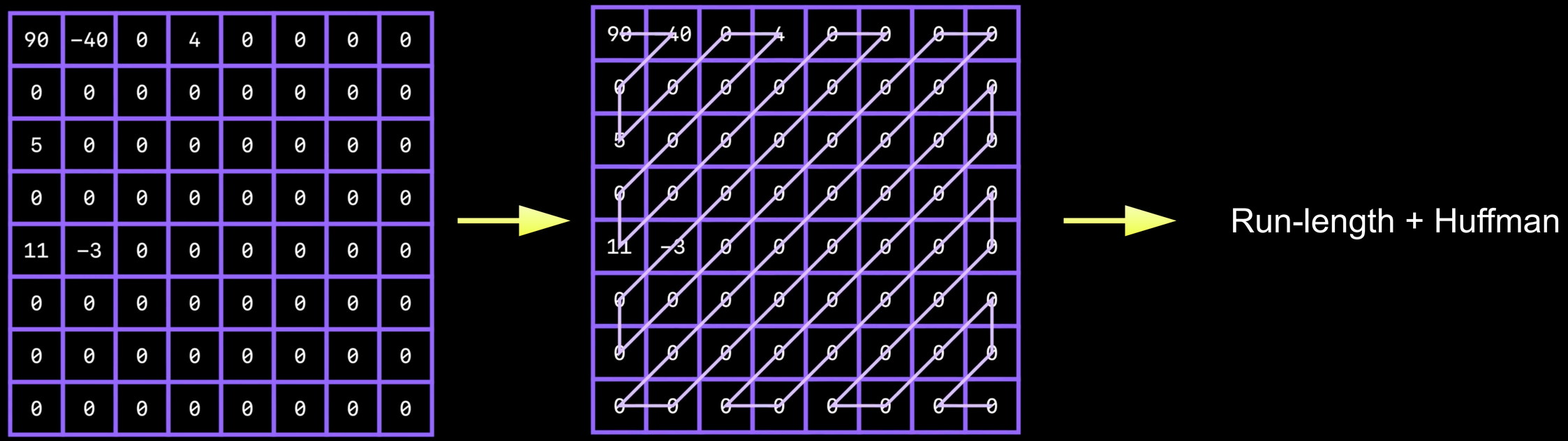
Different quantization tables
For different compression rates

JPEG Image Compression



Losslessly compress quantized DCT coefficients

JPEG Image Compression



Quantized, transformed Coefficients for one 8x8 block

{ 90 −40 0 5 0 0 4 0 0 0 11 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }

What are the issues with JPEG?

- Block size 8x8
- Blocks processed independently
- lossless coding can be improved (multiple possibilities with different compression/speed profiles)

Image Compression -> JPEG 137x

Uncompressed -> 1.1MB
JPEG -> 8KB (~137x!)

Image from Kodak dataset

Image Compression -> BPG

Uncompressed -> 1.1MB
BPG -> 8KB (~137x!)

Exploits correlation between blocks: Predictive coding

Use larger transform blocks: Better energy compaction, better compression

CABAC instead of Huffman: Adaptive Arithmetic coding instead of Huffman.

Image from Kodak dataset

HiFiC -> ML-based image compression

Image from Kodak dataset

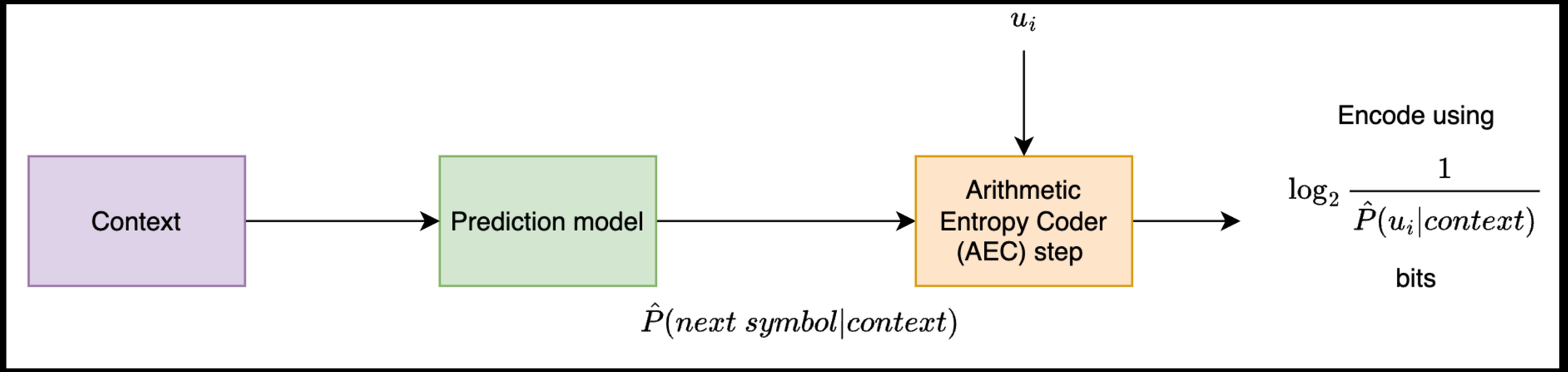
Uncompressed -> 1.1MB
HiFiC -> **8KB (~137x!)**

More on this soon!

Before we move on...

A quick primer on lossless
compression with arithmetic coding

Lossless Compression with AEC



Higher predicted probability => fewer bits used to encode

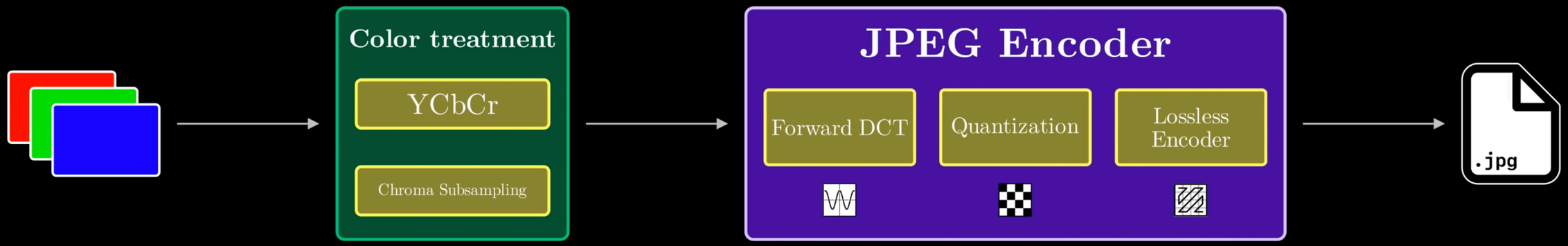
Lossless Compression with AEC

- Given a probability model for data, AEC can achieve lossless compression
  - Optimal compression (“entropy”) when model exactly matches true data distribution, otherwise overhead (“relative entropy”)
  - For better compression: fit better model, or modify data to fit model better!

- Good predictor => good compression
  - Prediction loss function (cross-entropy loss) identical to compressed size obtained with arithmetic coding!

Part II

Recap -> JPEG


Color treatment
YCbCr
Chroma Subsampling
JPEG Encoder
Forward DCT
Quantization
Lossless Encoder
.jpg

What next?

Beyond Linear transform: JPEG/JPEG2000/BPG all use variants of DCT, DWT etc. can we obtain better performance with non-linear transforms

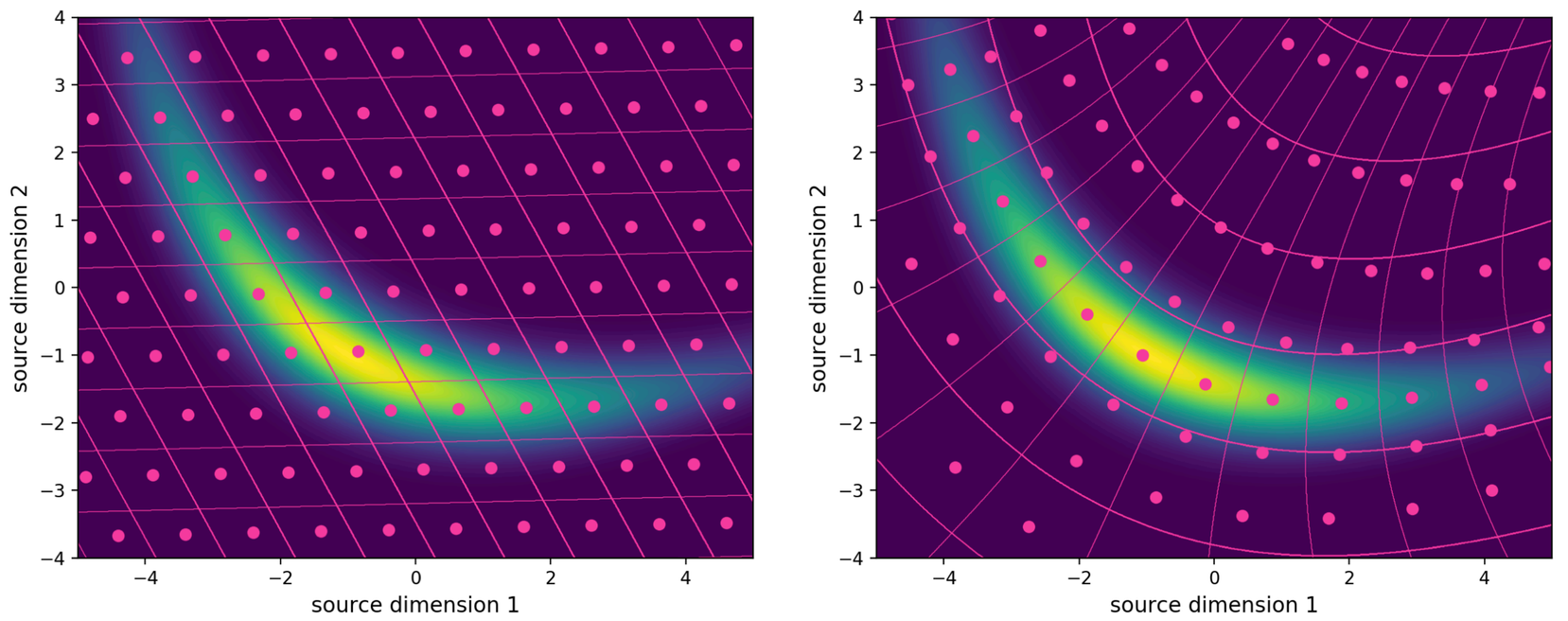


Fig. 1. Linear transform code (left), and nonlinear transform code (right) of a banana-shaped source distribution, both obtained by empirically minimizing the rate–distortion Lagrangian (eq. (13)). Lines represent quantization bin boundaries, while dots indicate code vectors. While LTC is limited to lattice quantization, NTC can more closely adapt to the source, leading to better compression performance (RD results in fig. 3; details in section III).

What next?

Beyond Linear transform: JPEG/JPEG2000/BPG all use variants of DCT, DWT etc. can we obtain better performance with non-linear transforms

End-to-End RD Optimization: JPEG uses very smart albeit heuristics for R-D optimization, e.g. rate needs to be shared between different channels. Can we make R-D decisions end-to-end?

Recent Learned Image/Video Codec Works

- **Learned Image Compression:**
  [Toderici, CVPR15], [Theis, ICLR17], [Agustsson, NIPS17], [Baig, NIPS17],
  [Balle, ICLR17], [Rippel, ICML17], [Balle, ICLR18], [Johnston, CVPR18],
  [Mentzer, CVPR18], [Choi, ICLR19], [Balle, ICLR19], [Lee, ICLR19], [Mentzer, CVPR19]
  [Lu, 2021], [Ma et al, 2021], [Yang et al, NIPS2021], [Mentzer et al. NIPS2021] . . .

- **Learned Video Compression**
  [Wu. et al, ECCV18], [Lu et al, CVPR19], [Cheng et al, CVPR19]
  [Rippel et al ICCV19], [Hu et al, 2020], [Agustsson et al. 2020], [Golinski et al. 2020]
  [Habiban et al.2019], [Lu et al. 2020], [Liu el.al.2020], [DVC, Lu et al. 2019] . . .

- **The CLIC Challenge**
  “Challenge on Learned Image Compression”
  -> ongoing image compression contest at CVPR

Lots of interesting works!

Our Goal: Understand the Key concepts

The Image Compression Problem

I

Encoder

$bits$

Decoder

$\hat{I}$

Target Image

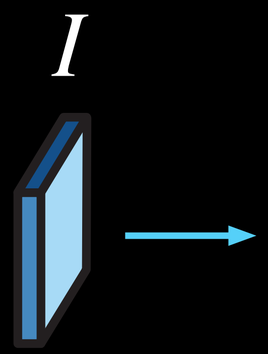
Target reconst

Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Rate

Distortion

Traditional Image Codecs

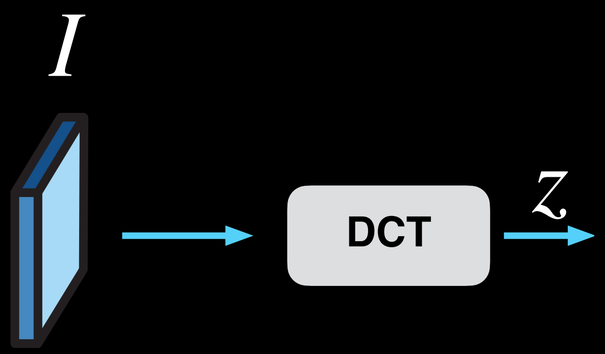


Target Image

Encoding proceeds in 3 steps:

CAUTION: Simplified framework

Traditional Image Codecs



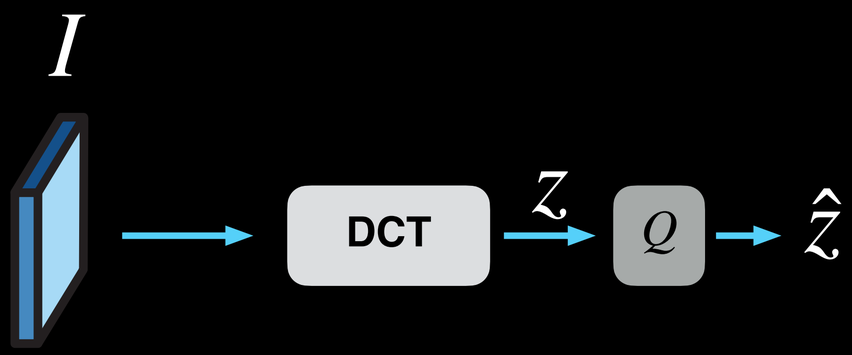
Target Image

Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels

CAUTION: Simplified framework

Traditional Image Codecs



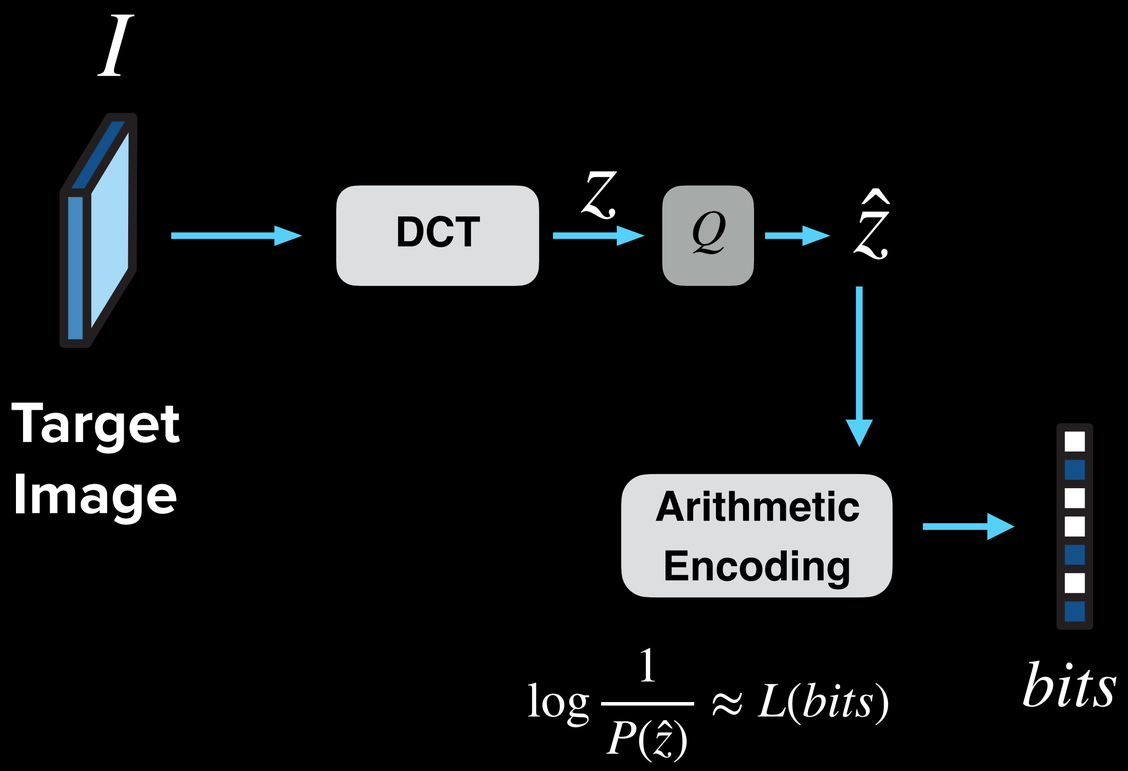
Target Image

Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels
- **Quantize ->** Loss of precision
  $Q([2.3,3.7]) = [2,4]$

