ECE 5578 Multimedia Communication

Lec 06 - Arithmetic Coding II
Context Models

Zhu Li

Dept of CSEE, UMKC

Office: FH560E, Email: lizhu@umkc.edu, Ph: x 2346.

http://l.web.umkc.edu/lizhu

slides created with WPS Office Linux and EqualX LaTex equation editor

Outline

- Lecture 05 Re-Cap
- Context Adaptive Arithmetic Coding
  - PAQ
  - CABAC in H264
  - Context model in deep learning compression

Arithmetic Coding ReCAp

Aliens visit earth….

- Arithmetic coding:
  - Symbols $A=\{a_1, a_2, \ldots, a_n\}$, with prob=$\{p_1, p_2, \ldots, p_n\}$
  - For a sequence of symbol, $X=[x_1, x_2, \ldots, x_m]$ compute the sequence prob, P(X), the binary code is,

$$l(X) = \left\lceil \log \frac{1}{p(X)} \right\rceil + 1 \text{ bits.}$$

of the F(X) representation.

- Sequence prob based, more efficient than Huffman codes

Example - Arithmetic

❑ Encoding:

- pmf: [0.7, 0.1, 0.2], CDF: $F_x()$=[0.7 0.8 1]

```
LOW=0.0, HIGH=1.0;
while (not EOF) {
   n = ReadSymbol();
   RANGE = HIGH - LOW;
   HIGH = LOW + RANGE * CDF(n);
   LOW  = LOW + RANGE * CDF(n-1);
}
output LOW;
```

- T(X)=(0.546+0.560)/2=0.553
- P(X)= 0.014; l(X)=7 bits
- Code: $(0.553)_2$ = 0.100011011

$$l(X) = \left\lceil \log \frac{1}{p(X)} \right\rceil + 1 \text{ bits.}$$

Consider a three-letter alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. Using the mapping of Equation (4.1), $F_X(1) = 0.7$, $F_X(2) = 0.8$, and $F_X(3) = 1$. This partitions the unit interval as shown in Figure 4.1.

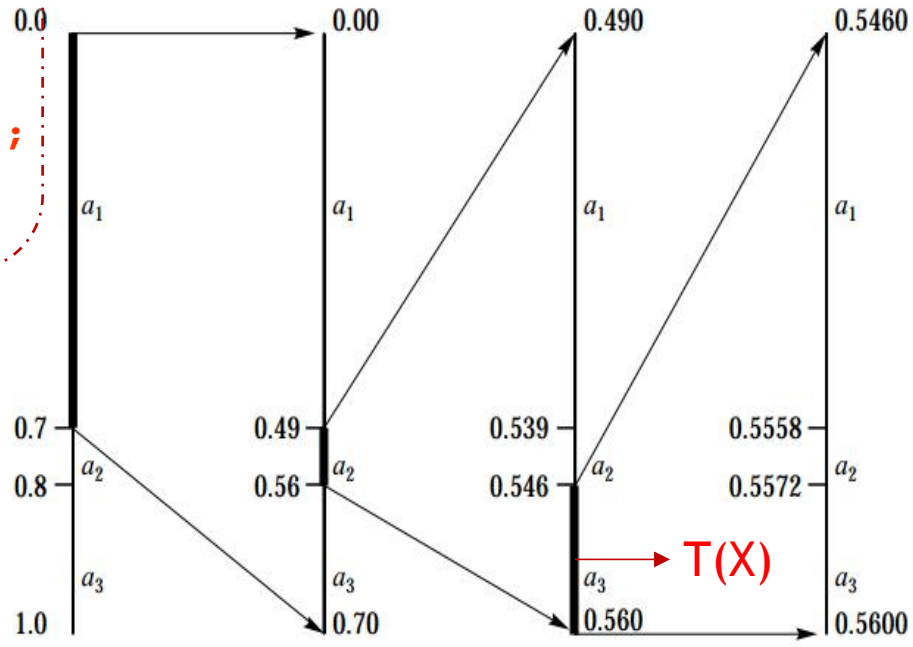
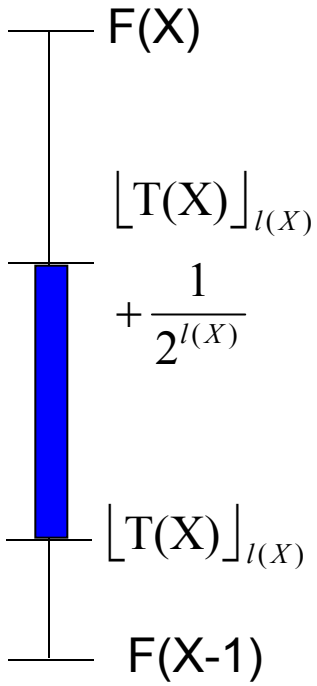


FIGURE 4. 1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

Uniqueness and Efficiency

- Prove the code is uniquely decodable (prefix free):
- Any code with prefix $\lfloor T(X) \rfloor_{l(X)}$ is in

$$\left[\lfloor T(X) \rfloor_{l(X)}, \lfloor T(X) \rfloor_{l(X)} + \frac{1}{2^{l(X)}} \right)$$



- Need to show that this is in [F(X-1), F(X)):

We already show $\lfloor T(X) \rfloor_{l(X)} \geq F(X-1).$

Only need to show $F(X) - \lfloor T(X) \rfloor_{l(X)} > \frac{1}{2^{l(X)}}$

$$F(X) - \lfloor T(X) \rfloor_{l(X)} > F(X) - T(X) = \frac{p(X)}{2} > \frac{1}{2^{l(X)}}$$

→ $\lfloor T(X) \rfloor_{l(X)}$ is prefix free if $l(X) = \left\lceil \log \frac{1}{p(X)} \right\rceil + 1$ bits.

Uniqueness and Efficiency

Efficiency of arithmetic code:

$$l(X_1^m) = \left\lceil \log \frac{1}{p(X_1^m)} \right\rceil + 1 \text{ bits.} \qquad X_1^m : \{x_1, \ldots, x_m\}$$

$$L = E\left\{p(X_1^m) l(X_1^m)\right\} = \sum P(X_1^m) \left(\left\lceil \log \frac{1}{p(X_1^m)} \right\rceil + 1 \right)$$

$$\le \sum P(X_1^m) \left(\log \frac{1}{p(X_1^m)} + 1 + 1 \right) = H(X_1^m) + 2$$

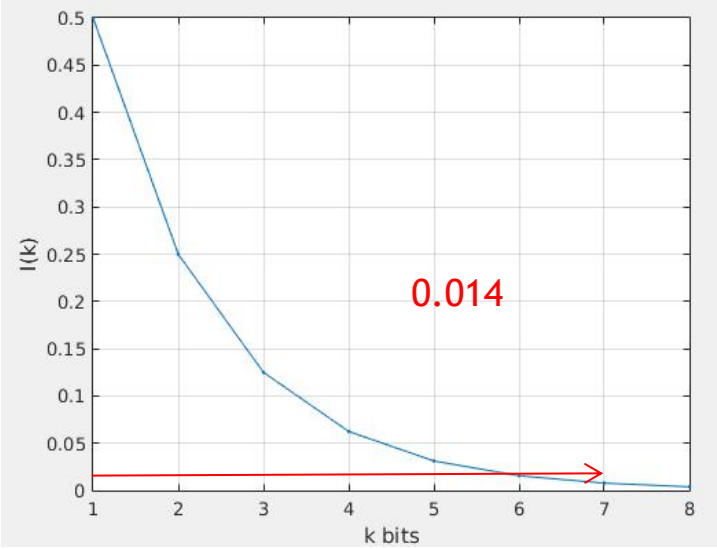
l(X) is the bits to code a sequence {x_1,x_2, …, x_m}.
Assume iid sequence, $H(X_1^m) = mH(X)$

$$H(X) \le \frac{L}{m} \le H(X) + \frac{2}{m}$$

L/m → H(X) for large m, stronger than prev results.

AC example

- Take pmf=[0.7, 0.1, 0.2], F(1)=0.7, F(2)=0.8, F(3)=1 , and seq: X={a_1, a_2, a_3}
  - p(X) = 0.5600 - 0.5460=0.014
  - l(X) = 7 bits
  - coding length: 8 bits/0.0039
  - 0.0078/7bits not working



Consider a three-letter alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. Using the mapping of Equation (4.1), $F_X(1) = 0.7$, $F_X(2) = 0.8$, and $F_X(3) = 1$. This partitions the unit interval as shown in Figure 4.1.