CAUTION: Simplified framework

Traditional Image Codecs



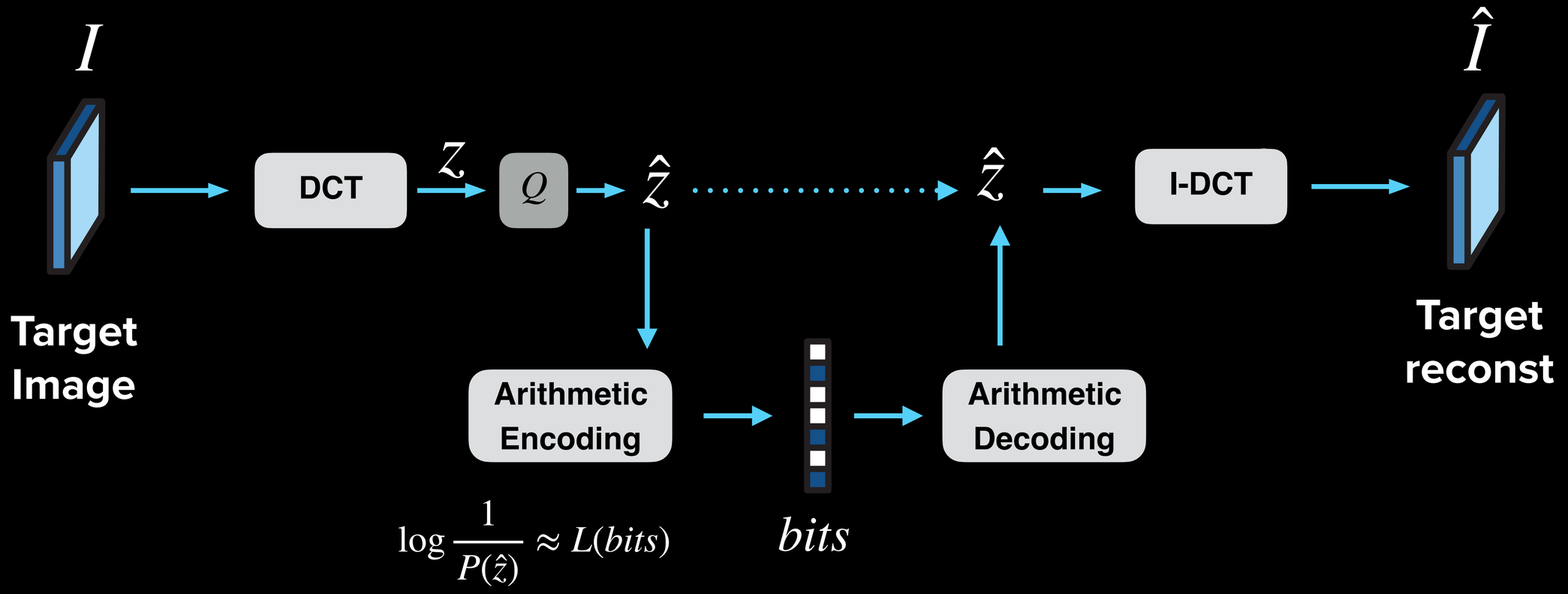
Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels
- **Quantize ->** Loss of precision
  $Q([2.3,3.7]) = [2,4]$
- **Arithmetic/Huffman ->** Lossless Compression
  In the simplest form, uses a discrete distribution $P(\hat{z})$ to encode $\hat{z}$ into a bitstream of length

$$L(bits) \approx \log \frac{1}{P(\hat{z})}$$

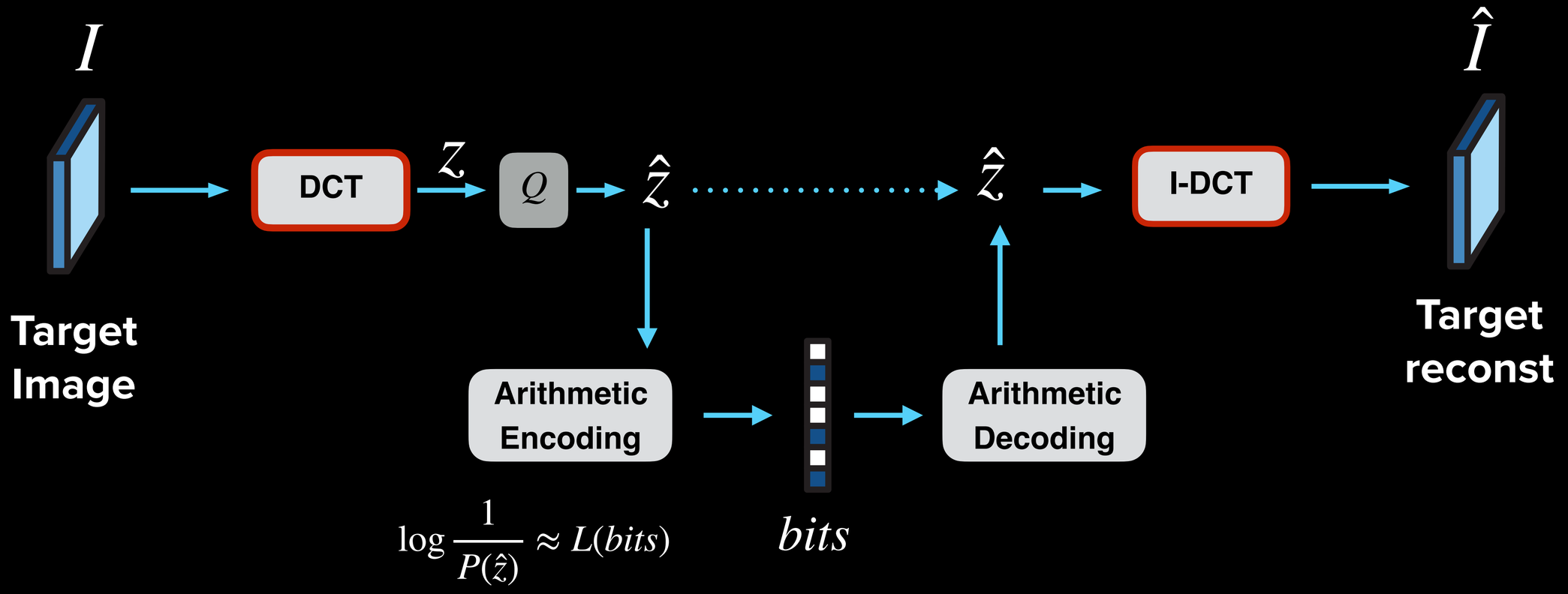
CAUTION: Simplified framework

Traditional Image Codecs



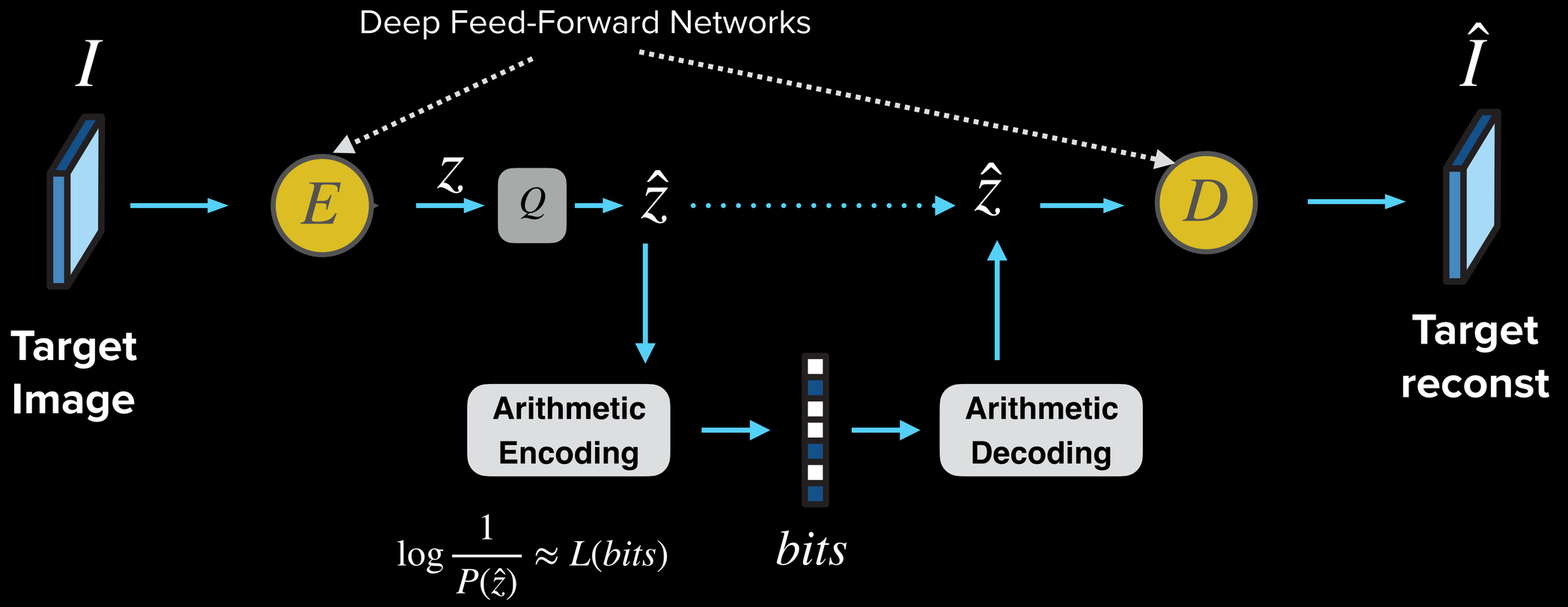
Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Traditional Image Codecs

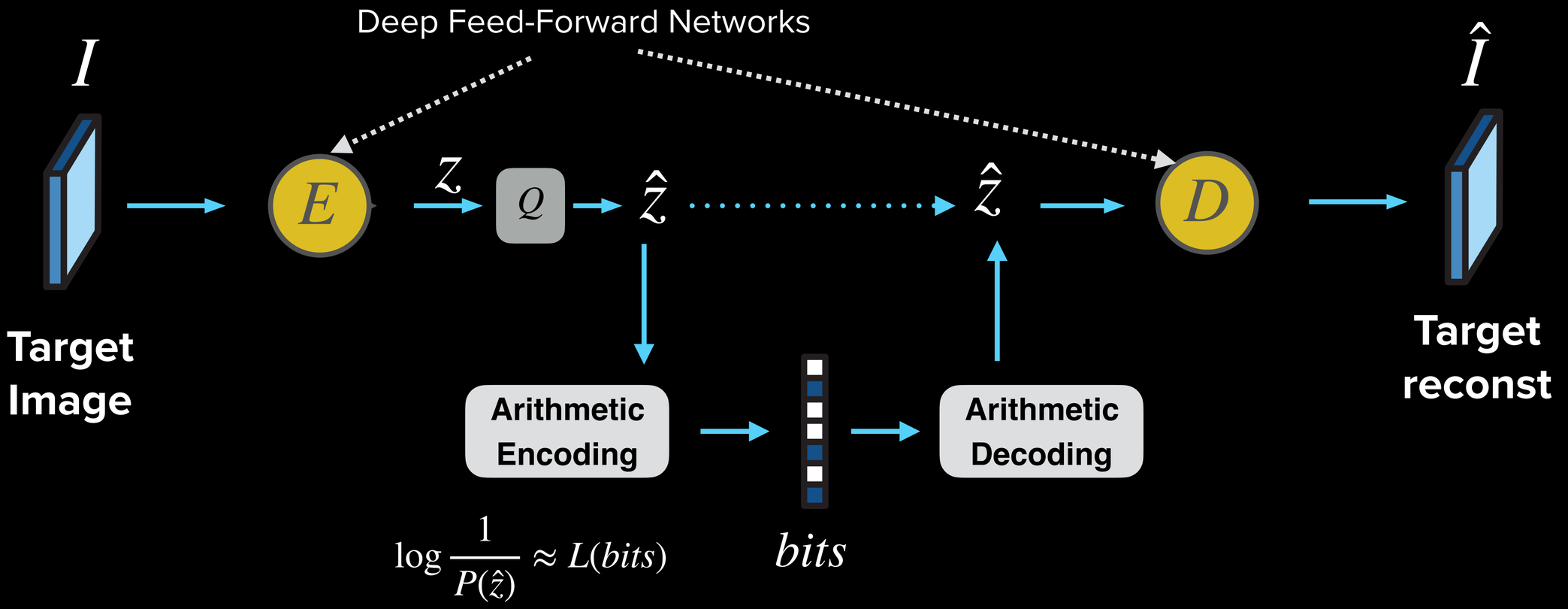


Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Learned Image Codecs

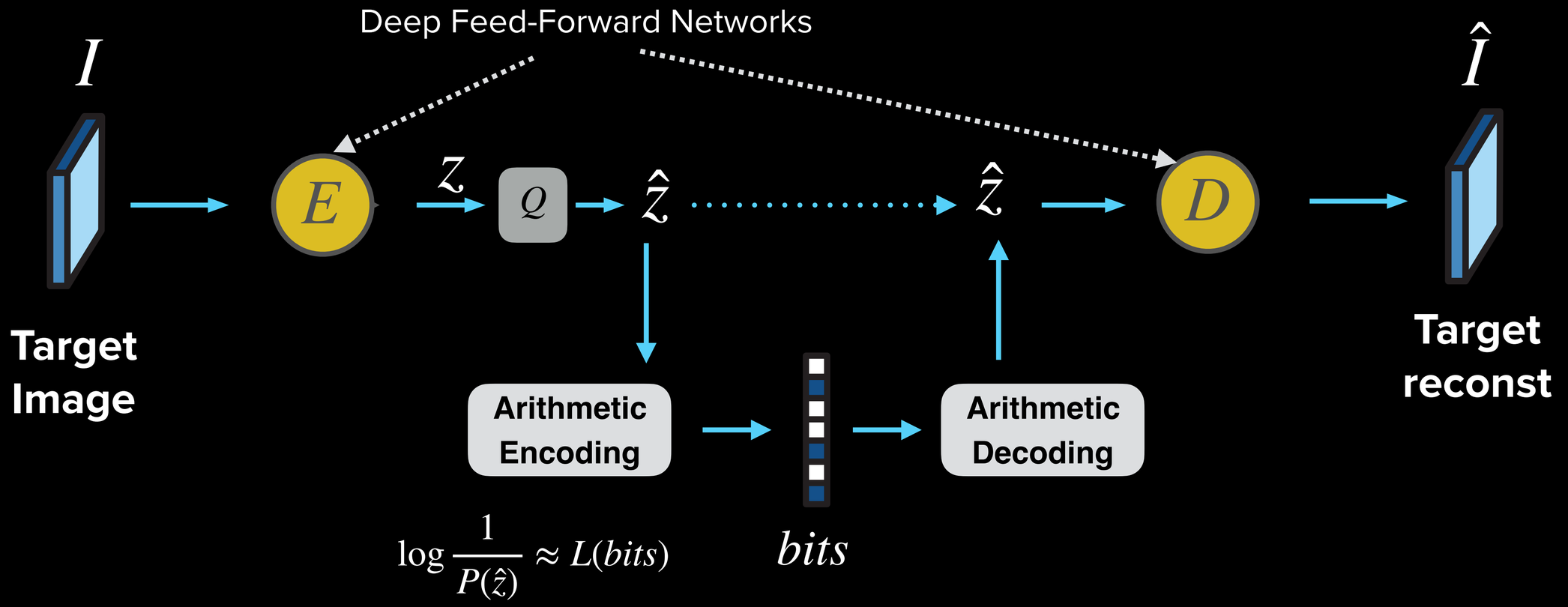


Learned Image Codecs



Question: How do we train the parameters?

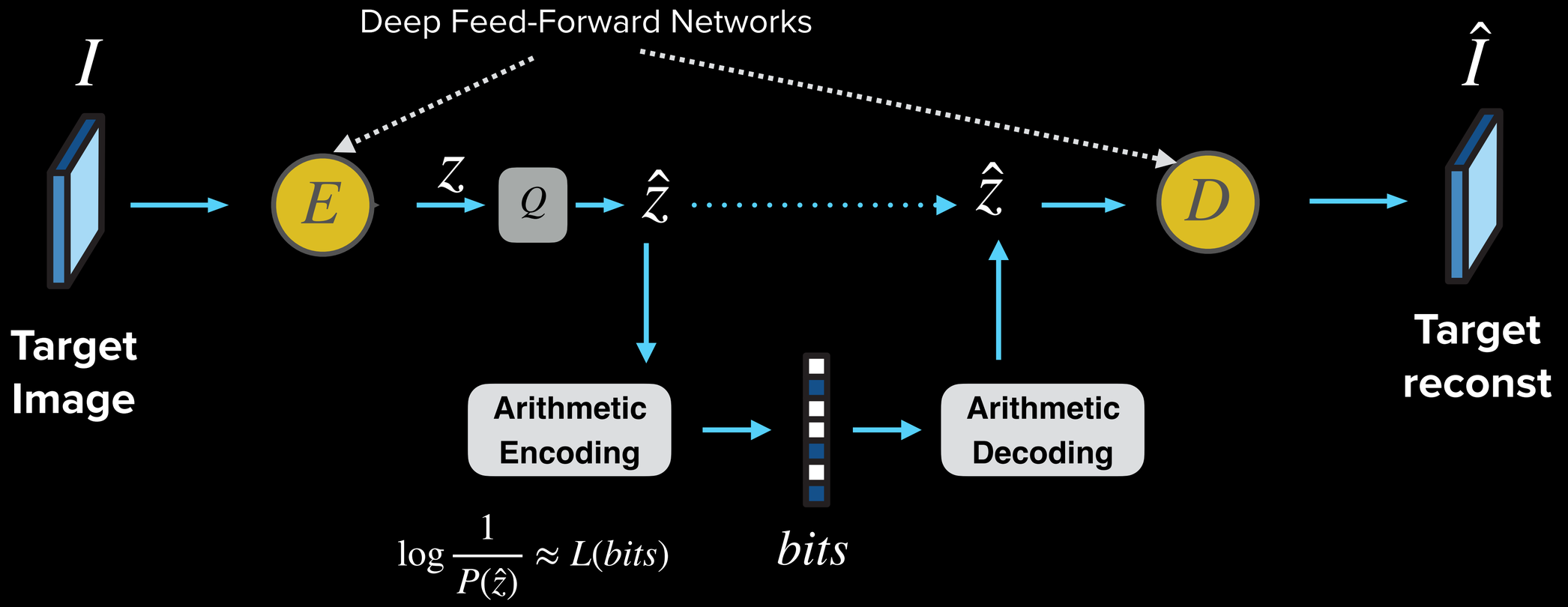
Learned Image Codecs



Loss Function = $L(bits) + \lambda d(I, \hat{I})$

Rate

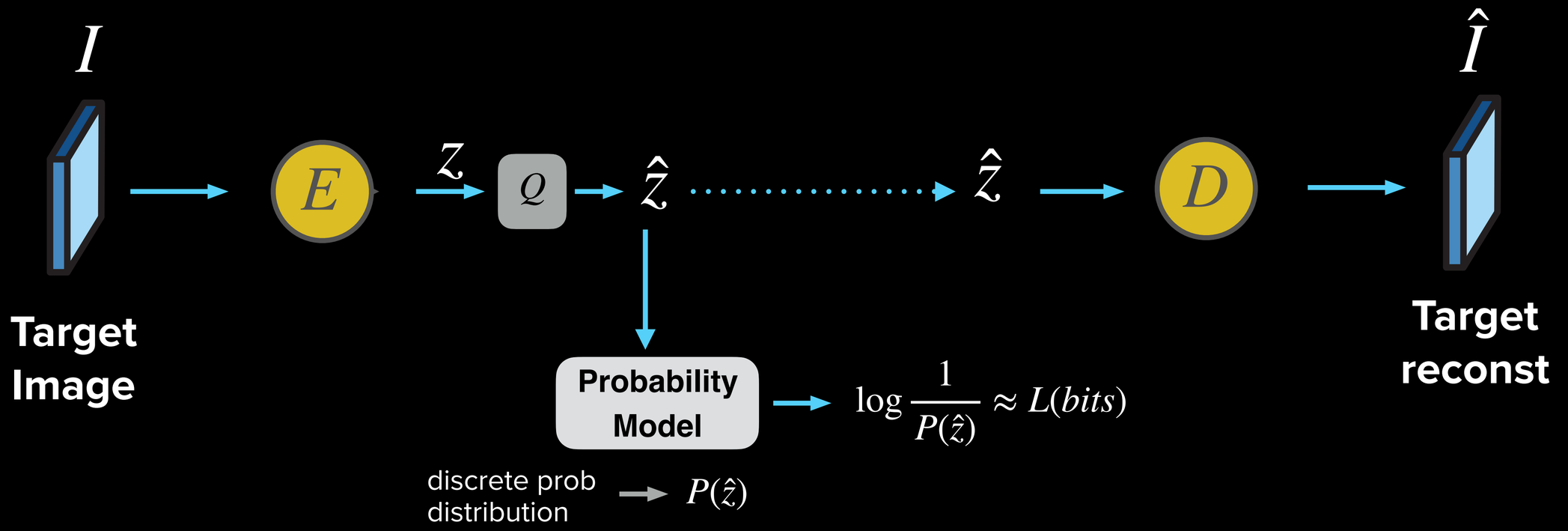
Learned Image Codecs



Loss Function = $L(bits) + \lambda d(I, \hat{I}) \approx \log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

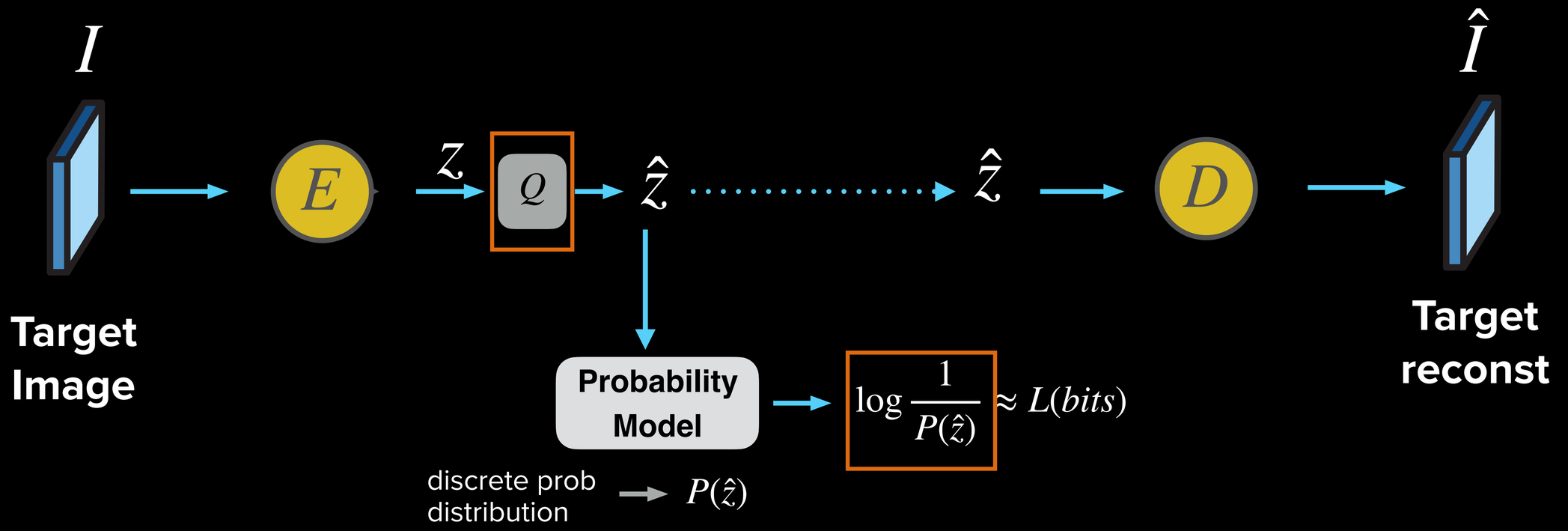
Rate

Learned Image Codecs



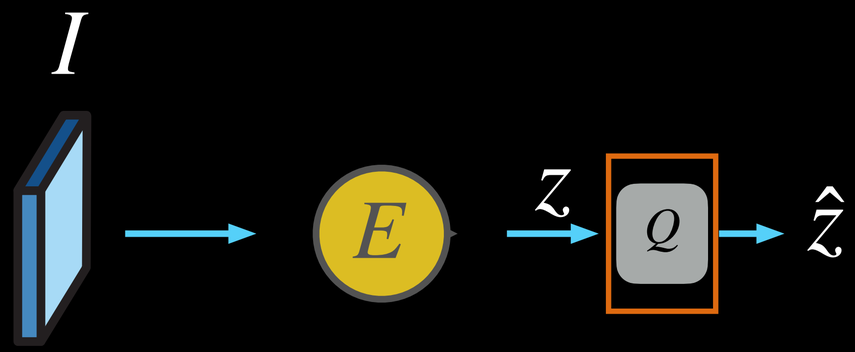
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

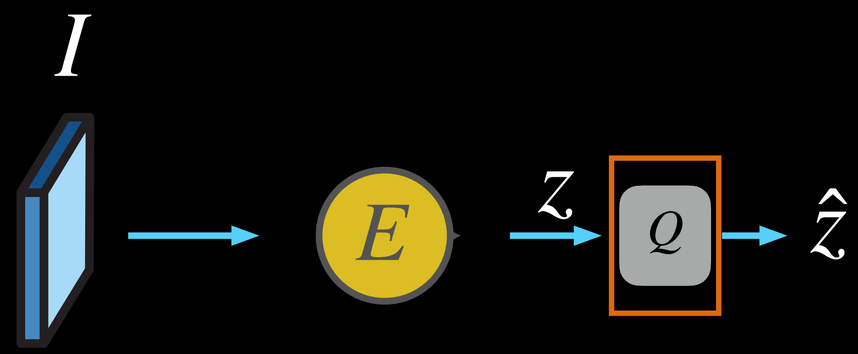


Target Image

- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



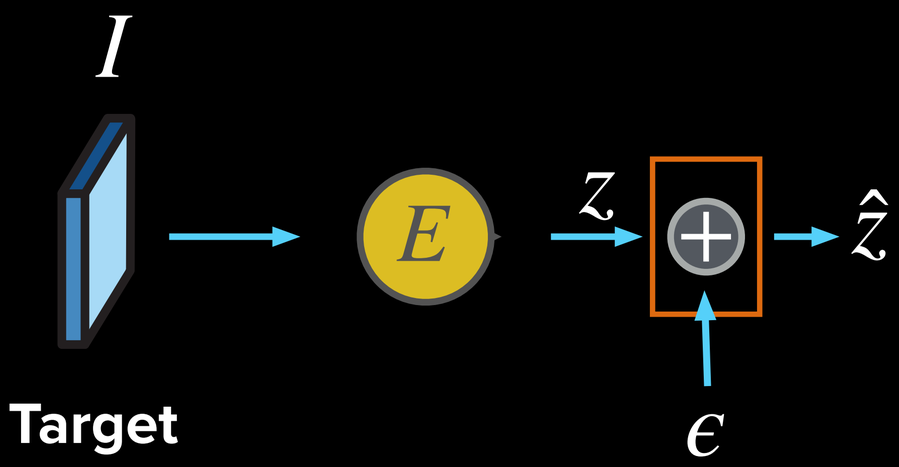
Target Image

- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$
- **Workaround-1:** model the quantizer as adding noise during training

  $\hat{z} = z + \epsilon$, where $\epsilon \sim U(-0.5,0.5)$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

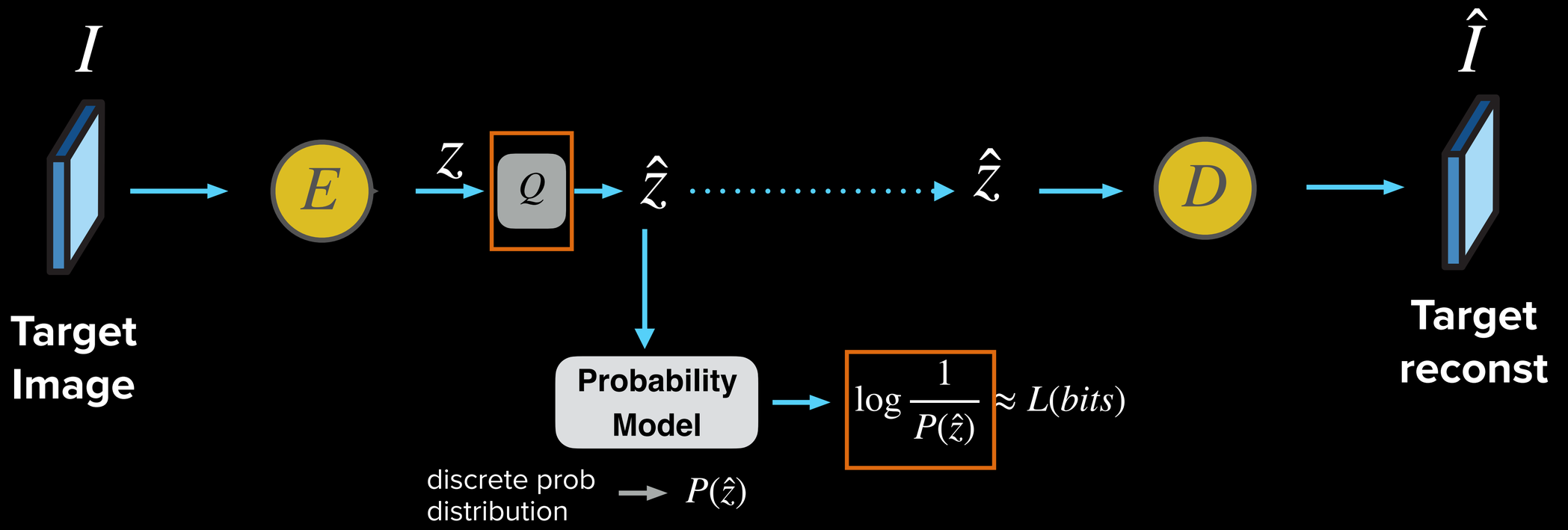


- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$
- **Workaround-1:** model the quantizer as adding noise during training

  $\hat{z} = z + \epsilon$, where $\epsilon \sim U(-0.5,0.5)$

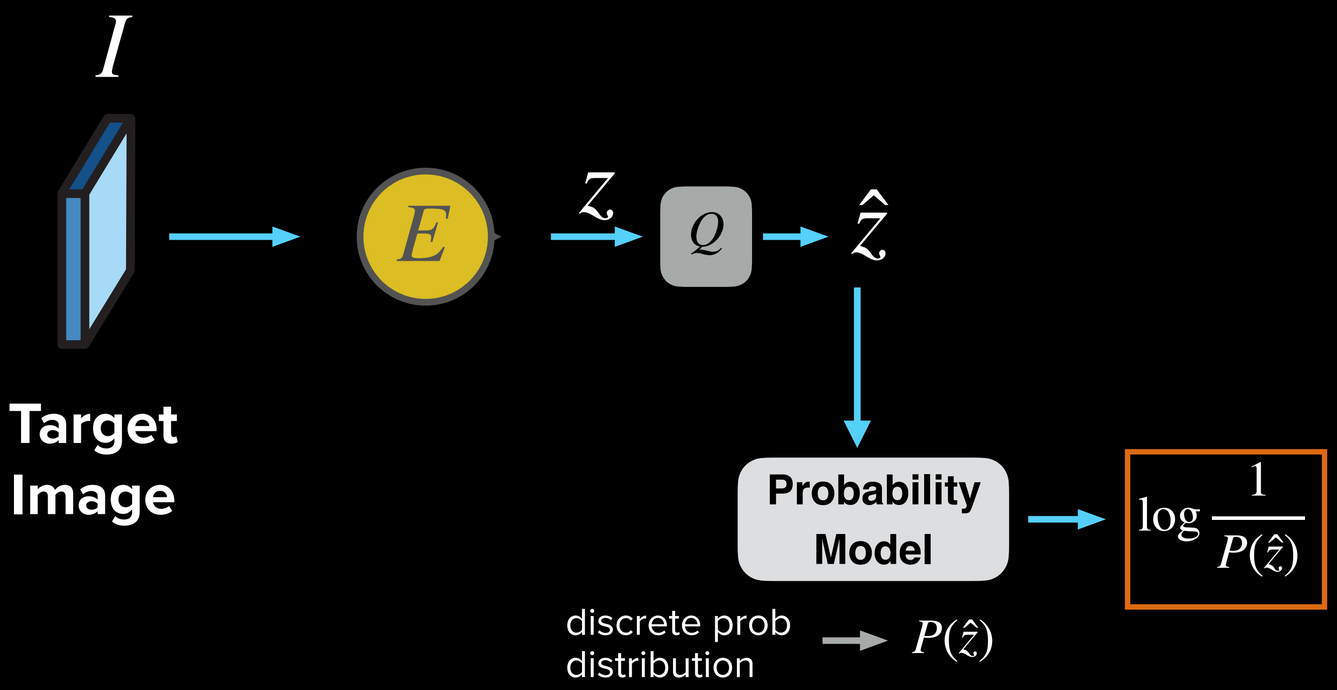
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