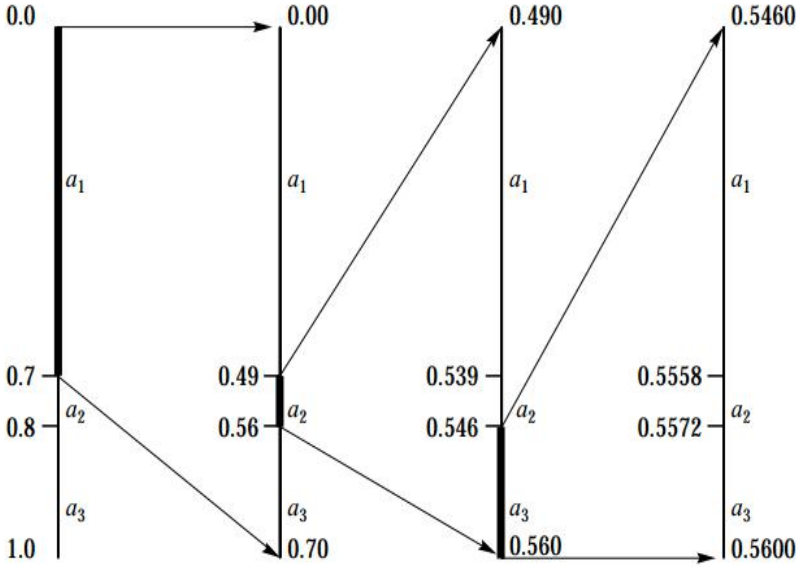


FIGURE 4. 1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

L(k) = 0.5000 0.2500 0.1250 0.0625 0.0312 0.0156 0.0078 0.0039

Encoding of Binary Arithmetic Coding

$$HIGH \leftarrow LOW + RANGE \times CDF(n)$$

$$LOW \leftarrow LOW + RANGE \times CDF(n-1)$$

LOW = 0, HIGH = 1

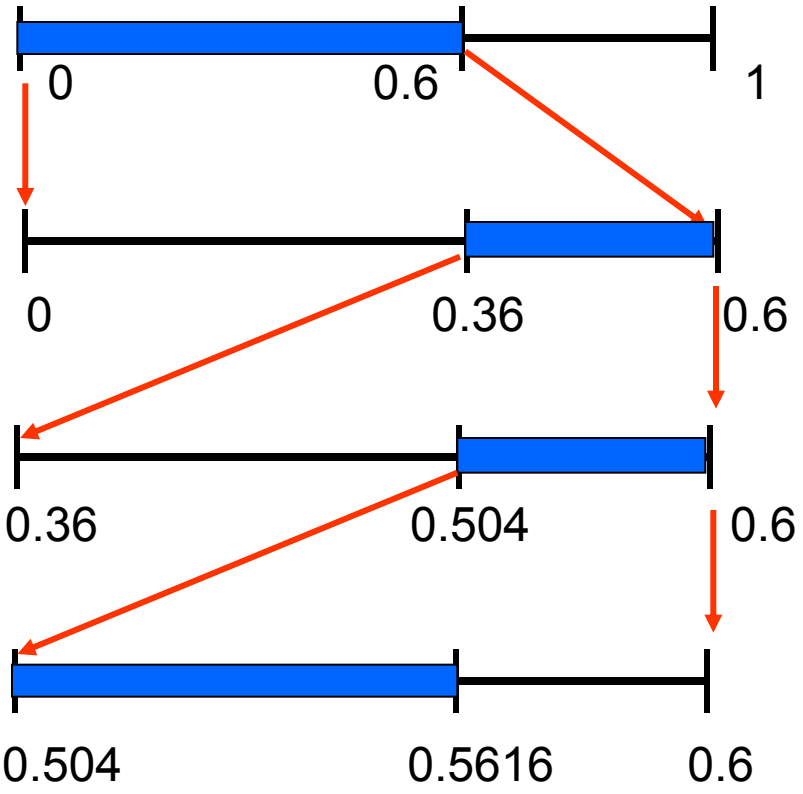
LOW = 0, HIGH = 0.6

LOW = 0.36, HIGH = 0.6

LOW = 0.504, HIGH = 0.6

LOW = 0.504, HIGH = 0.5616

Prob(0)=0.6. Sequence: 0110



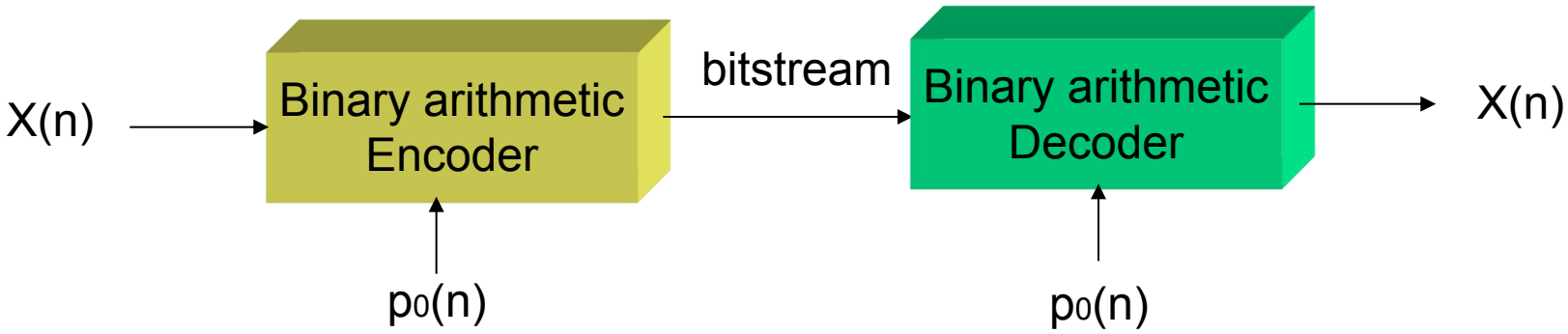
Only need to update LOW or HIGH for each symbol.

Outline

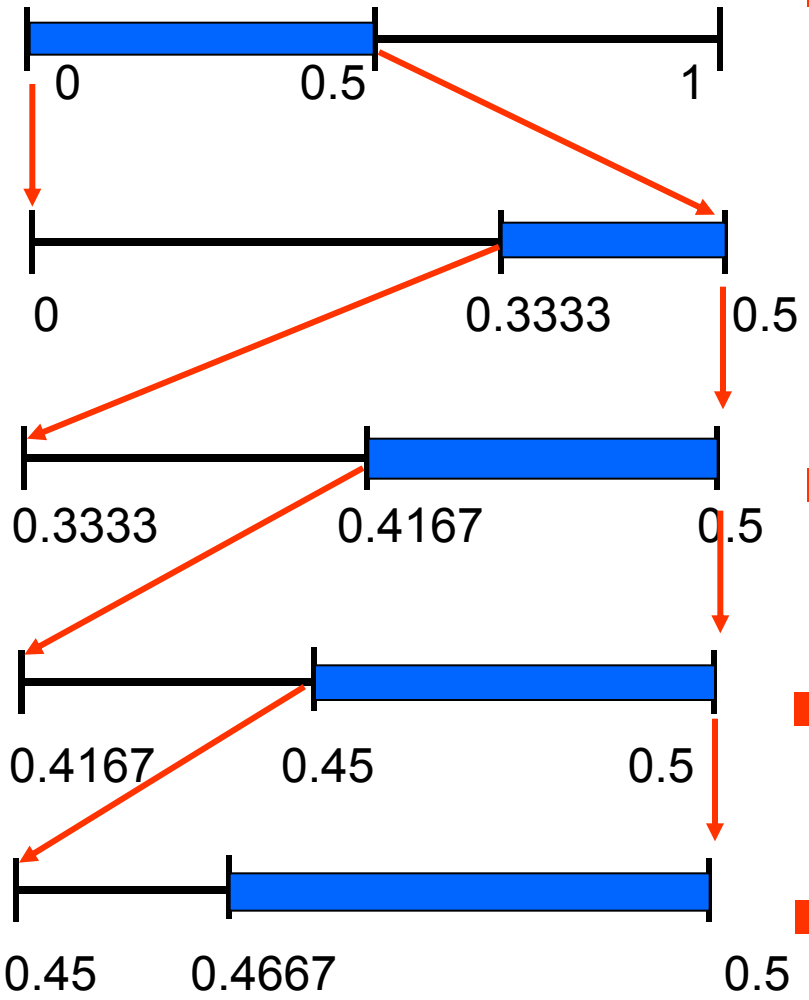
- Lecture 05 Re-Cap
- Context Adaptive Arithmetic Coding
  - PAQ
  - CABAC in H264
  - Context model in deep learning compression

Adaptive Arithmetic Coding

- Observation:
  - The partition of [0, 1) can be different from symbol to symbol
- The bit stream can be decoded perfectly as long as both encoder and decoder are synchronized (use the same probability).
- Only need to update the probabilities
- Adaptive Huffman code:
  - Has to redesign the codebook each time
  - Time-consuming.

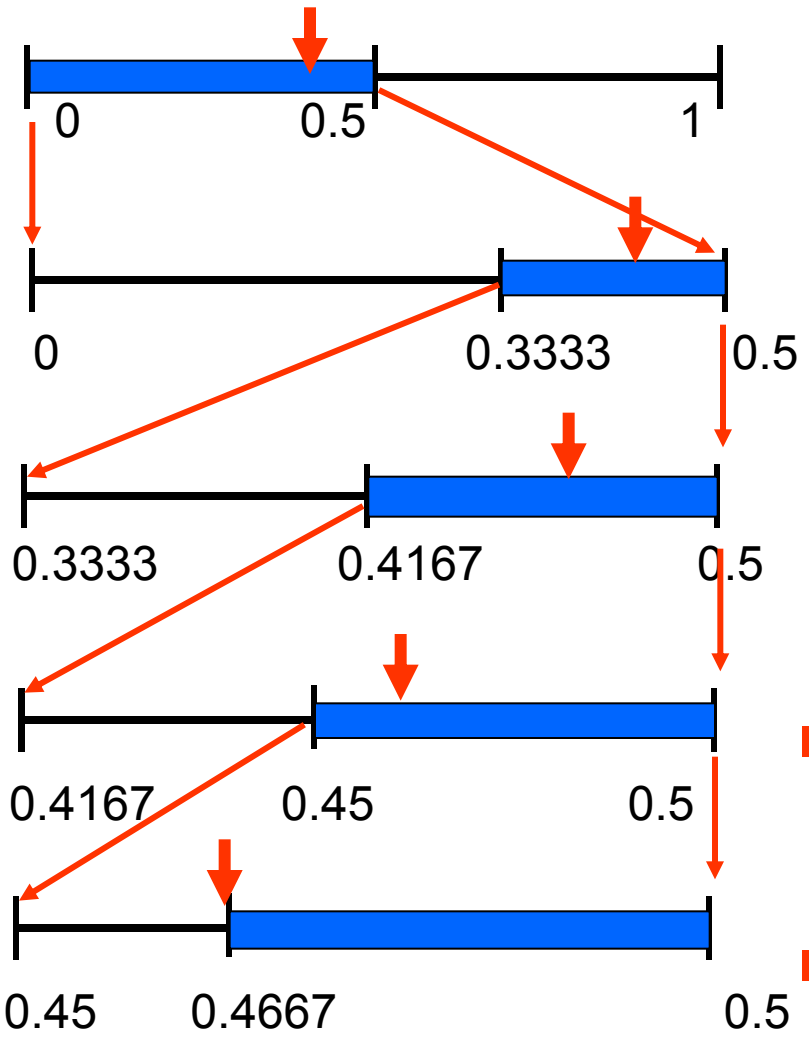


Example



- Binary sequence: 01111
- Initial counters for 0's and 1's: C(0)=C(1)=1.
  ➔ P(0)=P(1)=0.5
- After encoding 0: C(0)=2, C(1)=1.
  →P(0)=2/3, P(1)=1/3
- After encoding 01: C(0)=2, C(1)=2.
  →P(0)=1/2, P(1)=1/2
- After encoding 011: C(0)=2, C(1)=3.
  →P(0)=2/5, P(1)=3/5
- After encoding 0111: C(0)=2, C(1)=4.
  →P(0)=1/3, P(1)=2/3.
- Encode 0.4667.

Decoding - FIFO

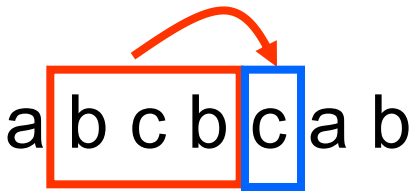


- Input 0.4667.
- Initial counters for 0's and 1's:
  C(0)=C(1)=1 ➔ P(0)=P(1)=0.5
  Decode 0
- After decoding 0: C(0)=2, C(1)=1.
  →P(0)=2/3, P(1)=1/3
  Decode 1
- After decoding 01: C(0)=2, C(1)=2.
  →P(0)=1/2, P(1)=1/2
  Decode 1
- After decoding 011: C(0)=2, C(1)=3.
  →P(0)=2/5, P(1)=3/5
  Decode 1
- After decoding 0111: C(0)=2, C(1)=4.
  →P(0)=1/3, P(1)=2/3.
  Decode 1

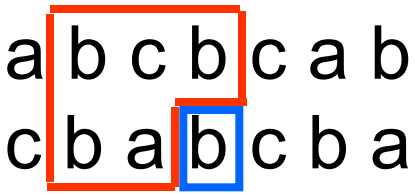
Context-adaptive Arithmetic Coding

- In many cases, a sample has strong correlation with its near neighbors.
- Idea:
  - Collect conditional probability distribution of a symbol for given neighboring symbols (context):

    P(x(n) | x(n-1), x(n-2), … x(n-k))
  - Use this conditional probability to encode the next symbol
  - More skewed probability distribution can be obtained (desired for arithmetic coding)



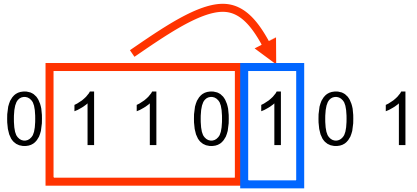
1-D Context template



2-D Context template

Example

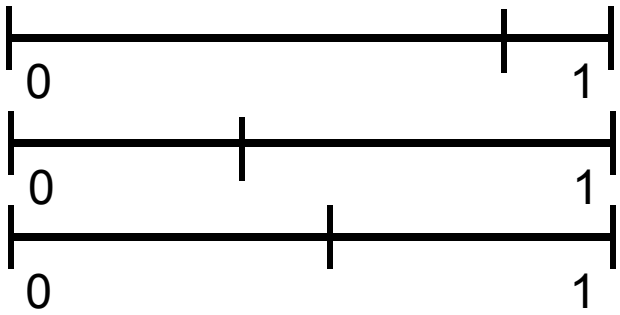
- Binary sequence: 0, 1
- Neighborhood (template) size: 3



➔2^3=8 possible combinations (contexts) of 3 neighbors (x(n-1), x(n-2), x(n-3)).

- Collect frequencies of 0's and 1's under each context

Context	C(0)	C(1)
(0, 0, 0)	9	2
(0, 0, 1)	3	6
(0, 1, 0)	10	10
(0, 1, 1)	…	…
(1, 0, 0)	…	…
(1, 0, 1)	…	…
(1, 1, 0)	…	…
(1, 1, 1)	…	…



Each symbol is coded with the probability distribution associated with its context.

PAQ1 Algorithm

- Modeling large context of binary strings..
  - for an input string 0000001111..., maintain the count of 0 and 1
  - update prob of 1 as p(1)

If the training bit is y (0 or 1) then increment c_y (c_0 or c_1).
If $c_{1-y} > 2$, then set $c_{1-y} = c_{1-y} / 2 + 1$ (rounding down if odd).

Input	c_0	c_1	p(1)+bland	weight
0000000	7	0	1/9 = 0.11	10
00000001	4	1	2/7 = 0.28	9
000000011	3	2	3/7 = 0.43	7
0000000111	2	3	4/7 = 0.57	7
00000001111	2	4	5/8 = 0.62	8

  - Then arithmetic coding with this updated p(1) as it moves along.
  - At the decoder, this can be repeated as AC decode the FIFO way.

PAQ1 Implementation and Performance

- PAQ1 is implemented with a large window on binary strings
  - rotating 4 MB (222 character) buffer and an index of 1M (220) 3-byte pointers
  - indexed by a hash of the last 8 bytes with no collision resolution.
  - reuse 8byte-64bit context model in driving the AC
- Performance:

Program	Options	Type	MB	14 files	Time	Concat.	Reuters
Original size				3,141,622		3,141,622	28,329,337
COMPRESS		LZ78	<1	1,272,772	**1.5**	1,318,269	10,406,527
PKZIP 2.04e		LZ	0.4	1,032,290	**1.5**	1,033,217	8,334,457
GZIP 1.2.4	-9	LZ77	<1	1,017,624	2	1,021,863	8,114,057
ACB	u	AC-s	<16	766,322	110	769,363	4,567,213
BOA 0.58b	-m15 -s	PPMZ-s	15	747,749	44	769,196	4,326,917
PPMD H	e -m64 -o16	PPMII	64	744,057	5	759,674	4,286,536
	e -m64 -o8	PPMII	64	746,316	5	762,843	3,850,243
SBC 0.910	c -m3 -b16	BWT	65	740,161	4.7	819,027	4,197,501
PPMONSTR H	e -m64 -o1	PPMII	64	719,922	13	736,899	4,465,502
	e -m64 -o8	PPMII	64	726,768	11	744,960	3,917,444
RK 1.02 b5	-mx3 -M64	PPMZ-s	64	707,144	44	750,744	4,404,256
	-mx3 -M64 -ft	PPMZ-s	64	717,384	44	750,744	4,207,960
RK 1.04	-mx3 -M64	PPMZ-s	64	712,188	36	755,872	3,978,800
	-mx3 -M64 -ft	PPMZ-s	64	730,100	39	755,876	3,857,780
PPMN 1.00b1	-M:50 -MT1 -O9	PPM	50	716,297	23	748,588	3,934,032
PPMONSTR Ipre	-m64 -o128	PPMII	64	696,647	35	**703,320**	4,514,279
(3/3/02)	-m64 -o8	PPMII	64	704,914	30	707,581	**3,781,287**
DEC-SPLIT		?	<64	685,341	(27 to decompress)		
DEC-SOLID		?-s	<64	**680,558**	(27 to decompress)		
PAQ1		NS-s	48	716,704	68.1	716,240	4,078,101

Larger Non-Binary Context Model - PAQ7

- Condition reduces entropy, $H(Y|X_1) > H(Y|X_1, X_2, \ldots.)$
- How to model very large context (x1, x2, ….)?
  - Use a (shallow/deep) neural network to model context
  - Large window of 552 input nodes
  - 7 sets of 3080 hidden nodes

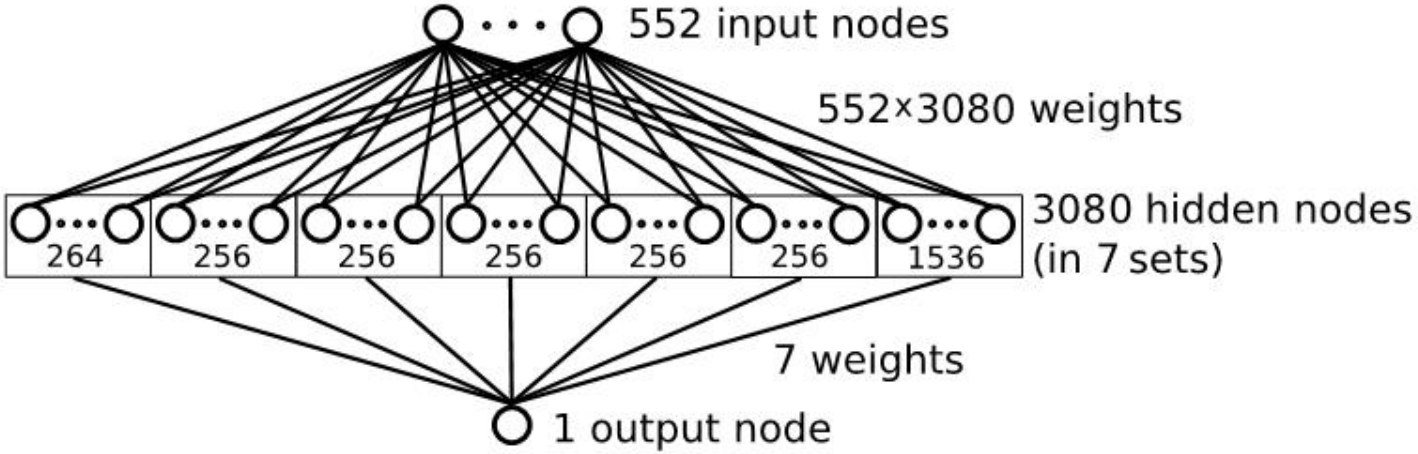
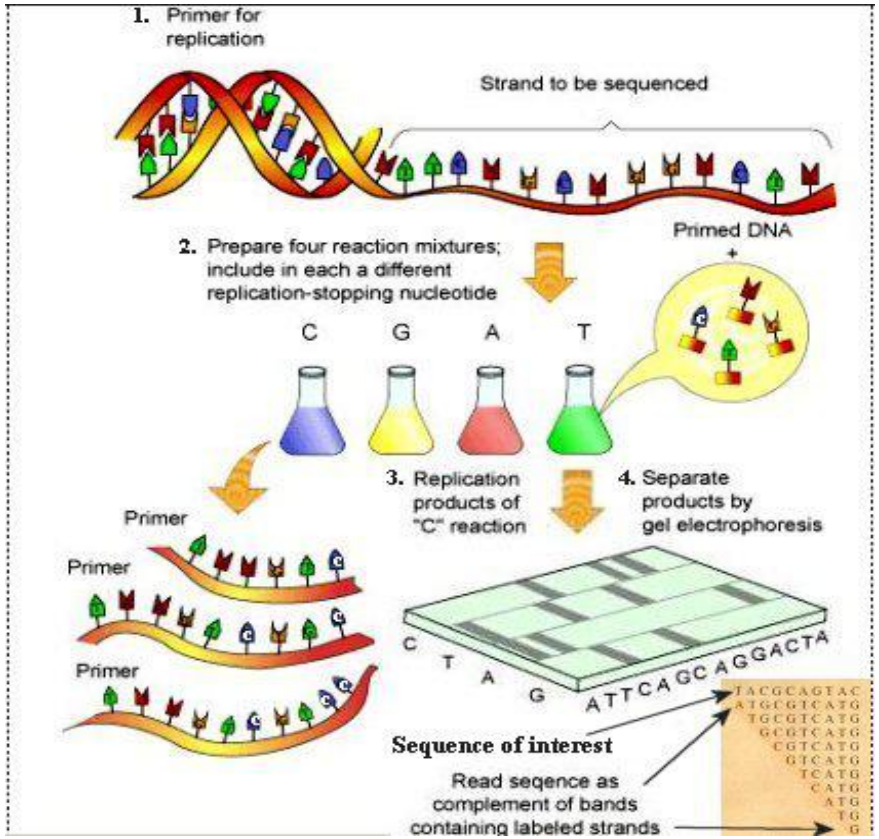


Figure 4: PAQ8 model mixer architecture.

- Implementation with a matlab wrapper (thanks to Biren Kathriya):
  https://umkc.box.com/s/kg7blpc0jpaoctgauhhfyszz178a0pcy

PAQ 7 Performance in DNA compression

- Application to DNA sequence compression



It is sequence of 4 nitrogenous bases;
GATCCTGAGCTAGG…….