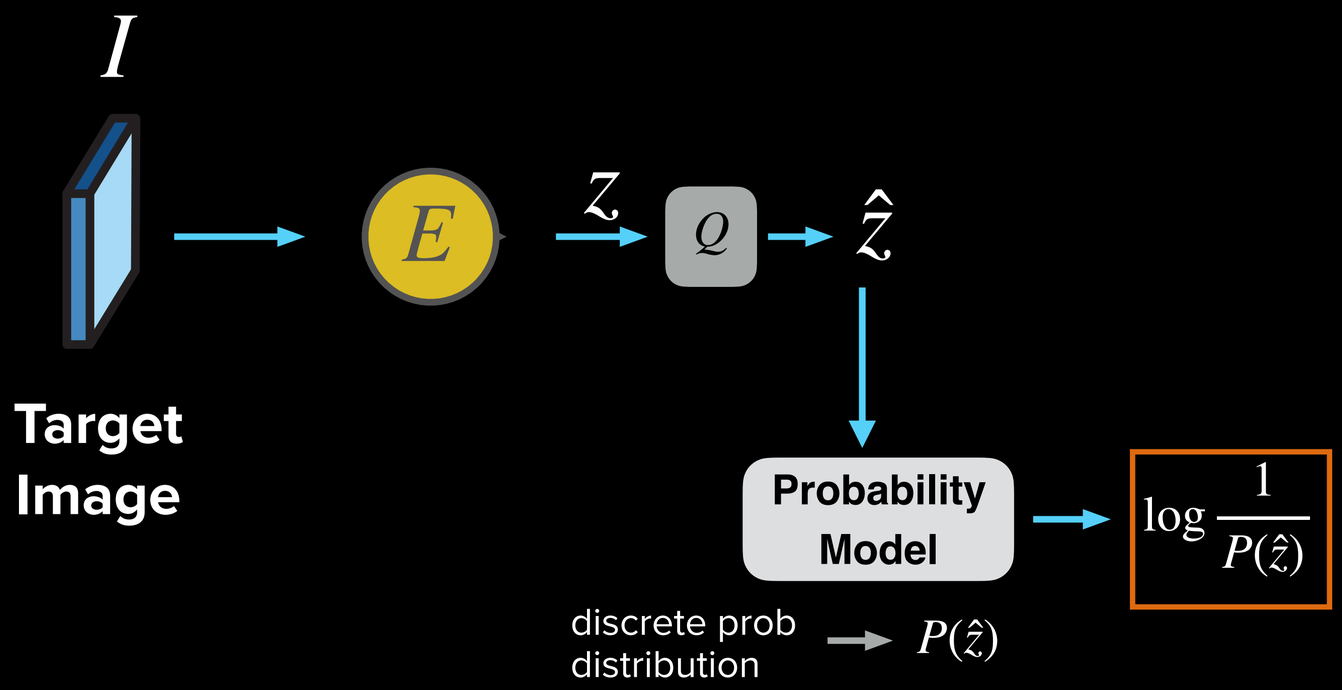
Learned Image Codecs



- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

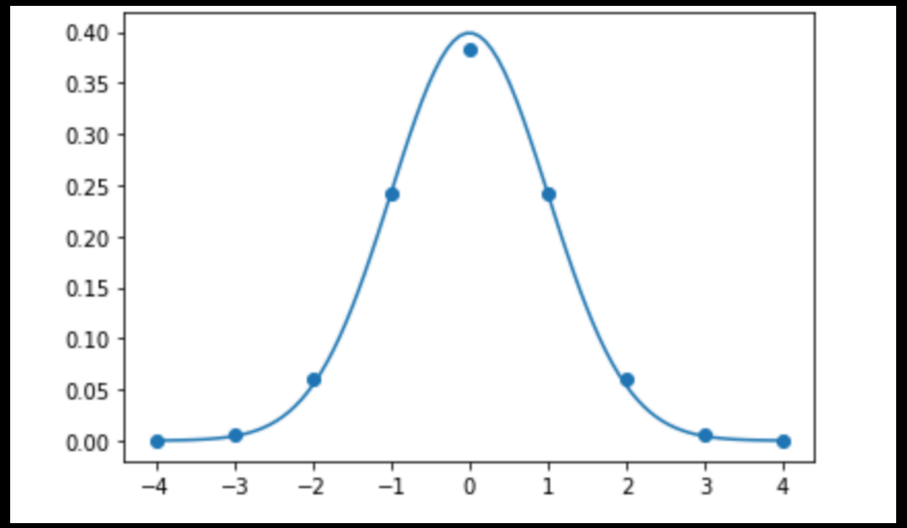


- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

$$P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

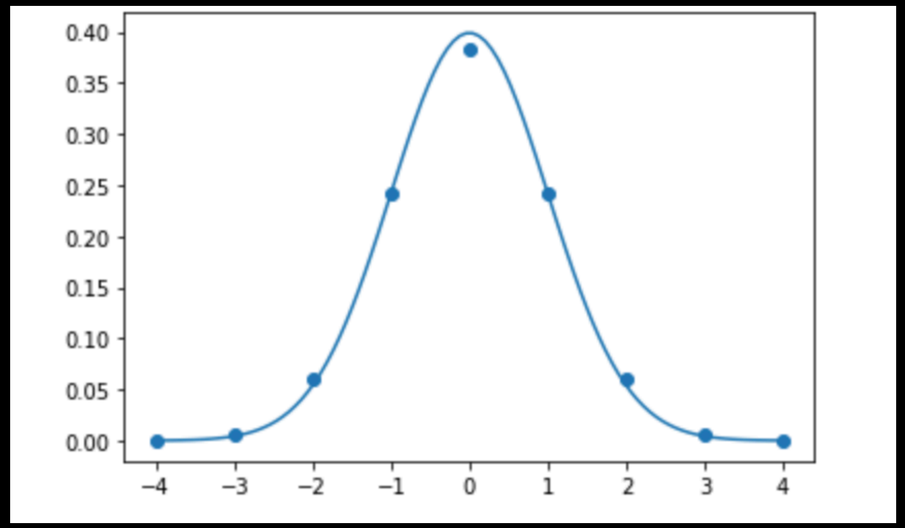


$$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$$

- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

$$P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$$

Learned Image Codecs



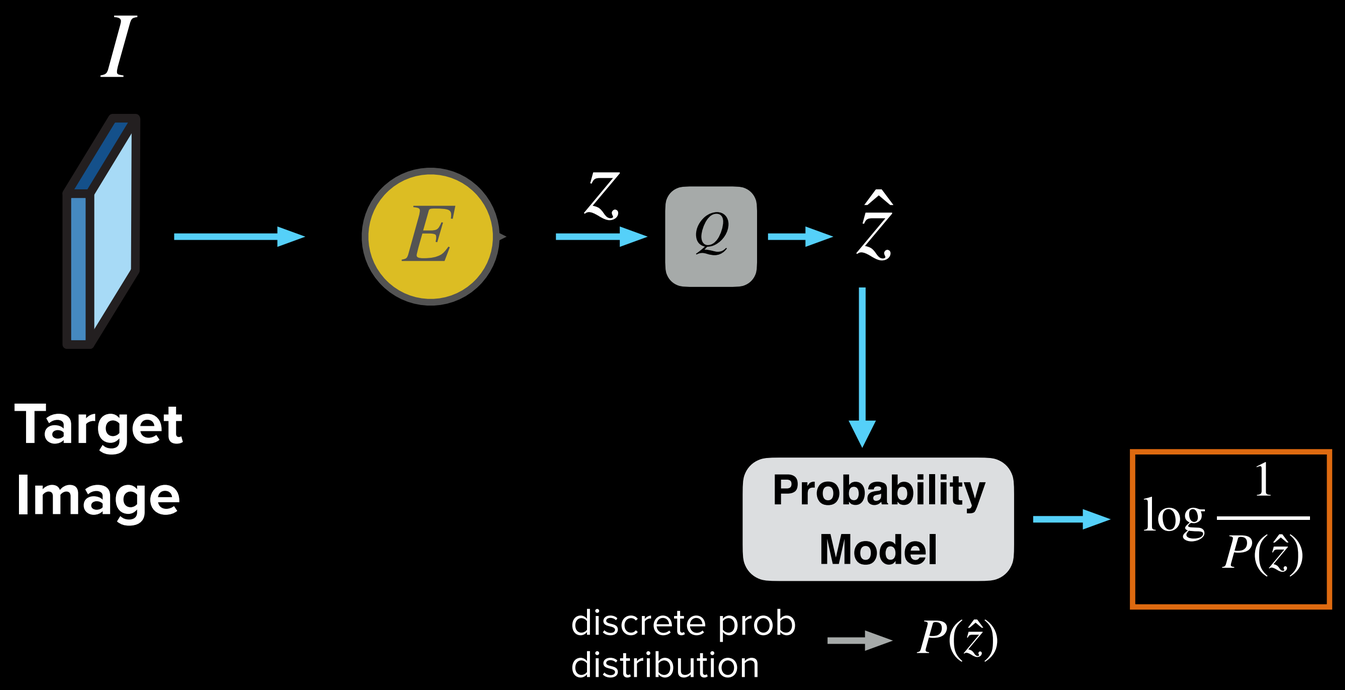
$$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$$

- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$
- Gradient is now well defined!

  $\frac{\partial P(\hat{z})}{\partial \hat{z}} = PDF(\hat{z} + 0.5) - PDF(\hat{z} - 0.5)$

Learned Image Codecs



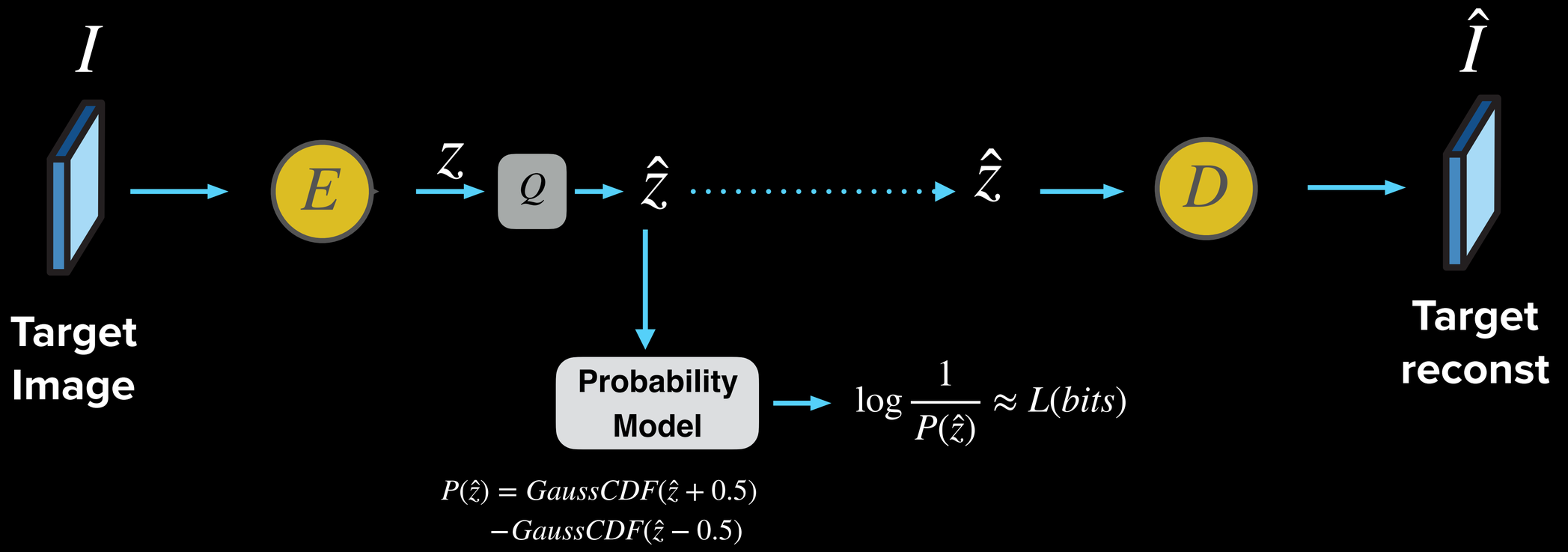
- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$
- Gradient is now well defined!

  $\frac{\partial P(\hat{z})}{\partial \hat{z}} = PDF(\hat{z} + 0.5) - PDF(\hat{z} - 0.5)$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

End-to-End Learned Image Codec



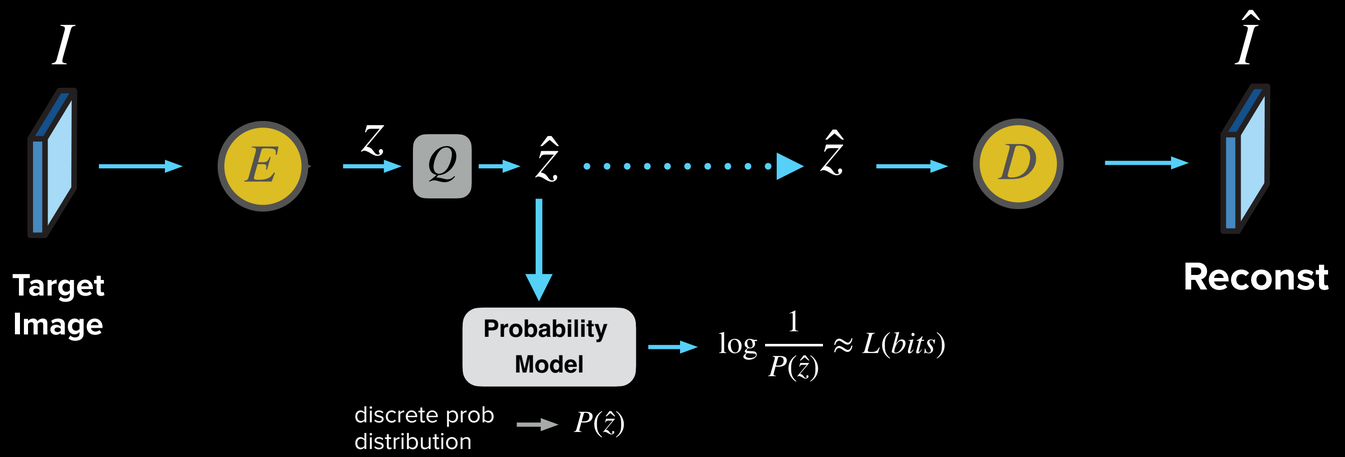
Possible to train end-to-end!

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Example -> MNIST

https://colab.research.google.com/drive/15QZ6PgQV65IBr2Qr55j0DqV_GVmdI5U0?usp=sharing

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

No hand-tuning parameters

Can “learn” the parameters

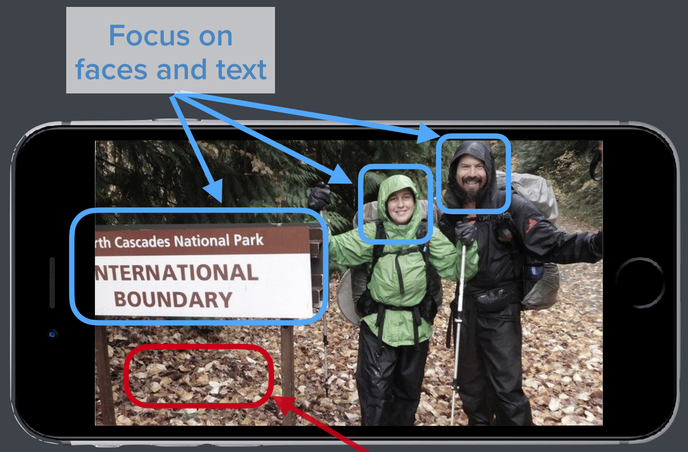
Possible to train end-to-end!

ML Offers Adaptivity Not Possible With Traditional Codecs

Domain-aware

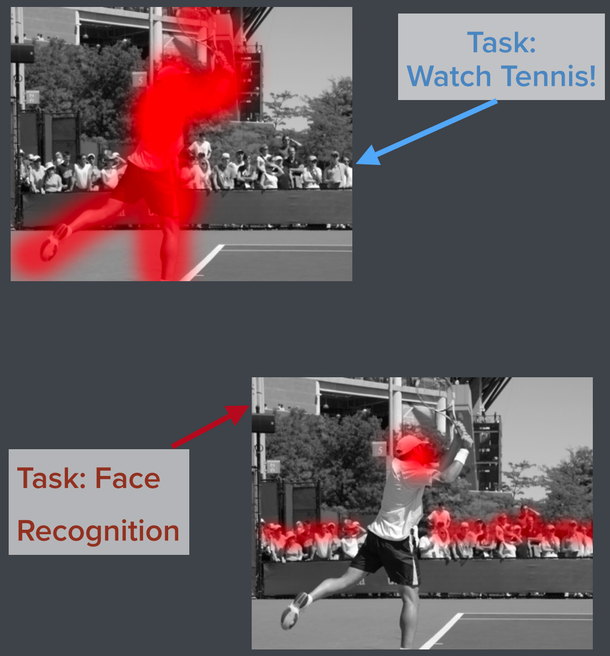
Custom codec for each domain

Content-aware



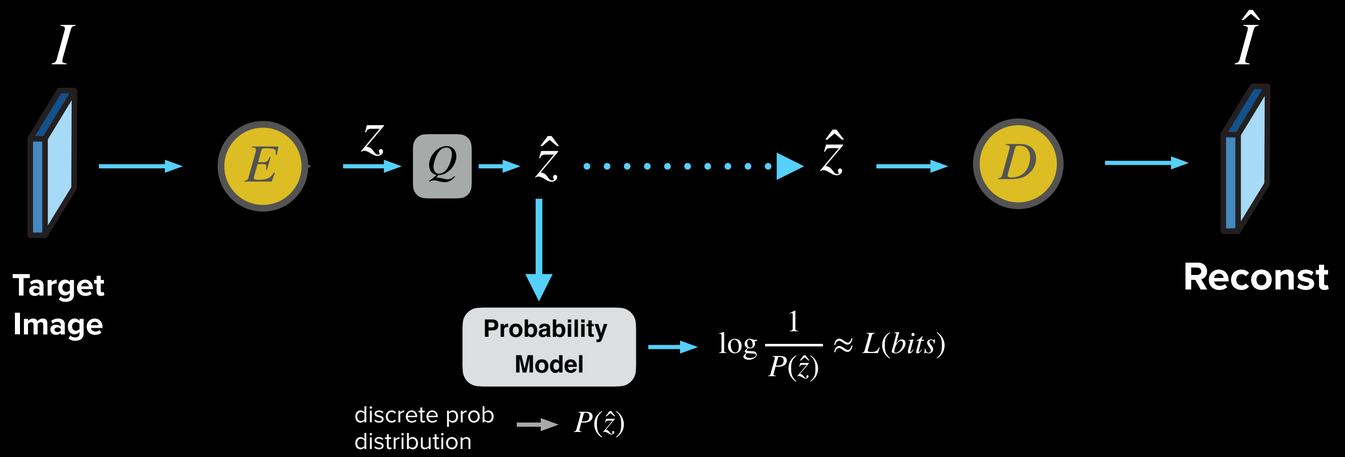
Focus on areas of high importance

Task-aware



Computation on compressed representation

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

No hand-tuning parameters

Can “learn” the parameters

Better distortion-Model separation

Easy to tune model to different distortion metrics

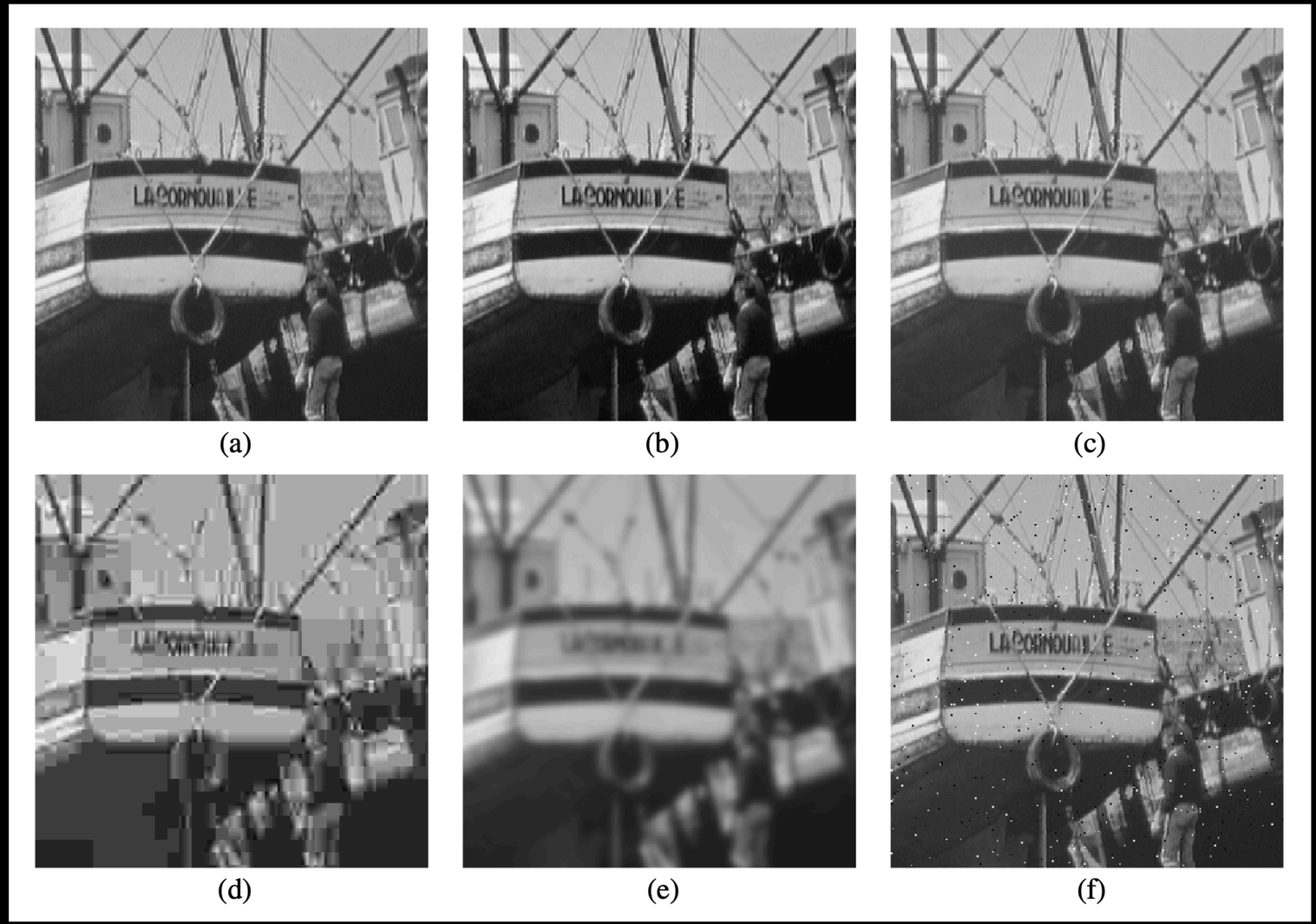
Possible to train end-to-end!

Given source image (a) which of the following images do you prefer visually?

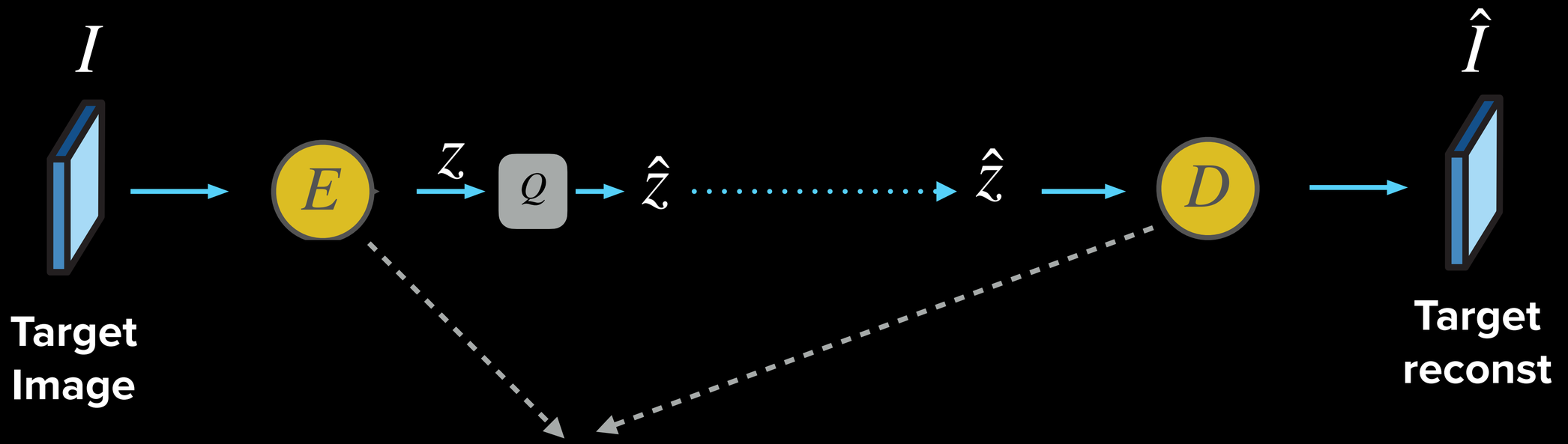
(b), (c), (d), (e), (f)

Given source image (a) which of the following images does a compressor with MSE distortion prefer?

(b), (c), (d), (e), (f)

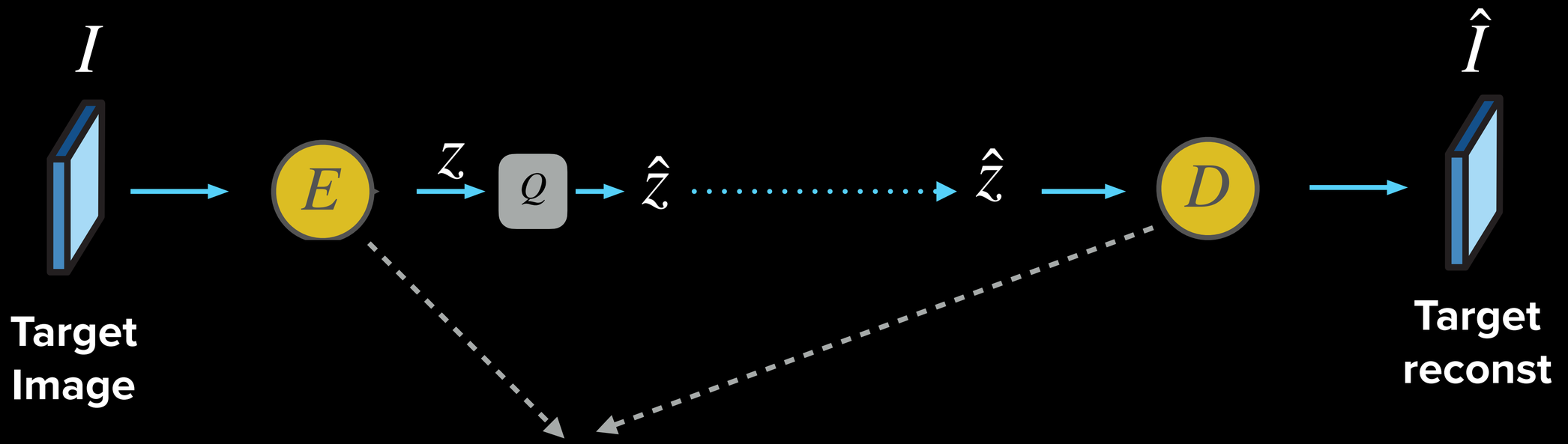


Design Decisions: (i) Backbones



- **Question:** How can the same model work for any image sizes?

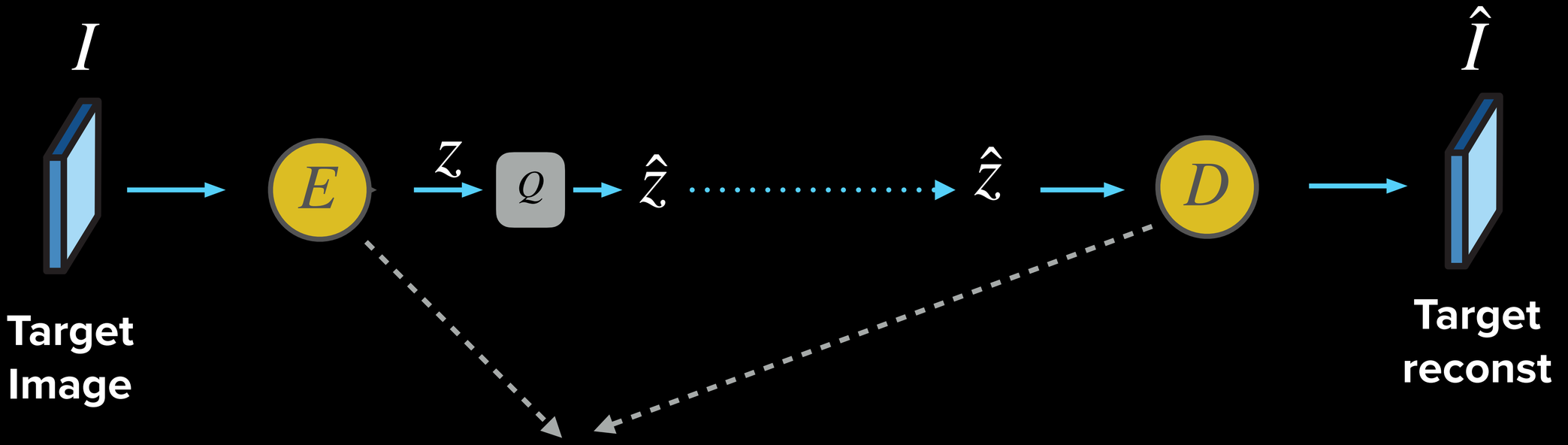
Design Decisions: (i) Backbones



- **Question:** How can the same model work for any image sizes?