ATCGTT

ATCGCT

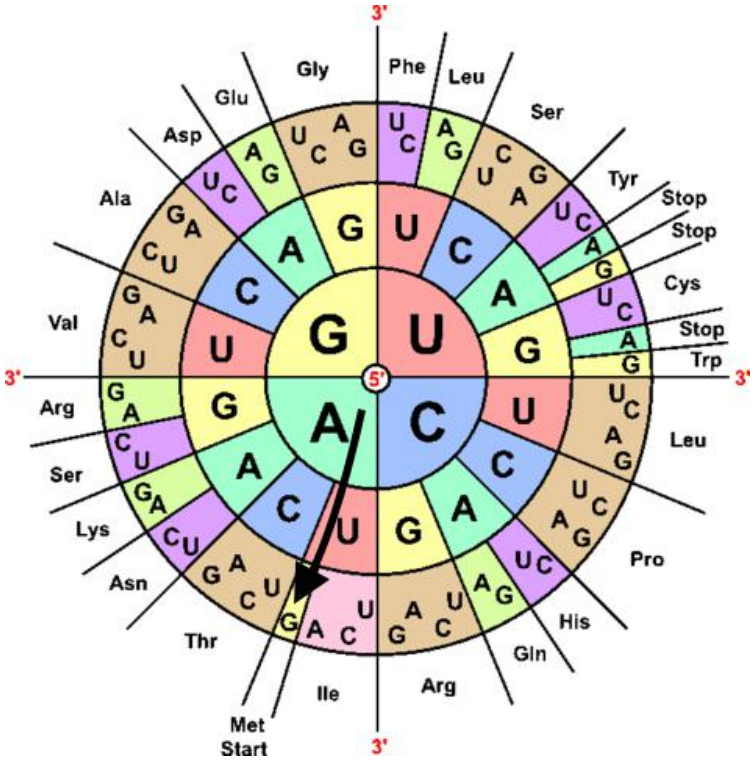
Genome Data Compression

- Compress DNA seq for bioinformatics/precision medicine applications
  - represent in Codon Seq, 64 symbols as if English alphabeta
  - Then apply PAQ for compression (need retraining)

Codon:
A sequence of three nucleotides that together form a unit of genetic code in a DNA or RNA molecule.

So 4 symbols will make 4^3=64 codons.

Codon represents a total of 20 amino acids.



Performance

- SOTA performance

Dataset	Number of Symbols	Number of bits after compression	Bit/symbol
NC_012920	16569	3990	0.2408
NC_0189127.2	8847	2117	0.2393
NC_000016.10	8847	2043	0.2309

- Can be improved with a LSTM engine for PAQ's shallow neural network.

Outline

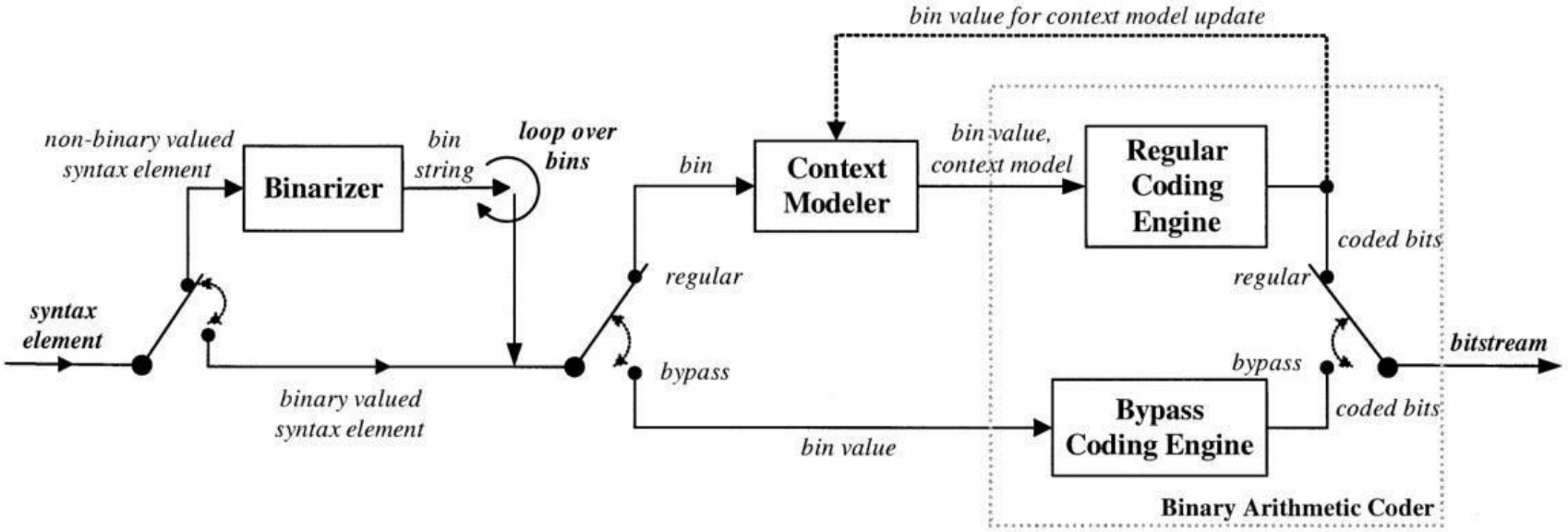
- Lecture 05 Re-Cap
- Context Adaptive Arithmetic Coding
  - PAQ
  - CABAC in H264
  - Context model in deep learning compression

Basic framework of CABAC (1)

- CABAC: Content-Based Adaptive Binary Arithmetic Coding
  - Content-Based Adaptive: Adjust the possibility of each bin according to the context or previous coded bins
  - Binary Arithmetic Coding: Arithmetic Coding based on 0 and 1

Basic framework of CABAC (2)

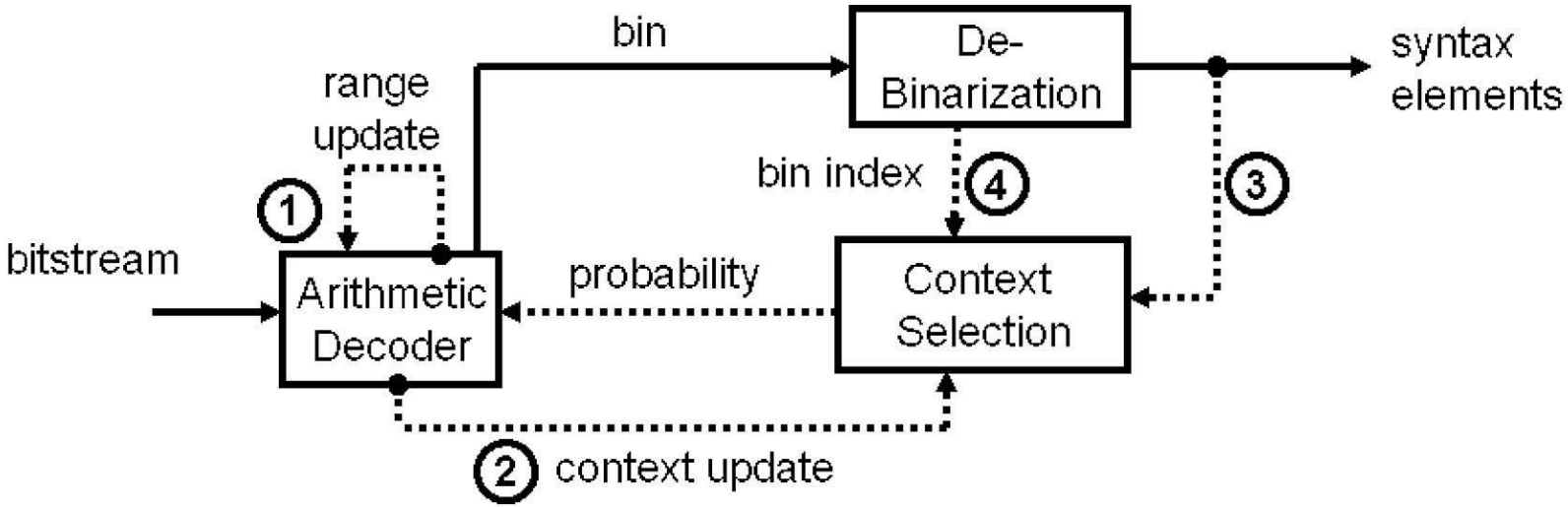
- CABAC Fraemwork
  - Step 1: Binarization
  - Step 2: Context model selection
  - Step 3: Regular/ Bypass coding engine
  - Step 4: Context model update



Basic framework of CABAC (3)

- Context model

  - Step 1: Context model selection
  - Step 2: Arithmetic decoding
  - Step 3: Context model updating
  - Step 4: De-binarization



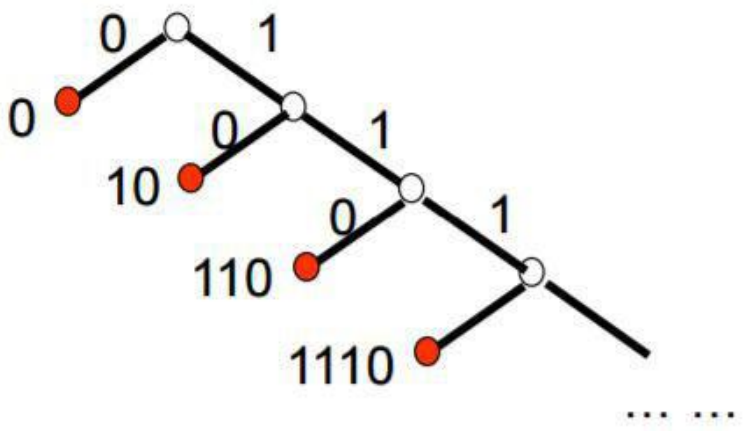
Binarization (1)

- Why Binarization? (One premise and two reasons)
  - Binarization is a lossless operation
  - Adaptive m-order (m>2) arithmetic coding is complex operation requiring at least two multiplications for each symbol to encode while binary arithmetic coding can be performed in a multiplication-free way
  - Binarization enables context modeling on a sub-symbol level
    - MSB uses context model, eg. group id in Golomb
    - LSB uses bypass coding mode (no context)

Binarization (2)

- Basic binarization schemes (1)
  - Unary Binarization

n	Codeword
0	0
1	10
2	110
3	1110
4	11110
5	111110



  - Truncated Unary (TU) Binarization
    - If the max number S is 5 then 5 can be binarized as 11111, the 0 in the last is not needed

Binarization (3)

- Basic binarization schemes (2)
  - Fix Length (FL) binarization
    - N is to the Lth power of 2: log2N
    - N is not to the Lth power of 2: example
  - Kth order Exp-Golomb Binarization (EGB)
    - A combination of unary binarization and fix length binarization
    - Refer to Prof. Li slides for more information

Binarization (4)