  - Only use Conv, Deconv ... (no Fully Connected Layers)

Design Decisions: (i) Backbones

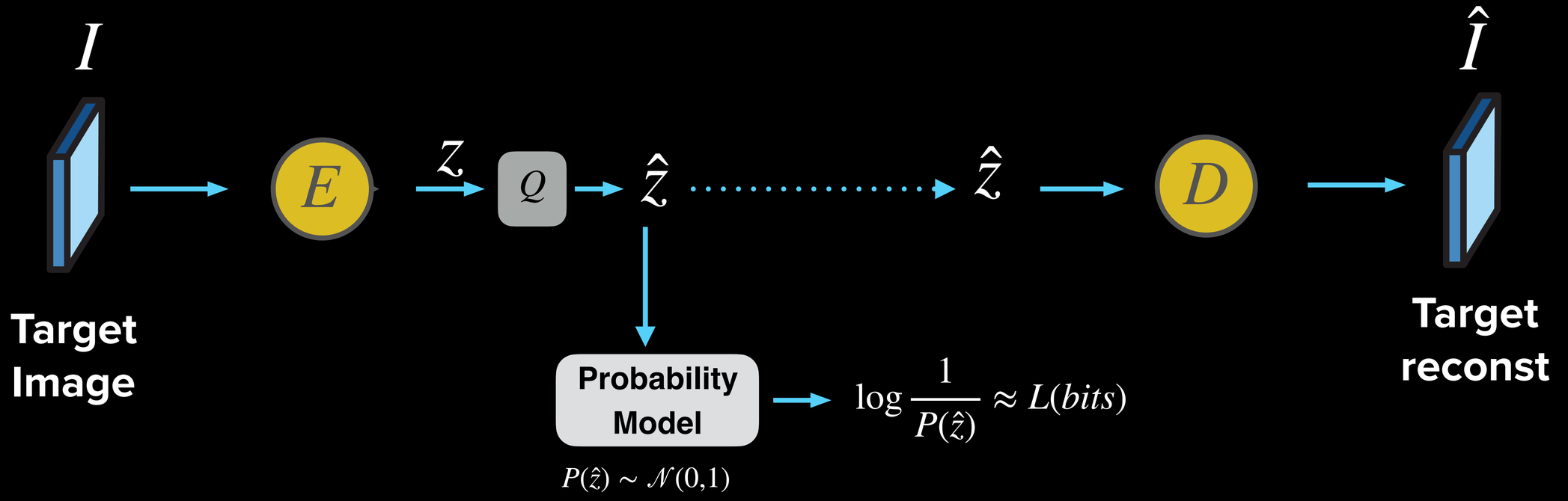


- **Other Improvements:**

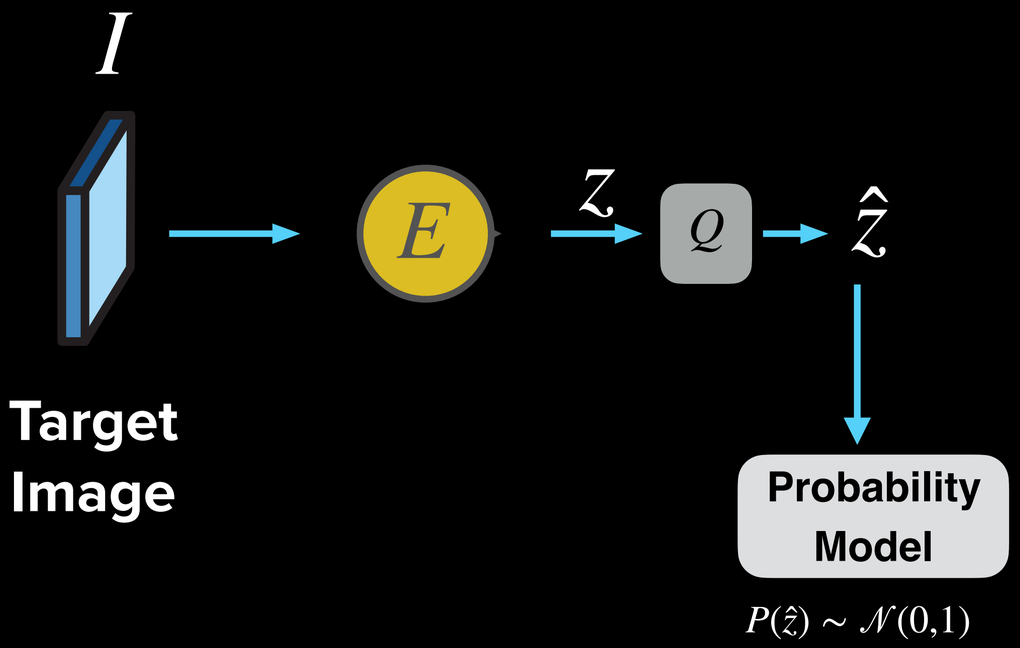
  - Multiscale Encoder/Decoder: [Rippel et. al. 2018]

  - using GDN non-linearity: [Balle, 2017/18]

Design Decisions: (ii) Probability Model



Design Decisions: (ii) Probability Model

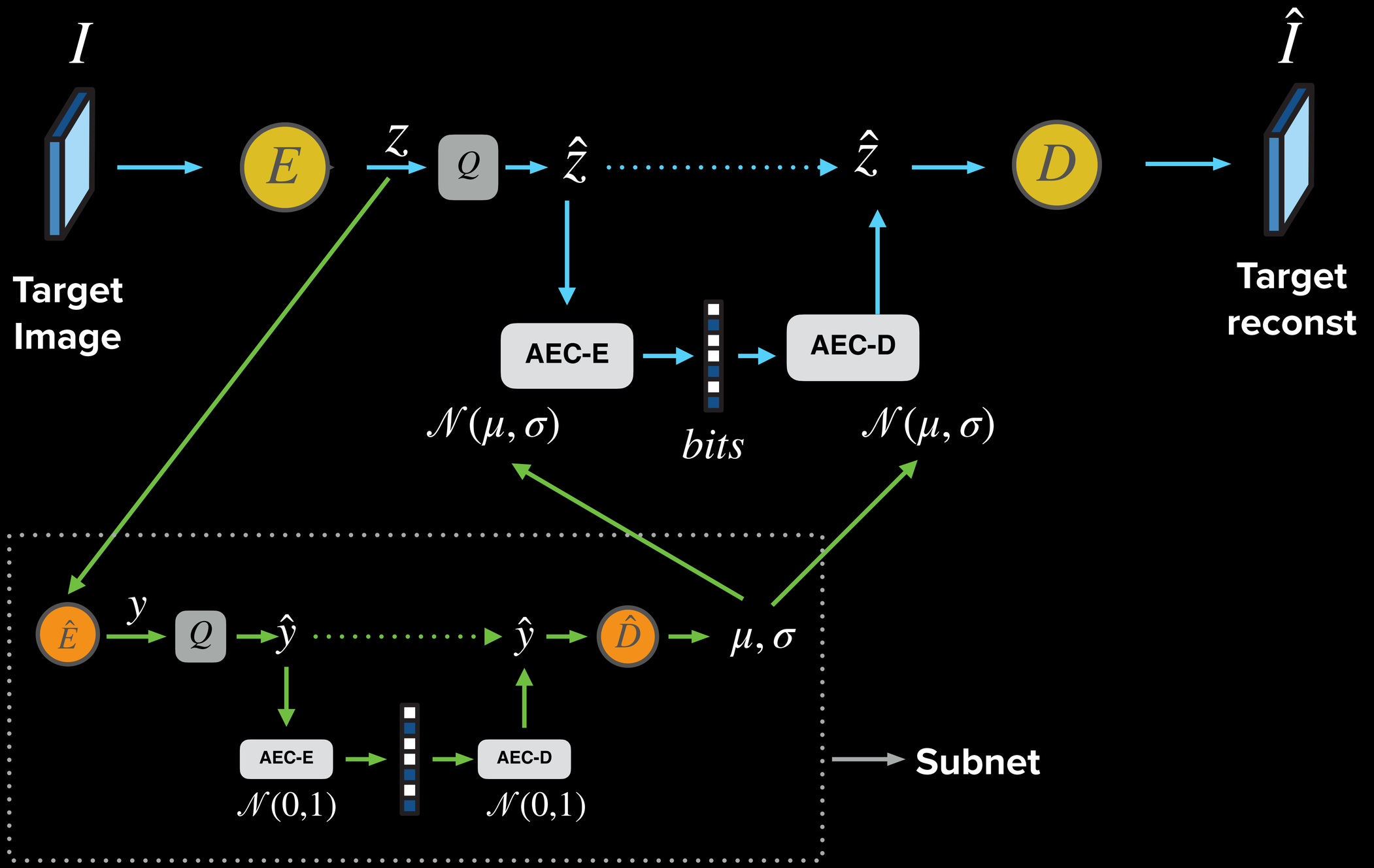


- More complex Probability models
- $PDF(\hat{z}_i) \rightarrow \mathcal{N}(\mu_i, \sigma_i)$,

  i.e: μ, σ are different per element of $\hat{z}$
- Need to now encode μ, σ tensors:

  - *Hyperprior* approach

  [Balle, ICLR18]


I
$\hat{I}$
E
z
Q
$\hat{z}$
$\hat{z}$
D
Target Image
Target reconst
AEC-E
AEC-D
$\mathcal{N}(\mu, \sigma)$
$bits$
$\mathcal{N}(\mu, \sigma)$
$\hat{E}$
y
Q
$\hat{y}$
$\hat{y}$
$\hat{D}$
μ, σ
AEC-E
AEC-D
$\mathcal{N}(0,1)$
$\mathcal{N}(0,1)$
Subnet

Design Decisions: (ii) Probability Model

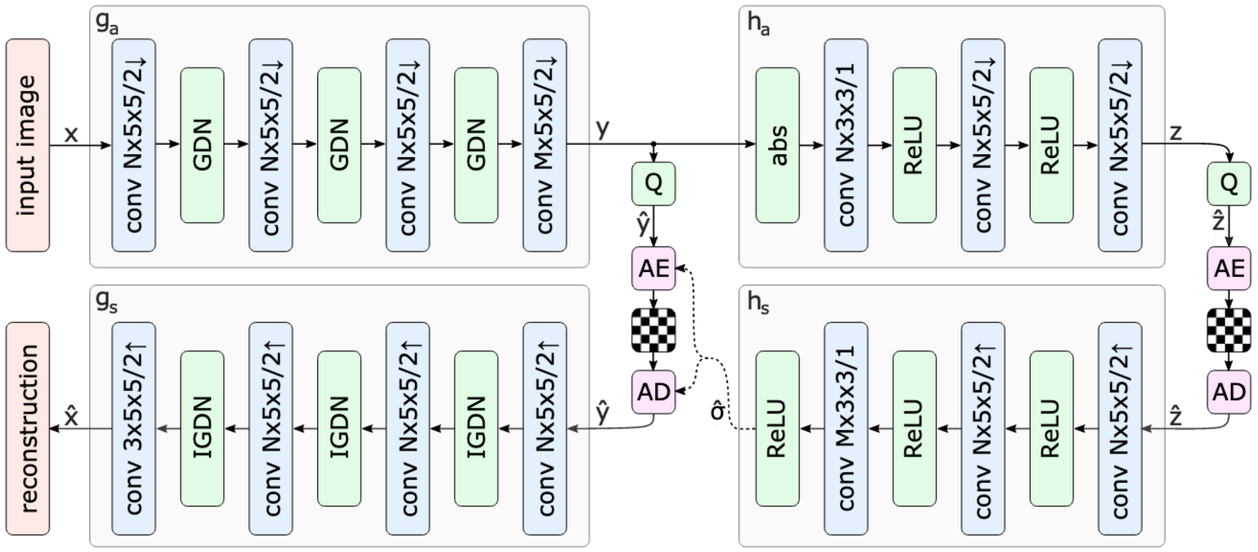


Figure 4: Network architecture of the hyperprior model. The left side shows an image autoencoder architecture, the right side corresponds to the autoencoder implementing the hyperprior. The factorized-prior model uses the identical architecture for the analysis and synthesis transforms g_a and g_s. Q represents quantization, and AE, AD represent arithmetic encoder and arithmetic decoder, respectively. Convolution parameters are denoted as: number of filters × kernel support height × kernel support width / down- or upsampling stride, where ↑ indicates upsampling and ↓ downsampling. N and M were chosen dependent on λ, with $N = 128$ and $M = 192$ for the 5 lower values, and $N = 192$ and $M = 320$ for the 3 higher values.

Variational image compression with a scale hyperprior

Balle et.al. 2018

Design Decisions: (ii) Probability Model

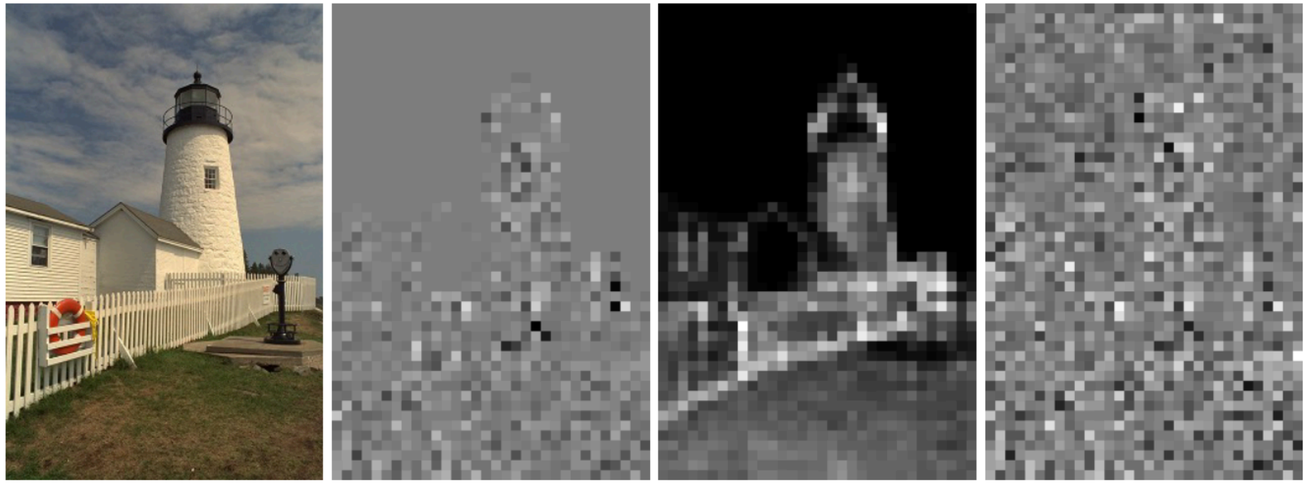
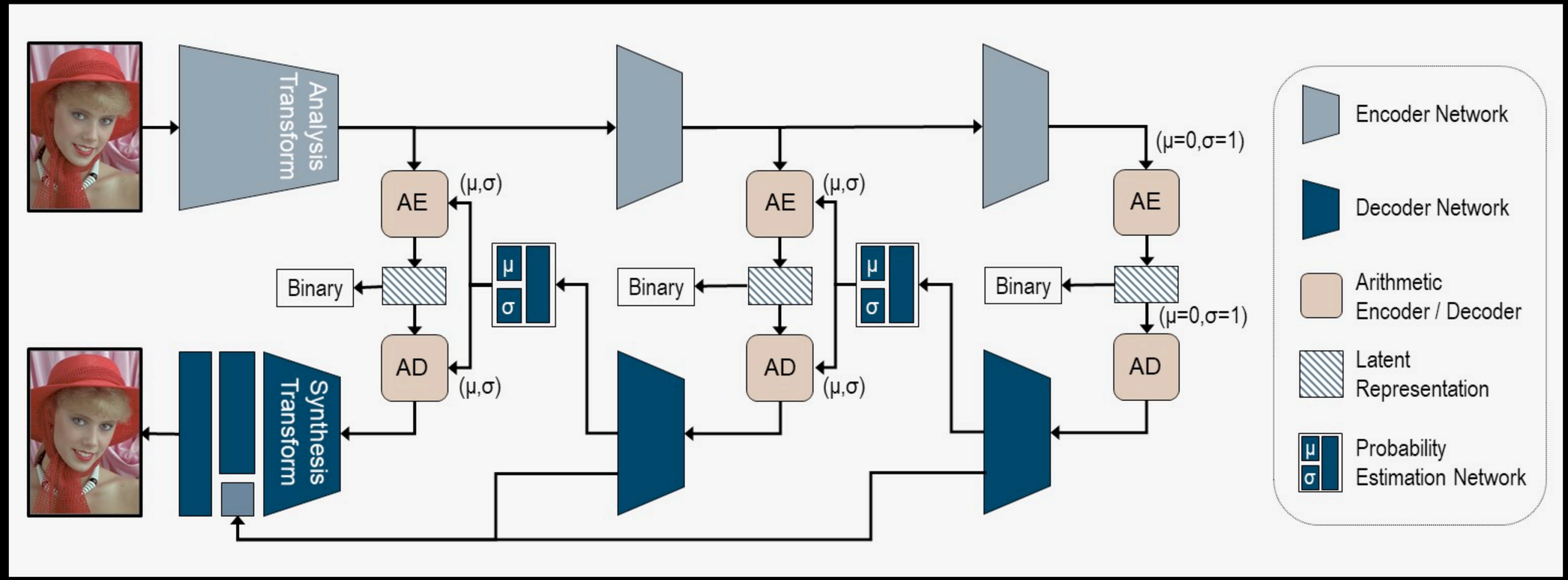


Figure 2: Left: an image from the Kodak dataset. Middle left: visualization of a subset of the latent representation $\boldsymbol{y}$ of that image, learned by our factorized-prior model. Note that there is clearly visible structure around edges and textured regions, indicating that a dependency structure exists in the marginal which is not represented in the factorized prior. Middle right: standard deviations $\hat{\boldsymbol{\sigma}}$ of the latents as predicted by the model augmented with a hyperprior. Right: latents $\boldsymbol{y}$ divided elementwise by their standard deviation. Note how this reduces the apparent structure, indicating that the structure is captured by the new prior.

Design Decisions: (ii) Probability Model



Hyper-hyper-prior models:
https://huzi96.github.io/coarse-to-fine-compression.html

Benchmarks

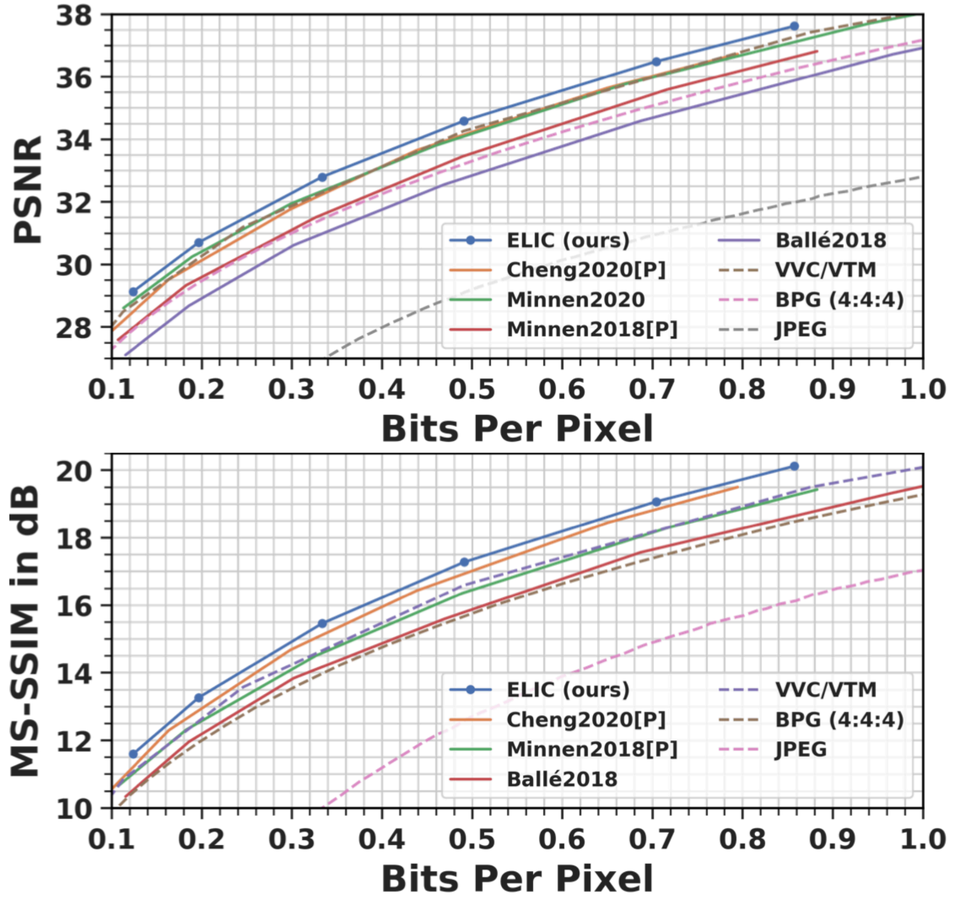


Figure 9. Rate-distortion curves of various image compression approaches. The results are evaluated on Kodak. All shown learned models are optimized for minimizing MSE.

ELIC: Efficient Learned Image Compression with Unevenly Grouped Space-Channel Contextual Adaptive Coding

He et.al. 2022

Benchmarks

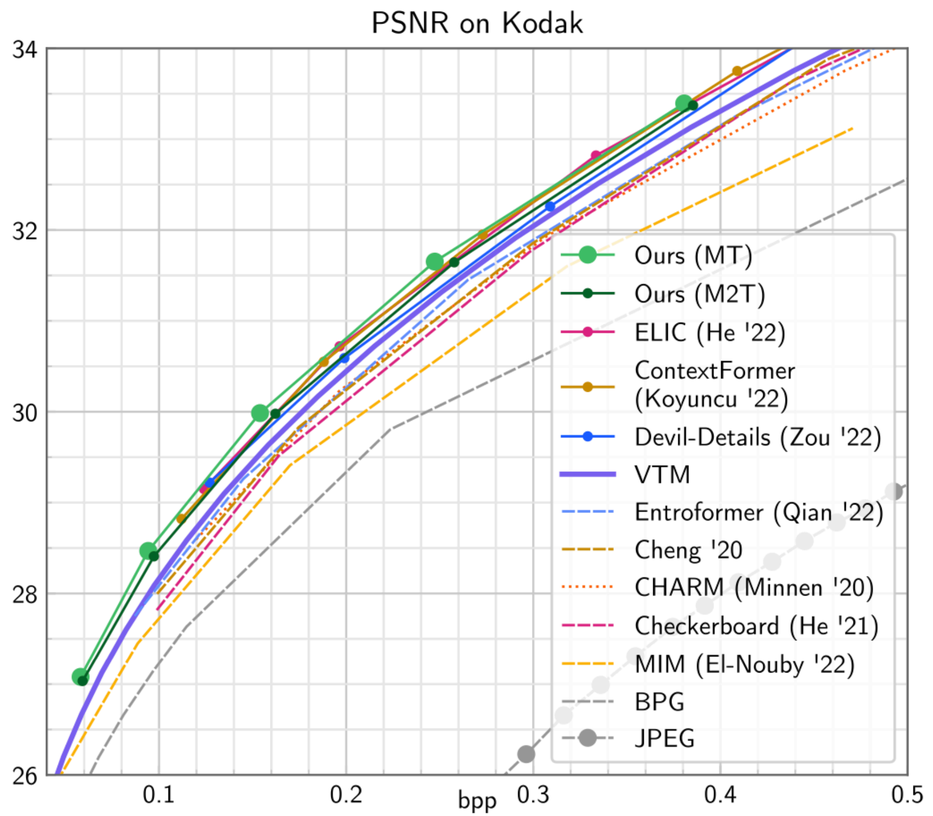
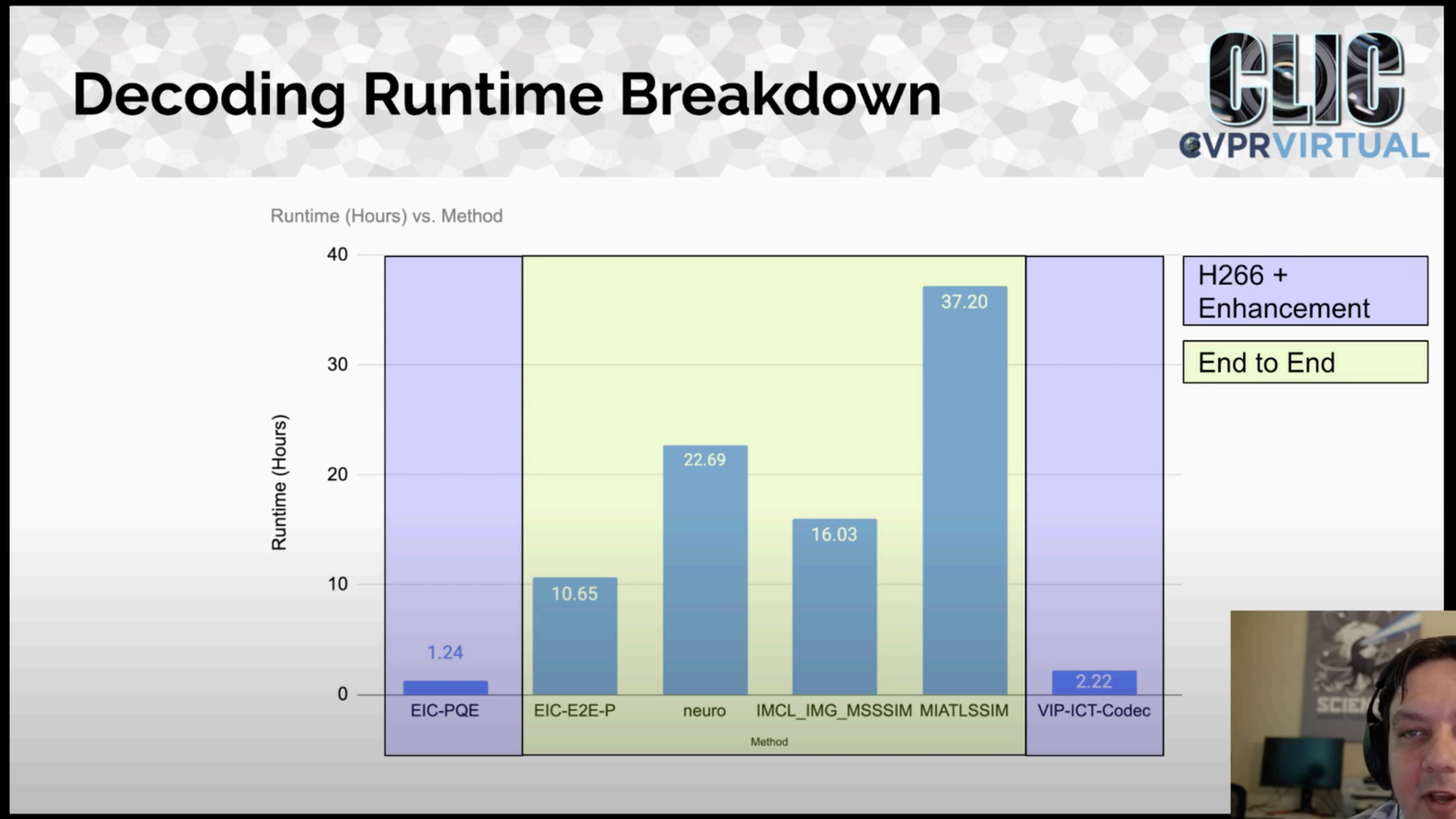


Figure 1: Rate distortion results on Kodak. Our *MT* outperforms the prior state-of-the-art ELIC [18]; *M2T* only incurs a small reduction in rate-distortion performance compared to MT while running about 4× faster on hardware (see Fig. 4)

M2T: Masking Transformers Twice for Faster Decoding
Mentzer et.al. 2023

Issues: (i) Speed!



Slide from Dr. Toderici's talk at CVPR20

Issues: (i) Speed!

- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

Issues: (i) Speed!

- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

- **For example:** [ELF-VC, Rippel et.al. 2021, ArXiv]

  real-time HD 720 decode on mid-range GPU

  - VGA 640x480: encode @ 47 FPS, decode @ 91 FPS

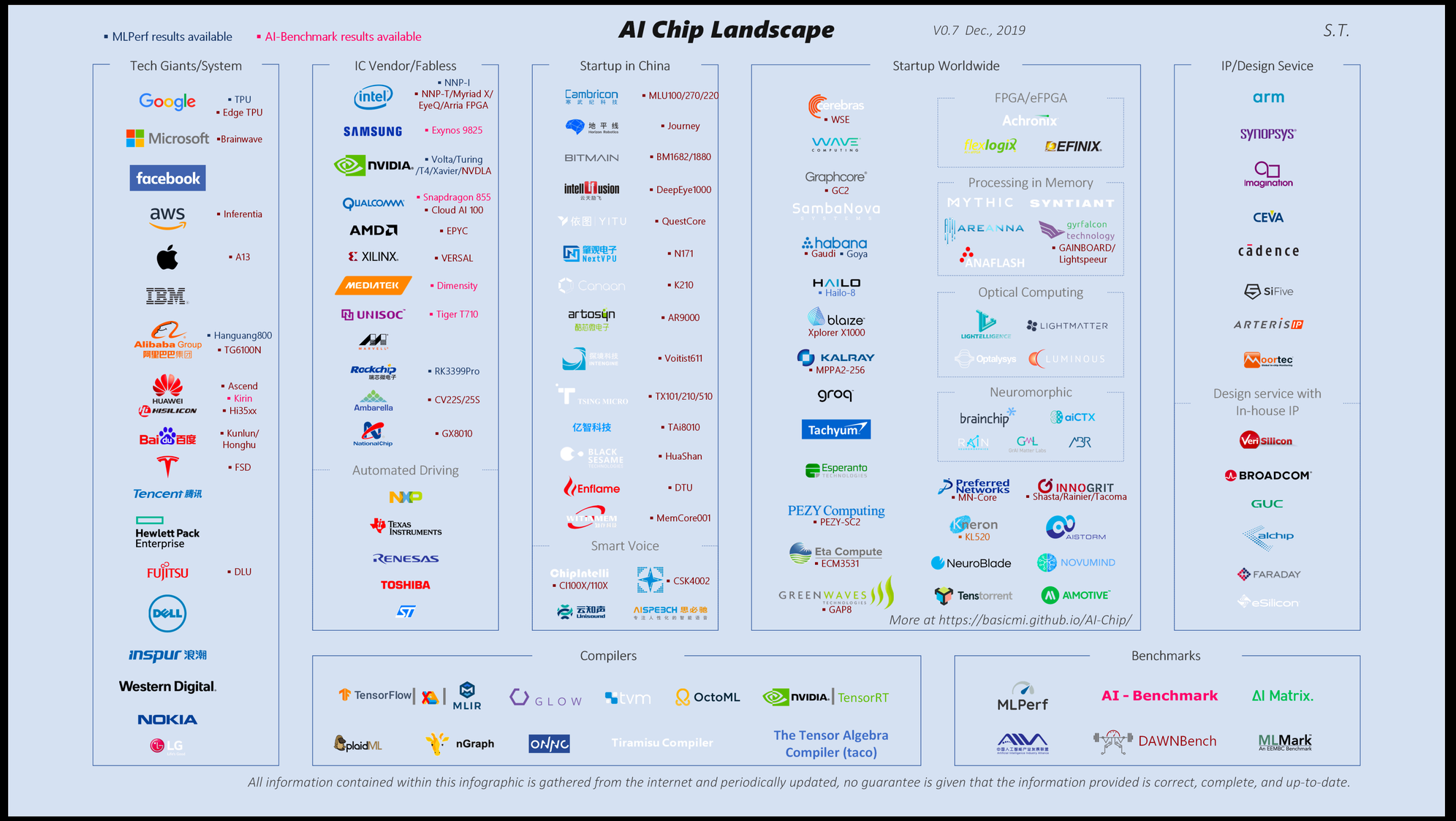
  - HD 1280x720: encode @ 19 FPS, decode @ 35 FPS

Issues: (i) Speed!

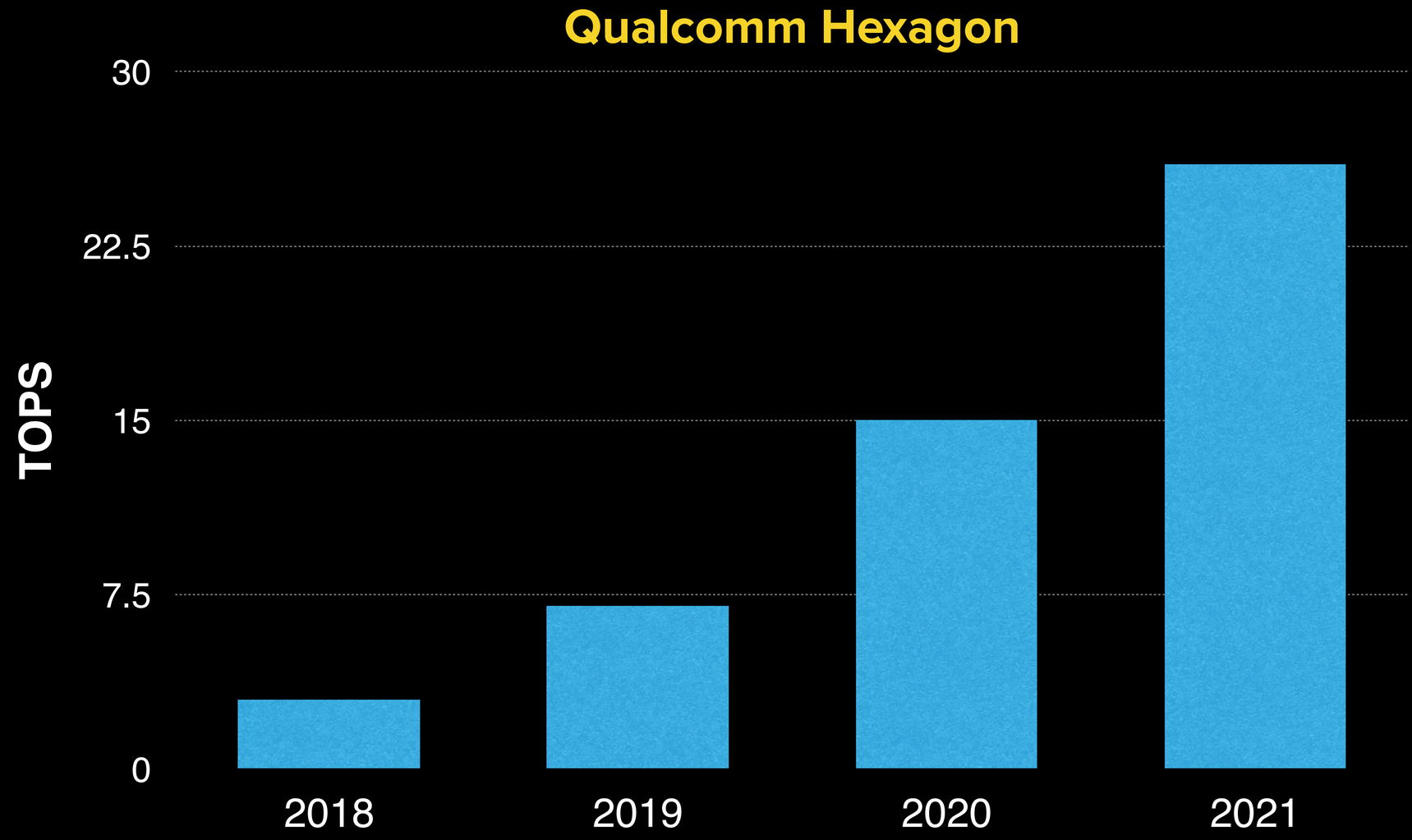
- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

- **Faster Hardware:** Hardware support for NN keeps improving year by year

Issues: (i) Speed!