- Concatenation of basic binarization scheme
  - Coded block pattern: 4-bit FL prefix and a TU suffix with max number S equal to 2
  - TU prefix and Kth order Exp-Golomb Binarization
    - Absolute motion vector difference: TU prefix with max value of 9 and 3-order Exp-Golomb
    - Transform coefficient levels: TU prefix with max value of 14 and 0-order Exp-Golomb

Binarization (5)

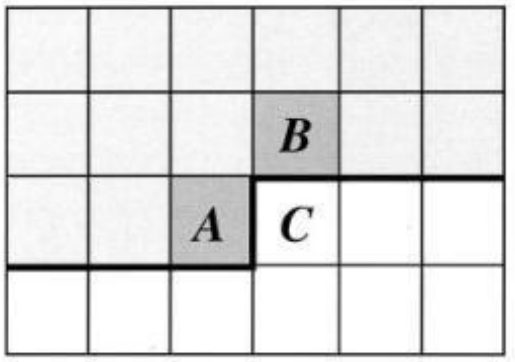
- After binarization, all the symbols are converted into bin string
  - Reference index 3: 111 (TU)
- For the binary symbol, the binarization process is ignored (skip flag)
- Then for each bin, we will choose a suitable context model and finish the arithmetic coding

Context modeling (1)

- What is context modeling
  - Context modeling is used to estimate an accurate possibility of each bin
  - Context
    - The coded results of the previous blocks
    - The coded results of the previous bins
    - The position of the current block
  - Different context models mean different possibilities for various bins

Context modeling (2)

- Context modeling type (1)
  - Just one context model to represent the possibility (maintain only one possibility distribution)
- Context modeling type (2)
  - the neighboring element to the left and on top of the current syntax element (3 possibility distributions
  - Examples for Skip flag



Context modeling (3)

- Context modeling type (3)
  - The values of prior coded bins are used for the choice of a model for a given bin

- Context modeling type (4)
  - The position in the scanning path
  - The accumulated number of encoded levels

Context modeling (4)

- Context model initialization (initial possibility distribution)
  - Based on two parameters (Slice type and Quantization parameters)
  - The initialized context model is based on some training data
  - According to the AVS2 (audio video standard), the initialized context model will not lead to significant BD-rate difference

Arithmetic Coding (1)

- Arithmetic coding
  - Update the Low Bound and Range

$$HIGH \leftarrow LOW + RANGE \times CDF(n)$$

$$LOW \leftarrow LOW + RANGE \times CDF(n-1)$$

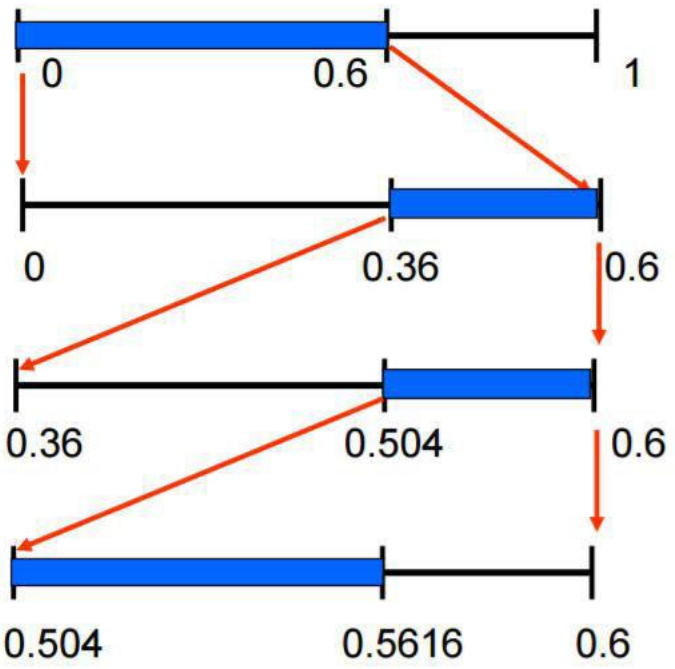
LOW = 0, HIGH = 1

LOW = 0, HIGH = 0.6

LOW = 0.36, HIGH = 0.6

LOW = 0.504, HIGH = 0.6

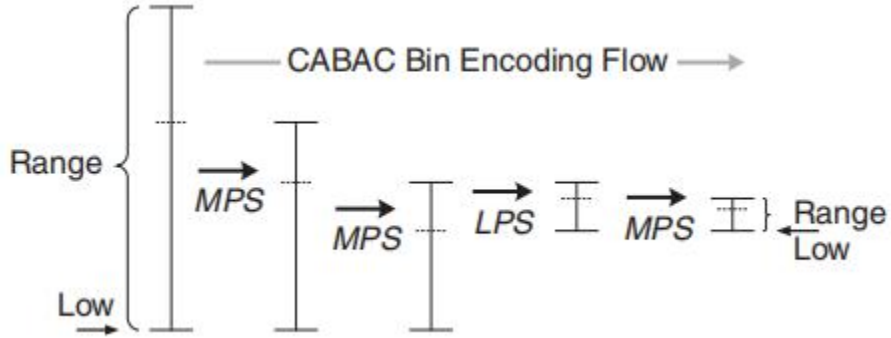
LOW = 0.504, HIGH = 0.5616



Bin Arithmetic Coding (2)

- Multiplication operation:

$$R_{LPS} = R \times P_{LPS}$$



MPS: Most Prob Symbol
LPS: Least Prob Symbol
R: Range
$P_{LPS} = 1 - P_{MPS}$

- Multiplication is replaced with a lookup table implementation
- R is quantized with 4 values only when calculating the R_{LPS}
  - p_{LPS} is quantized with 64 values
  - A look-up table with 64 by 4 entries will be used to replace the multiplication operation

Arithmetic Coding (3)

- R is quantized with 4 values
  - $2^8 <= R < 2^9$ (corresponding to [0.5,1))
  - The 4 parts: [256, 320], [320, 384], [384, 448], [448, 512] (ρ=**0,1,2,3**)
  - Quantization: replaced using the median value of the zone: **288, 352, 416, 480**

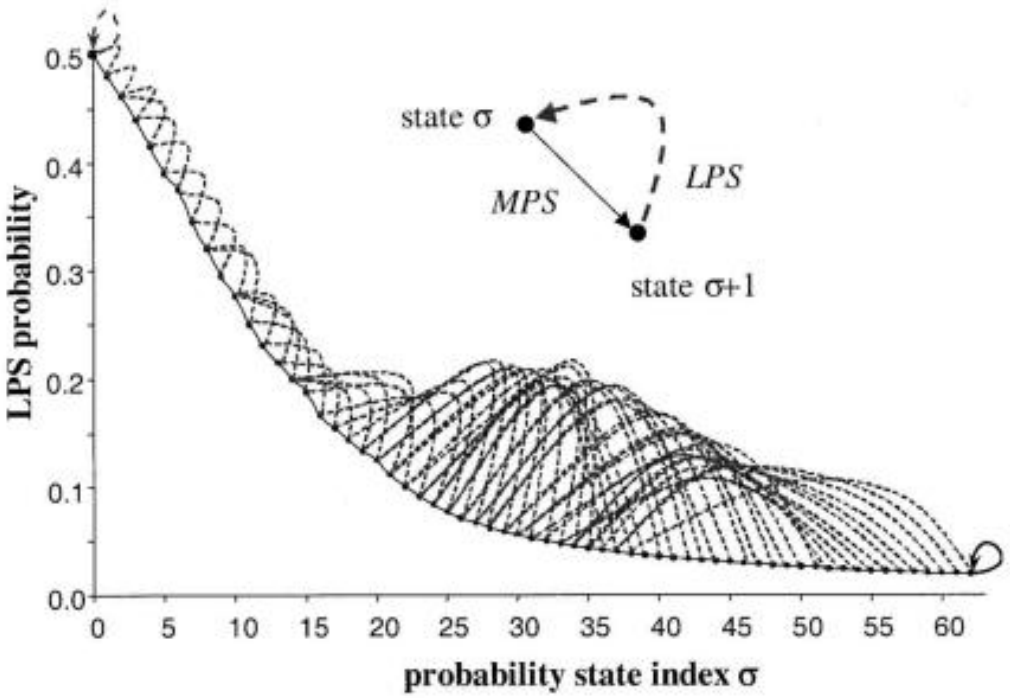
- P_{LPS} is quantized using 64 values
  - P_{LPS} is within [0.01875, 0.5]
  - $P_\sigma = \alpha \times P_{\sigma-1}$ (α is approximated as 0.95)

$$\alpha = \left(\frac{0.01875}{0.5}\right)^{1/63}$$

Arithmetic Coding (4)

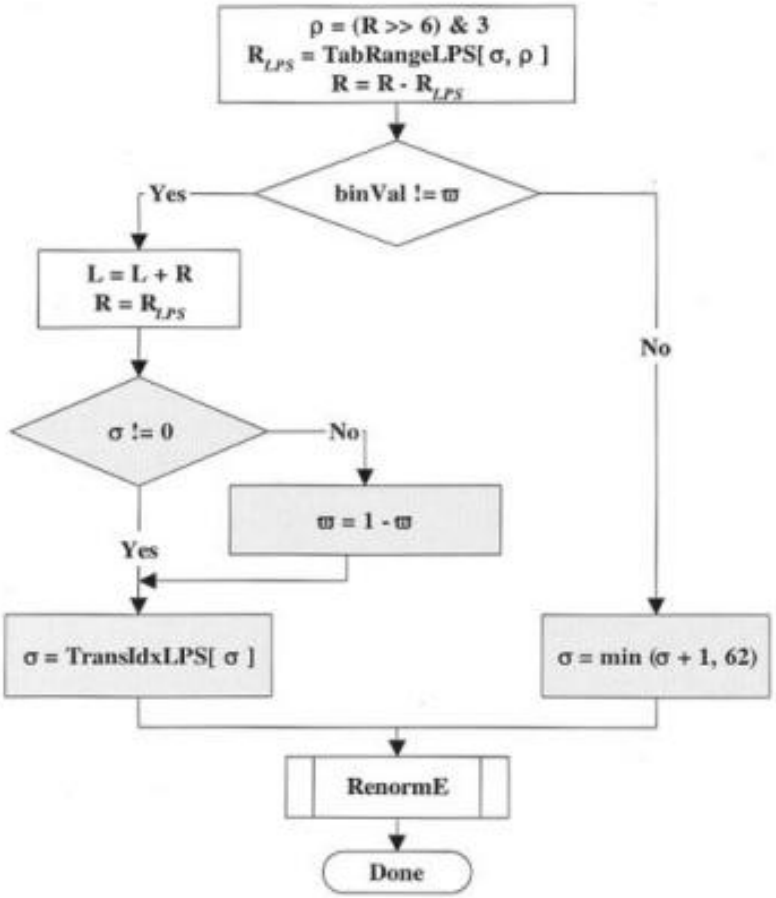
- Prob update table

$$p_{\text{new}} = \begin{cases} \max(\alpha \cdot p_{\text{old}}, p_{62}), & \text{if a MPS occurs} \\ \alpha \cdot p_{\text{old}} + (1-\alpha), & \text{if a LPS occurs} \end{cases}$$

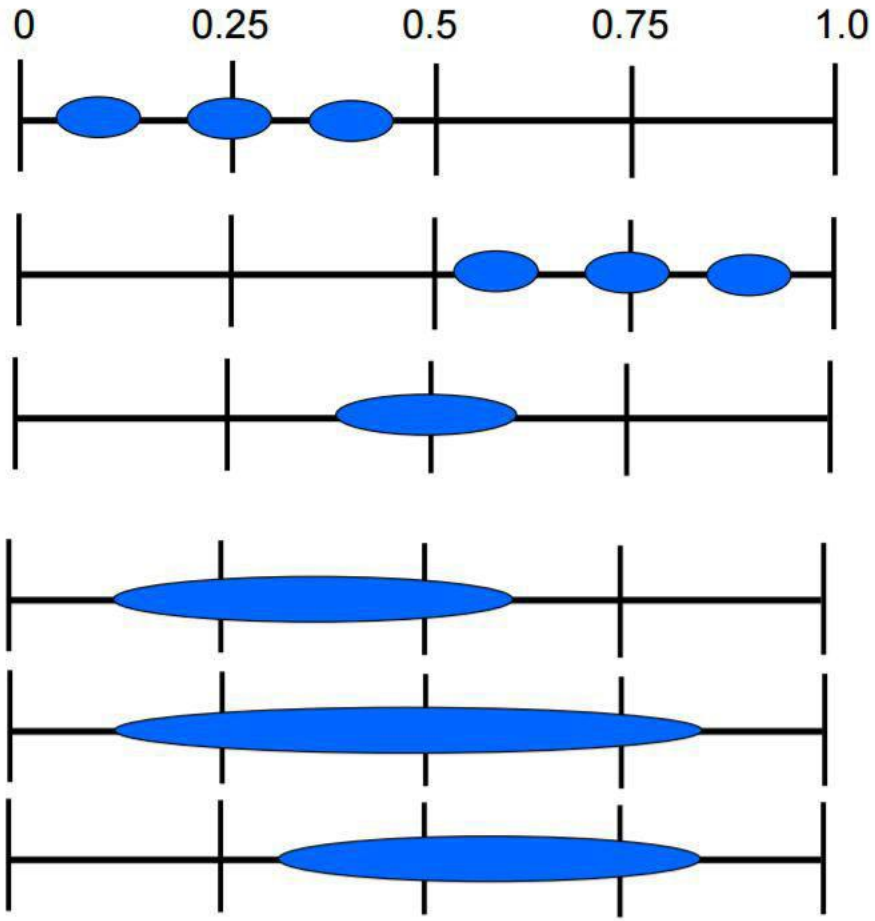


Arithmetic Coding (5)

- Encoding process
  - Calculate R_{LPS} according to the look-up table
  - If the current bin is MPS, update R and possibility model
  - If the current bin is LPS, update R, Lower Bound, and possibility model
  - Renorm Process is used to make sure R is in a suitable range ($2^8 <= R < 2^9$)
    - Refer to the three scalar operations in the slides provided by E1, E2, E3 conditions

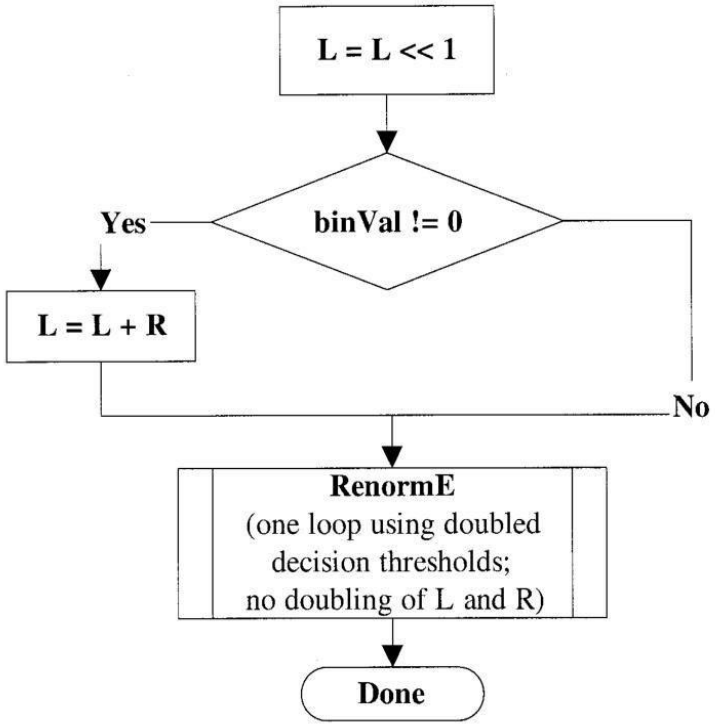


Scaling


0
0.25
0.5
0.75
1.0
Need E1 scaling
Need E2 scaling
Need E3 scaling
No scaling is required.
Continue to
encode/decode the next
symbol.

Arithmetic coding - Bypass Logic

- Bypass coding (No need to update context model)
  - Both the possibilities of LPS and MPS are 0.5



Outline

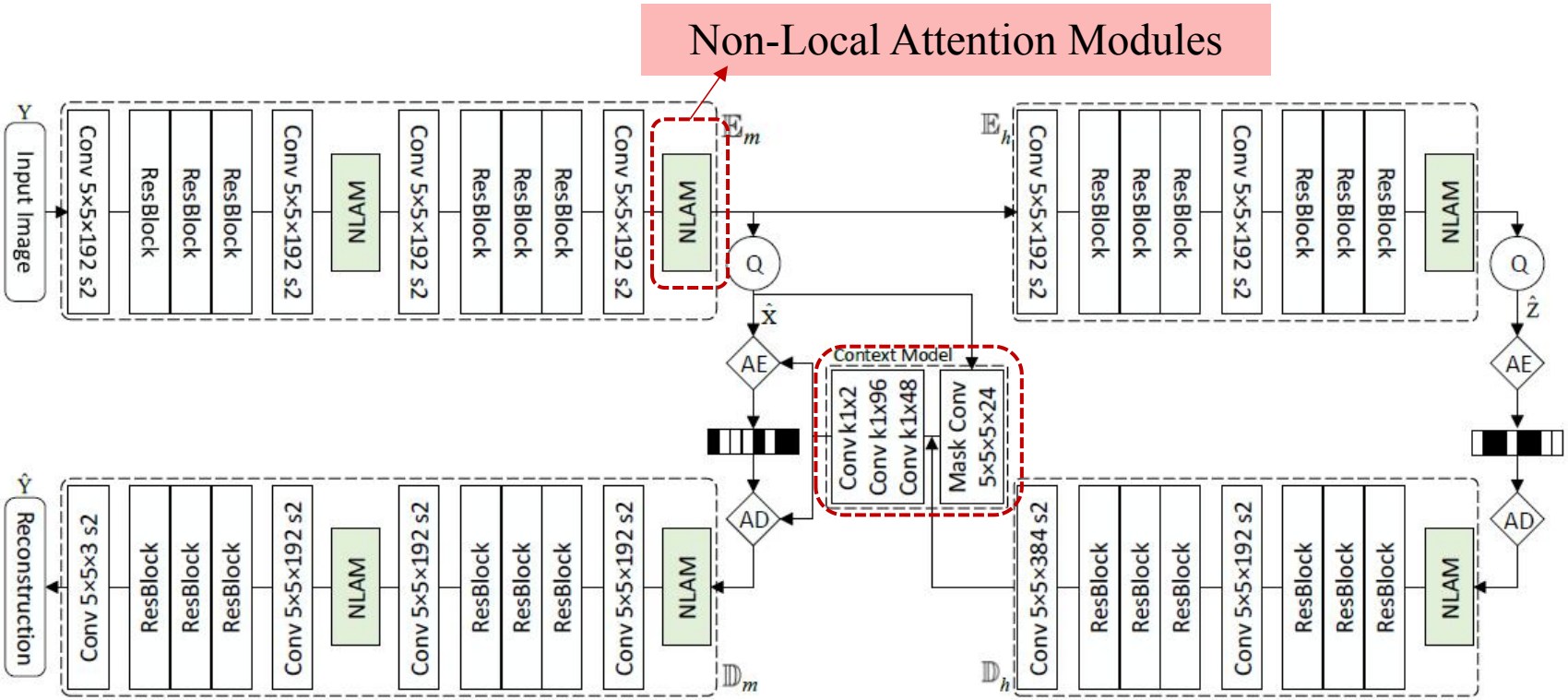
- Lecture 05 Re-Cap
- Context Adaptive Arithmetic Coding
  - PAQ
  - CABAC in H264
  - Context model in deep learning compression

Neural Image Compression via Non-Local Attention Optimization and Improved Context Modeling
Tong Chen, Zhan Ma, et.al.

https://arxiv.org/pdf/1904.09757.pdf

Introduction - NIC

- A variational autoencoder with
  - Non-local operations
  - Attention mechanism
  - Context model for spatial-channel correlations



Non-Local Attention Modules (NLAM)

❑Non-local Network

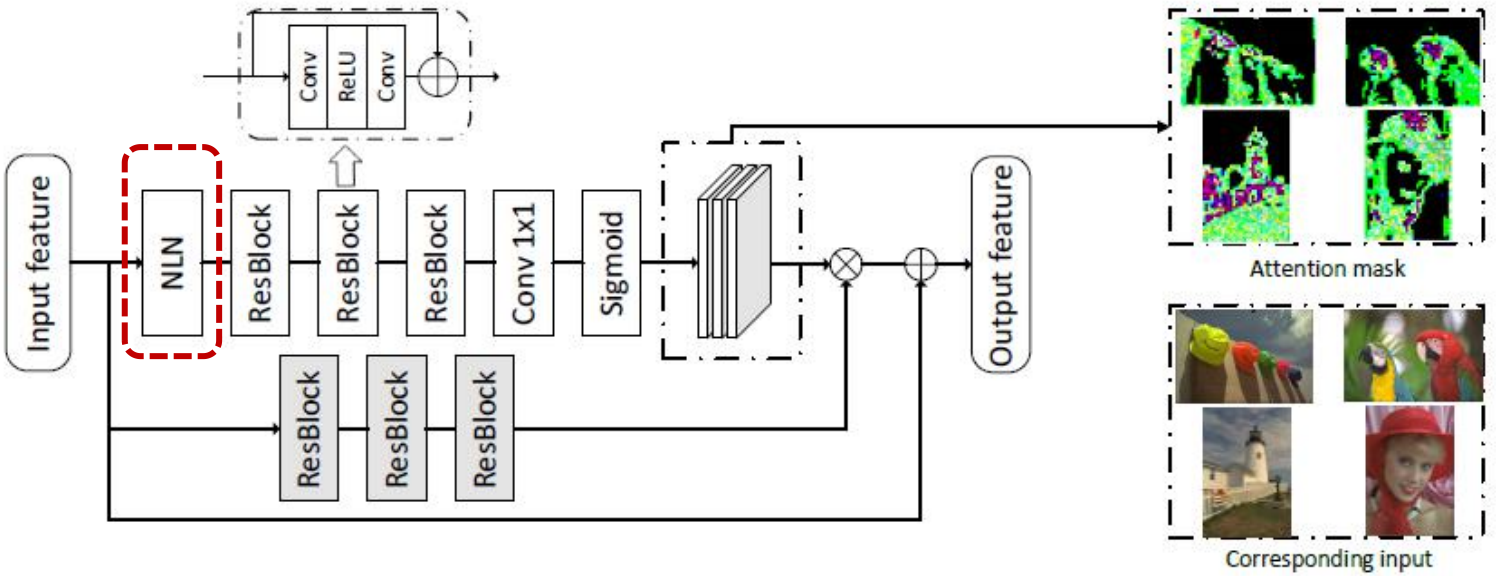
- This NLN computes the output at *i*-th position, Y*i*, using a weighted average of the transformed feature values of input X, as below:

$$\mathbf{Y}_i = \frac{1}{C(\mathbf{X})} \sum_{\forall j} f(\mathbf{X}_i, \mathbf{X}_j) g(\mathbf{X}_j), \qquad \text{(X and Y share the same size)}$$