Issues: (i) Speed!

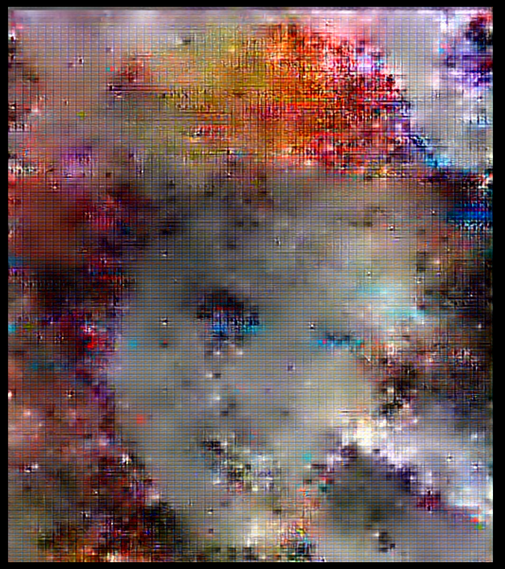


Issues: (ii) Determinism!

Different hardware =>

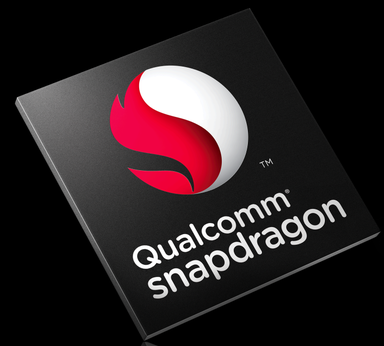
different floating point implementation =>

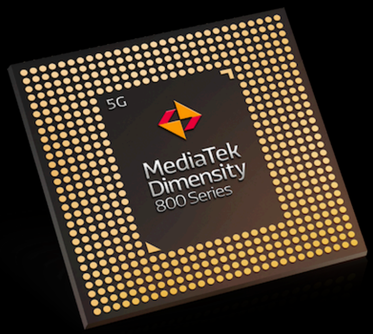
catastrophic failures!



Issues: (ii) Determinism!

- Encoding/decoding must yield exactly the same outputs, irrespective of hardware architecture





- Floating point (FP32/16) models don’t work:

  Model Quantization necessary [E.g: Ballé 2019]

Take-aways from Today

Good predictor implies ***good compressor***; and vice-versa

Traditional image compressors have ***lot of hand-tuned*** parameters

ML-based codecs allow for ***learned encoder-decoder transforms*** and therefore

(i) better fidelity to chosen probability model over latents => better rate-distortion

(ii) allow for substituting distortion of the choice

(iii) domain adaptable and flexible

Main idea to achieve ML-based codec is to ***overcome (automatic) differentiation*** over

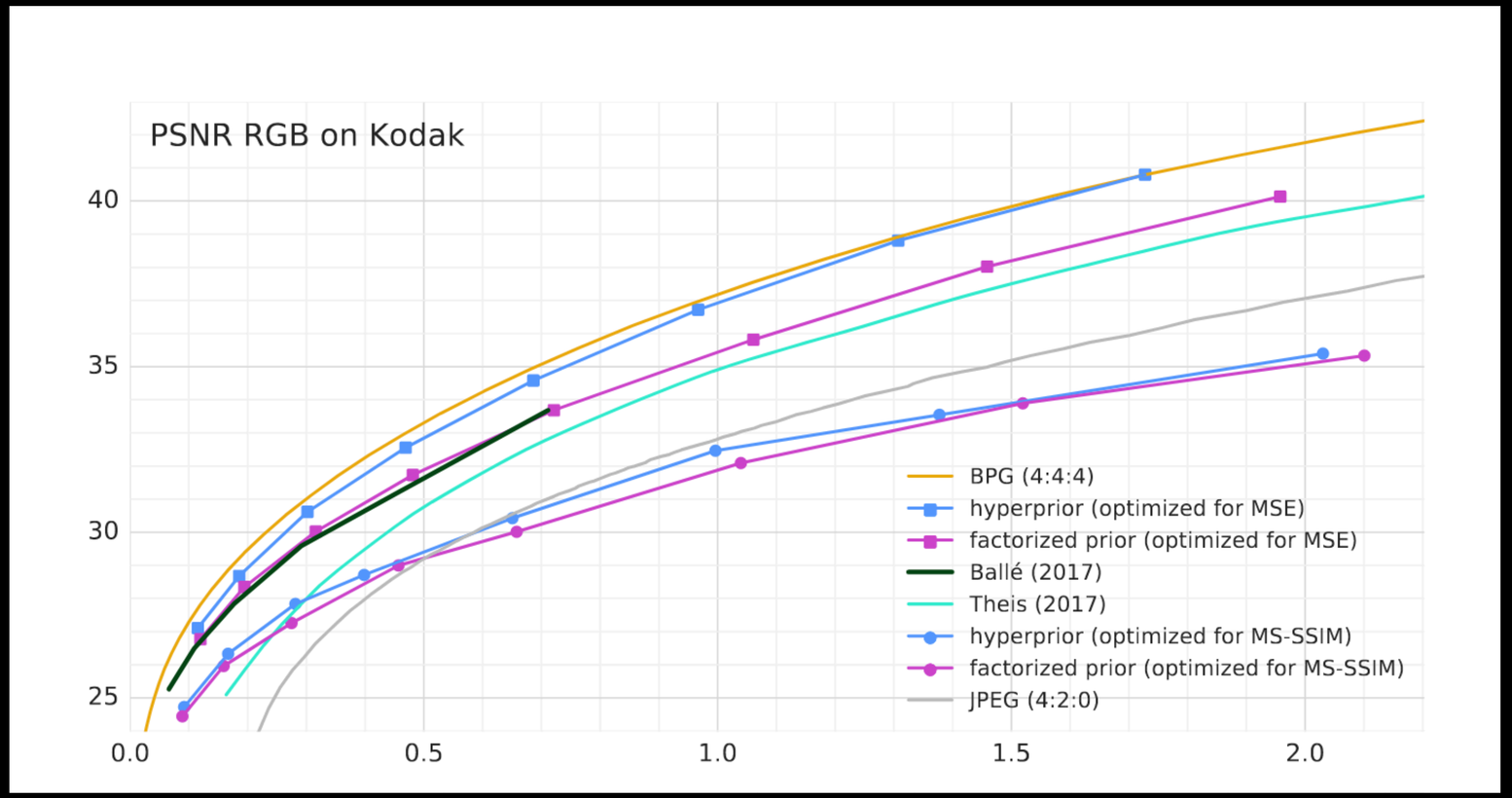
(i) quantization and (ii) discrete probability models

ML-based methods will likely form the ***basis of future compression***

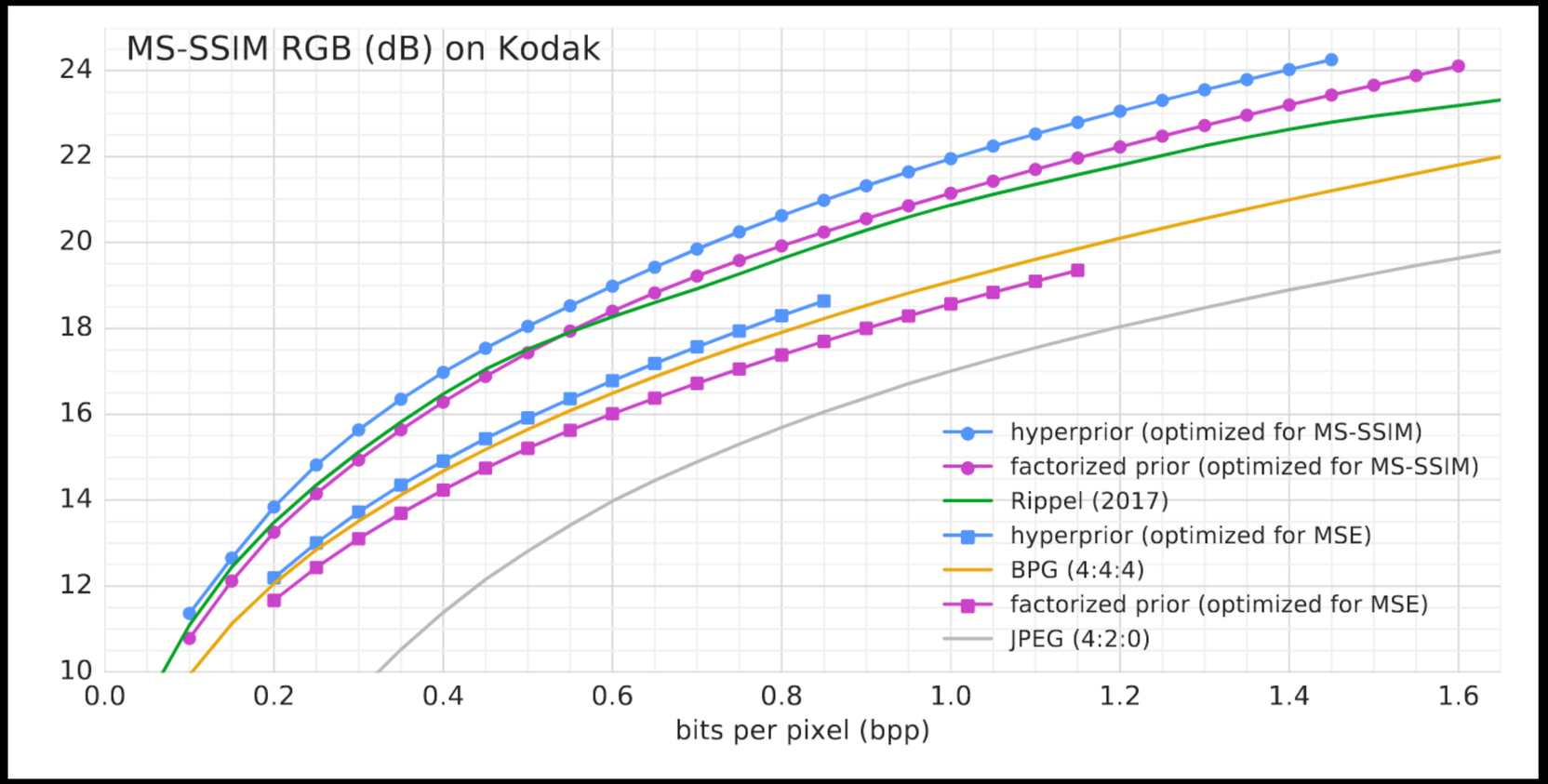
Thank you!
Questions?!

Backup

Compression Performance



Compression Performance



Hyper-hyper-prior models

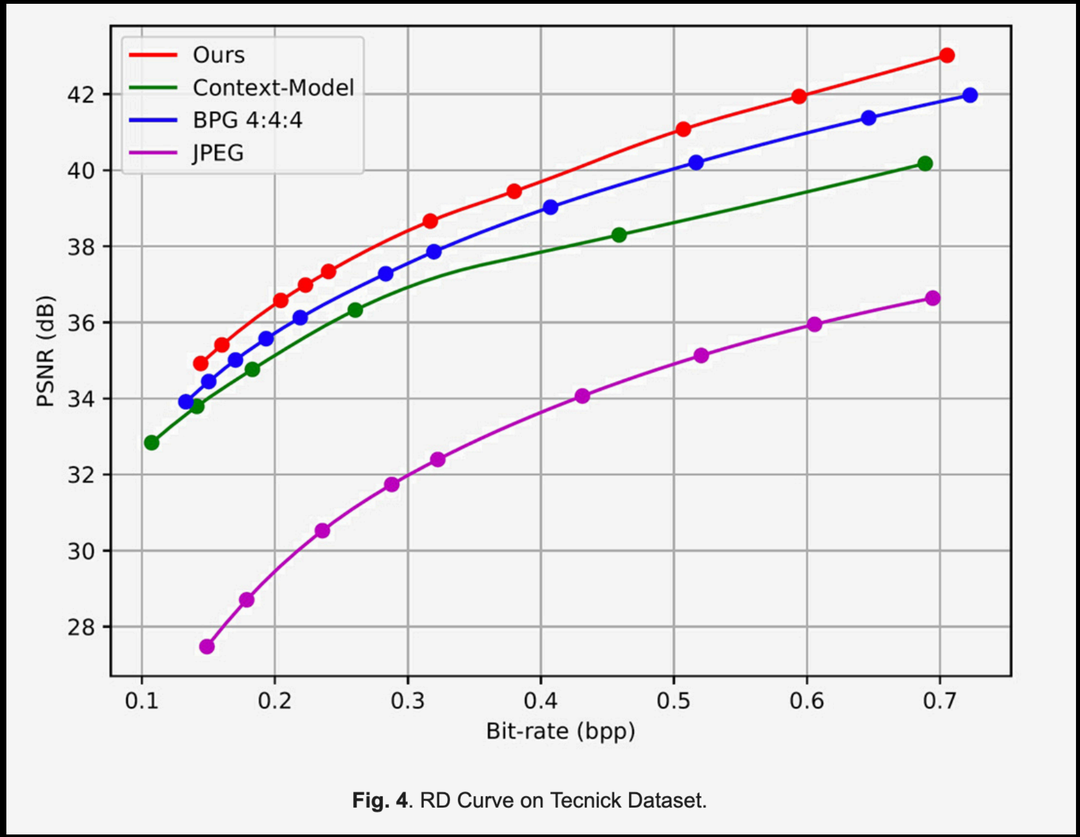
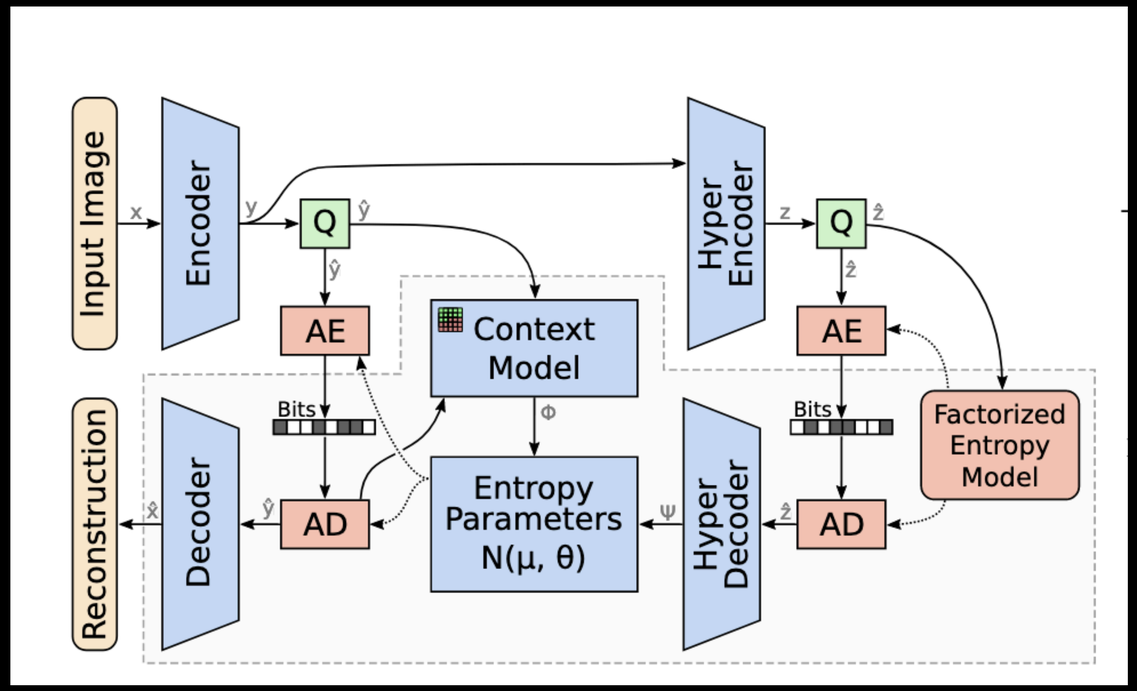


Fig. 4. RD Curve on Tecnick Dataset.

- https://huzi96.github.io/coarse-to-fine-compression.html

Hyperprior with Context model



Joint Autoregressive and Hierarchical Priors for Learned Image Compression [2019, Minnen et.al]