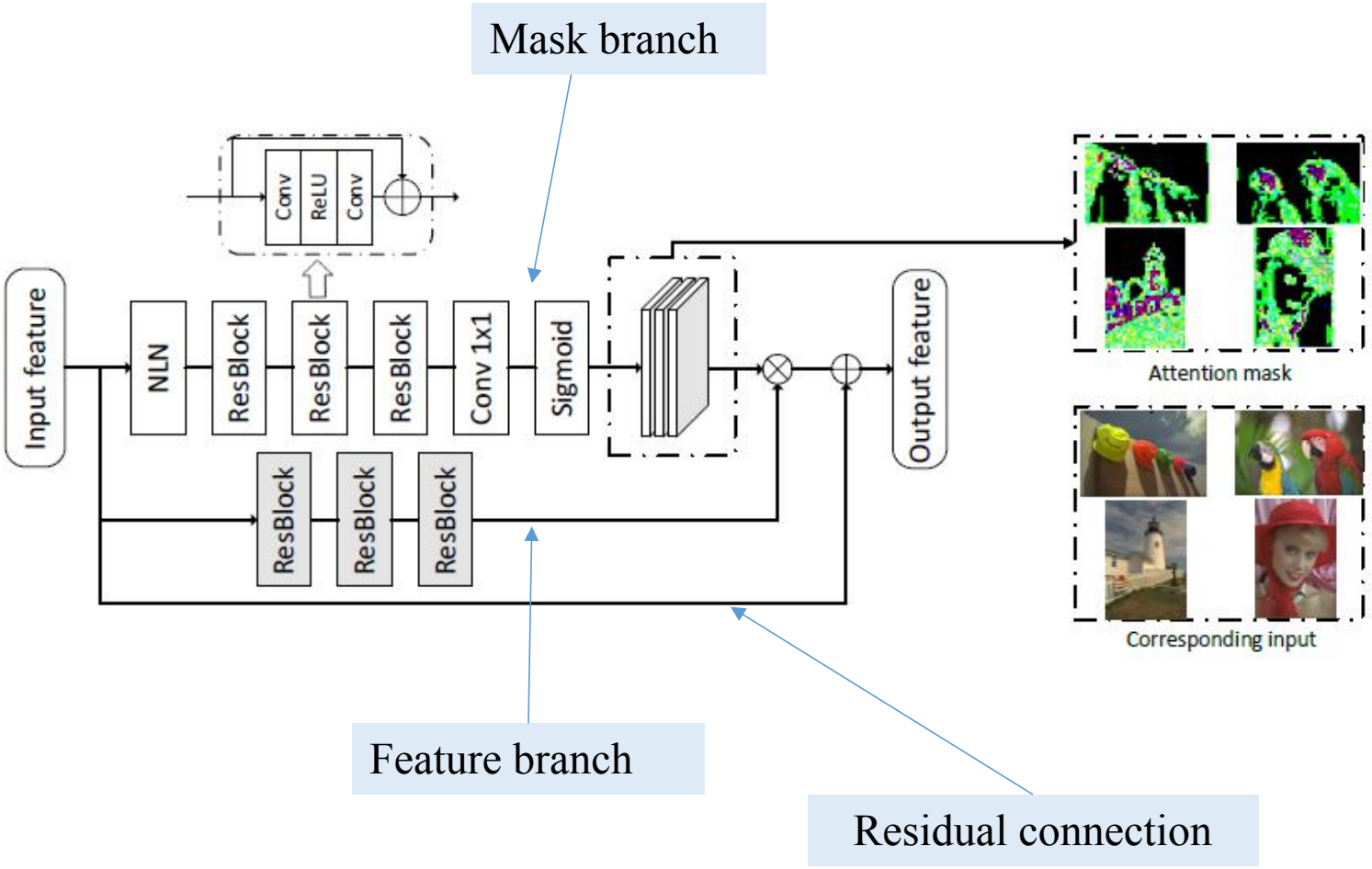
function f(.) computes the correlations between X*i* and X*j*

function g(.) derives the representation of the input at the position j

C(X) is a normalization factor: $C(\mathbf{X}) = \sum_{\forall i} f(\mathbf{X}_i, \bar{\mathbf{X}}_j)$



Non-Local Attention Modules (NLAM)



Non-Local Attention Modules (NLAM)

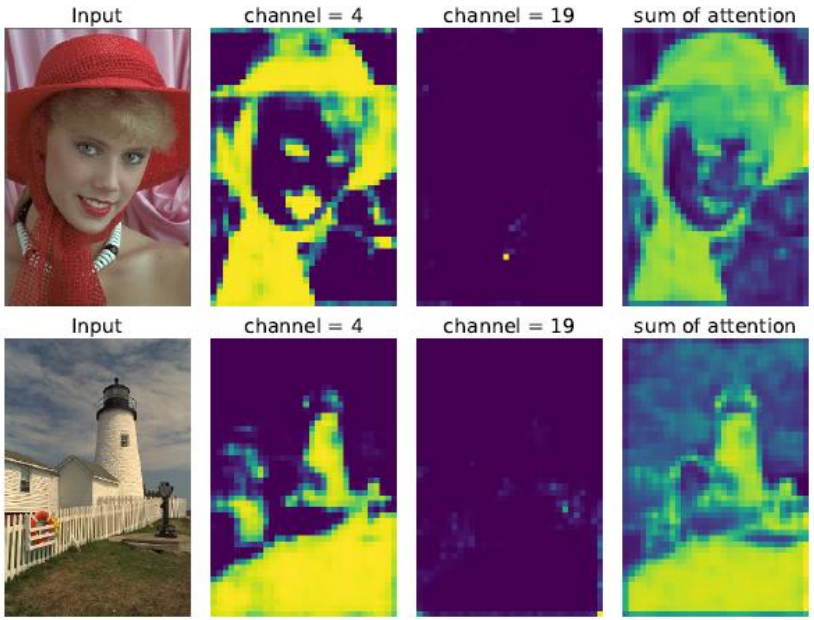
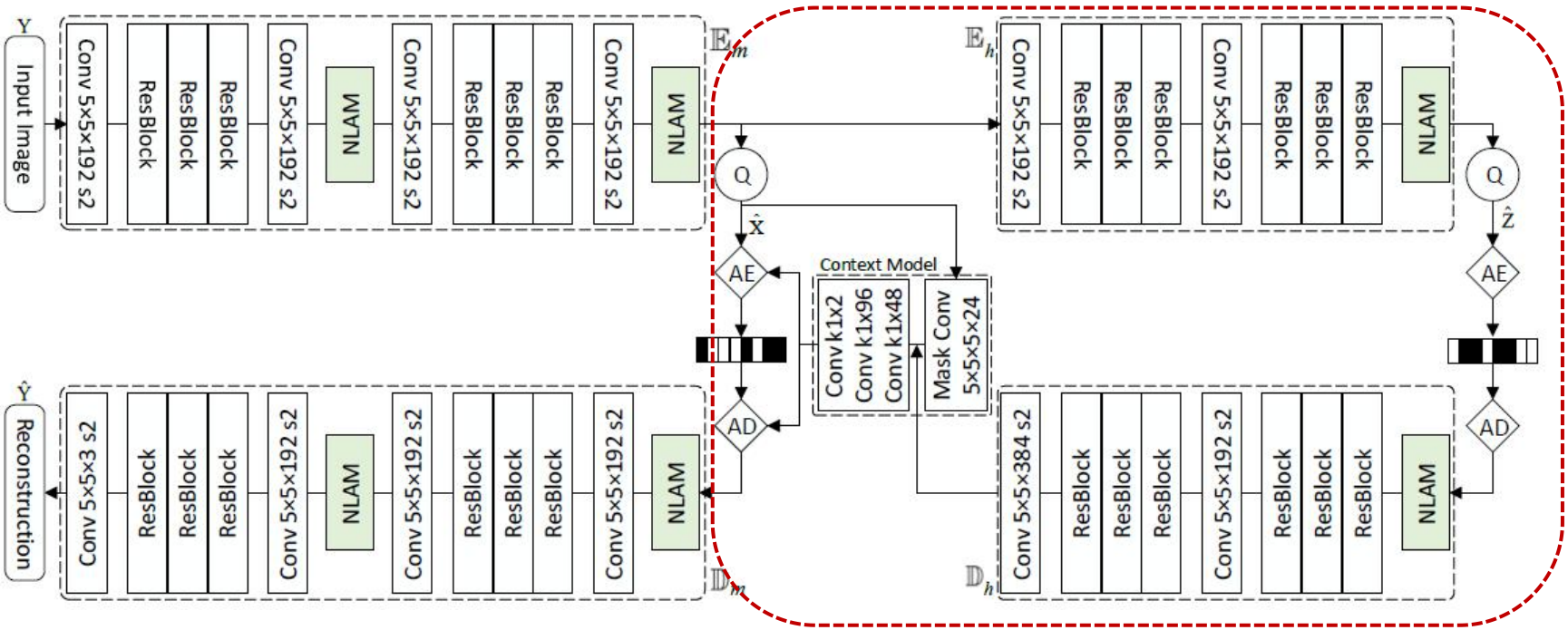


Fig. 2: **Visualization of attention masks generated by NLAM.** Brighter means more attention. Attention masks have the same size as latent features with several channels. Here channel 4 and 19 are picked to show the spatial and channel attention mechanism (values are in the range of $(0, 1)$). Term "sum of attention" denotes the accumulated attention maps over all channels.

Conditional Entropy Rate Modeling

- Context Modeling Using Joint Autoregressive Spatial-Channel Neighbors and Hyperpriors
  - Local image neighbors usually present high correlations



For quantized latent features $\hat{\mathbf{x}}$, each element $\hat{\mathbf{x}}_i$ can be modeled as a conditional Gaussian distribution as:

$$p_{\hat{\mathbf{x}}|\hat{\mathbf{z}}}(\hat{\mathbf{x}}|\hat{\mathbf{z}}) = \prod_i (\mathcal{N}(\mu_i, \sigma_i^2) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}))(\hat{\mathbf{x}}_i),$$

Hyperprior only

$$p_{\hat{\mathbf{x}}}(\hat{\mathbf{x}}_i|\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_{i-1}, \hat{\mathbf{z}}) = \prod_i (\mathcal{N}(\mu_i, {\sigma_i}^2) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}))(\hat{\mathbf{x}}_i),$$

Conditional Entropy Rate Modeling

- Context Modeling Using Joint Autoregressive Spatial-Channel Neighbors and Hyperpriors
  - Local image neighbors usually present high correlations

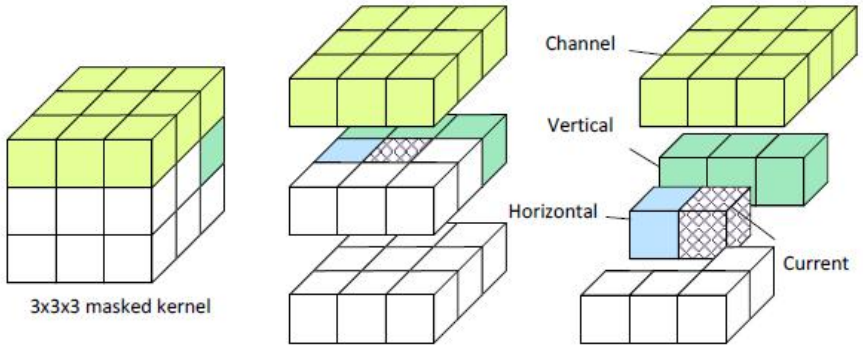
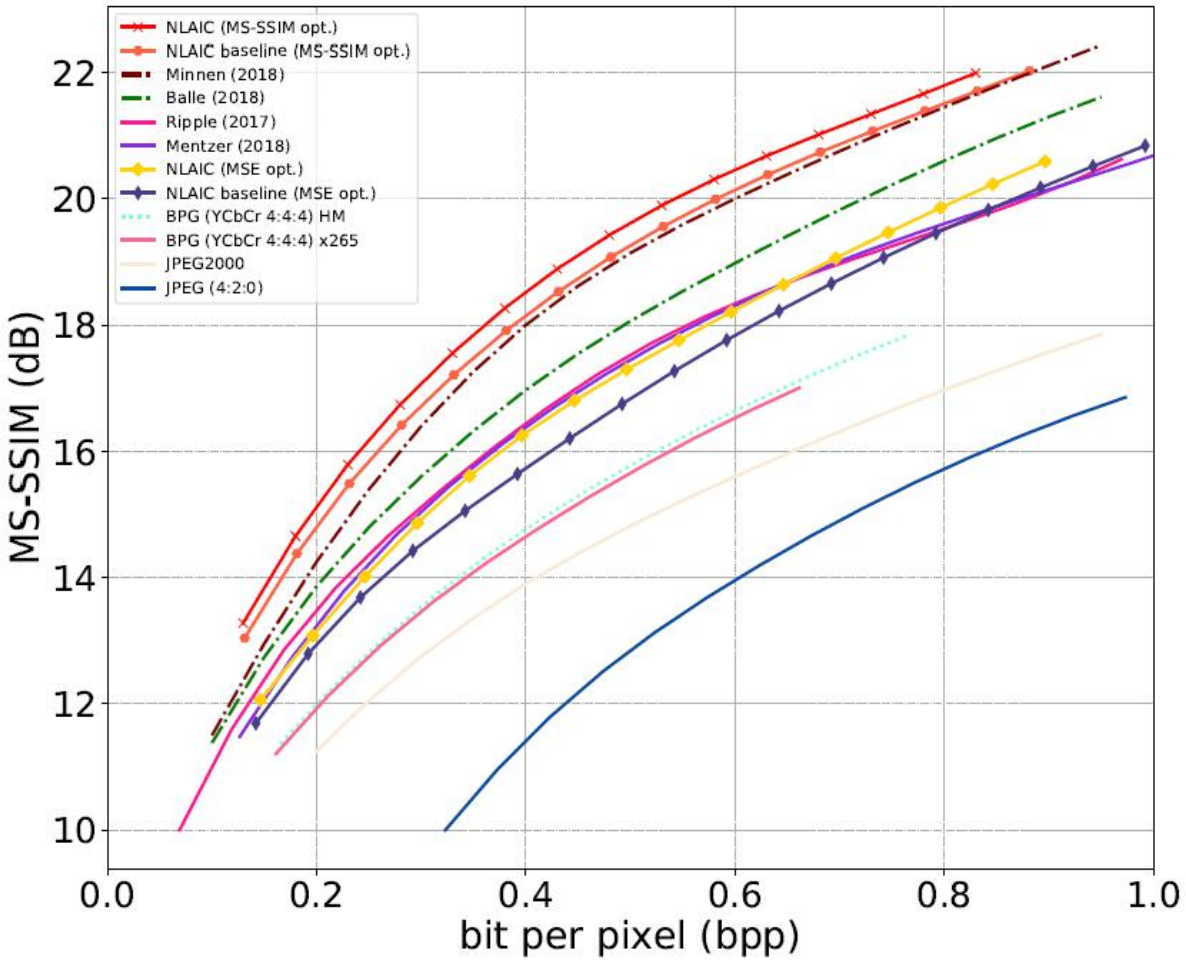


Fig. 3: **3D Masked Convolution.** A 3×3×3 masked convolution exemplified for exploring contexts of spatial and channel neighbors jointly. Current pixel (in purple grid cube) is predicted by the causal/processed pixels (in yellow for neighbors from previous channel, green for vertical neighbors, and blue for horizontal neighbors) in a 3D space. Those unprocessed pixels (in white cube) and the current pixel are masked with zeros.

Experimental Results

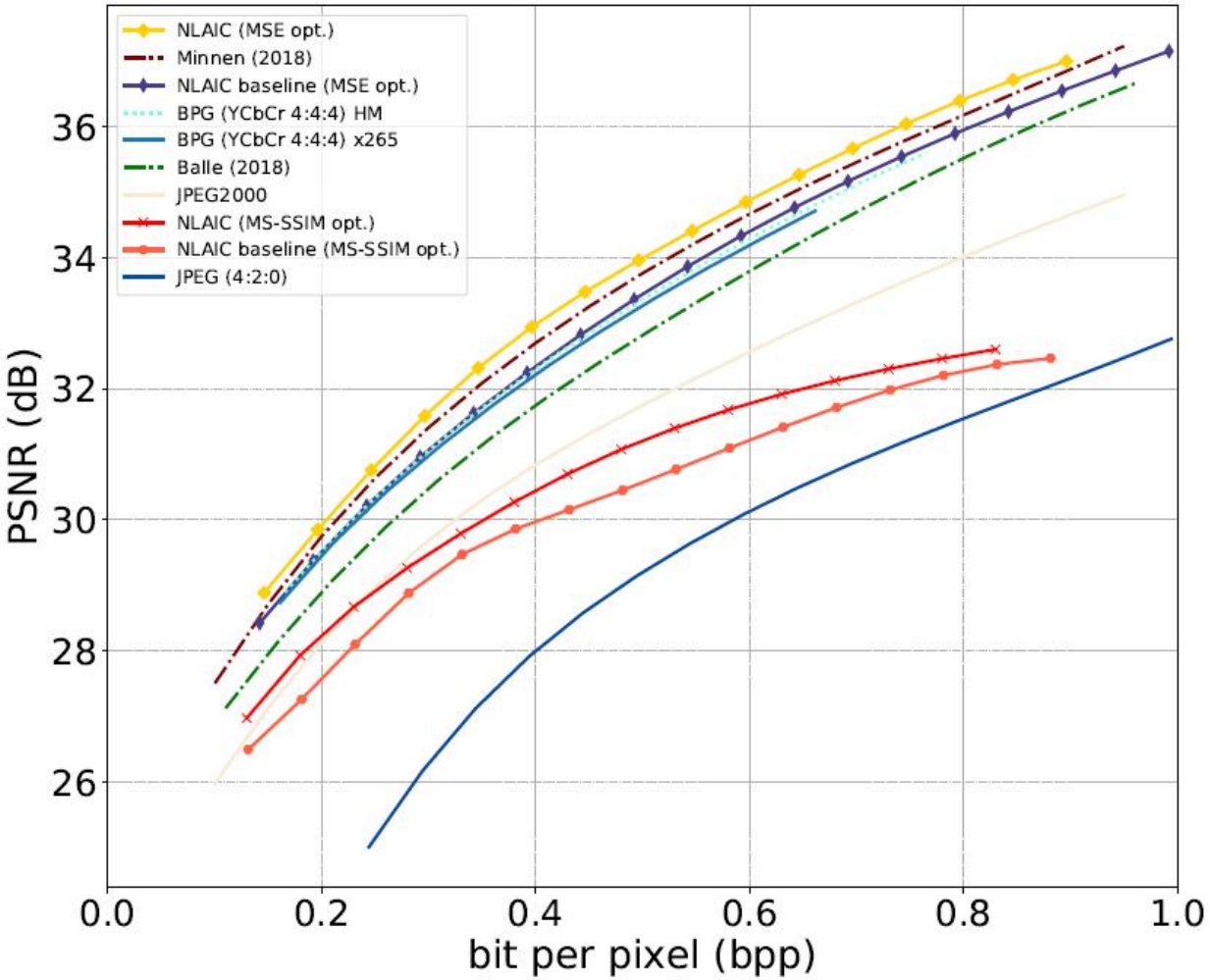
- RD performance



(a)

Experimental Results

- RD performance



(b)

Experimental Results

❑ Subjective results

Fig. 8: **Subjective Evaluation.** Visual comparison among JPEG420, BPG444, NLAIC MSE opt., MS-SSIM opt. and the original image from left to right. Our method achieves the best visual quality containing more texture without blocky nor blurring artifacts.

Summary

- Adaptive Arithmetic Coding
  - AC can be adaptive by changing the prob of 1 and 0s
  - Simple solutions PAQ1
  - Shallow neural network solutions: PAQ6 and variations
  - H264 CABAC implementation
  - Context modeling in new leanring based compression schemes