# Digital Image Processing

## Lecture 15: Lossless Image coding

IMAGE Compression

↳

Lossless: No Information is lost

(we squeeze it to the smallest it will get, then send it - No info lost)

- Archival Documents (Medical/Legal)
- Satellite Images
- Medical Images

Lossy: Slightly change Image/ Lose information to get better compression

- JPG/WEB Images

To measure performance, we can use the Compression Ratio:

$$\text{compression ratio} = \frac{\#\text{ of bits per Symbol Before}}{\#\text{ of bits per Symbol After}}$$

for real images we can get Lossless Compression Ratio in the Range 2–10

→ for images $\frac{8}{\#}$ 8 bits per Symbol

BMP - No compression → RGB values every Pixel

Gif → much smaller (Lossless compression)

(2) IMAGE Histogram — Think of as a PMF

$$p_i = \text{Prob}(I(x,y) = i)$$

(probability an image pixel has intensity $i$)

$$= \frac{\#(I(x,y) = i)}{\text{Total \# of Pixels}}$$

↑ number of times this is true

If Level $i$ is represented with $b_i$ bits

The average Bits per pixel =

$$L_{av} = \sum_{i=1}^{N} p_i \cdot b_i = E(b)$$

↖ Expected value of the number of bits

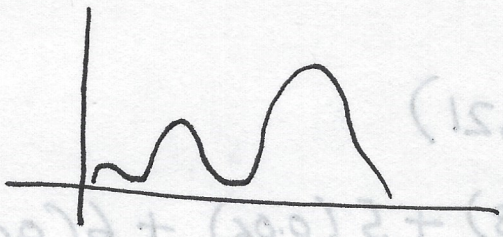
The sum over all my levels of the probability of that level multiplied by the number of bits used to store that

Uniform coding, $b_i = b$ (e.g. 8 bits)

If Every Symbol/Level $i$ coded with 8 Bits

$$L_{av} = \sum_{i=0}^{N-1} 8 p_i = 8\left(\sum p_i\right) = 8 \cdot 1 = 8$$

Say the histogram looks like,



Why use 8 bits to encode frequent and infrequent symbols?

IDEA: use fewer bits to describe more frequent symbols
(e.g. morse code)

E.

Q ——— . ——. ; - more common
; - less common

- willing to tolerate more bits to cover the latter

Q. - We have Gray levels in images
⊗ → Different number of bits for code words

Consider 8-levels

| Level | Pi | uniform code | variable code | length |
|---|---|---|---|---|
| 0 | 0.19 | 000 | 00 | 2 bits |
| 1 | 0.25 | 001 | 11 | 2 |
| 2 | 0.21 | 010 | 01 | 2 |
| 3 | 0.16 | 011 | 101 | 3 |
| 4 | 0.08 | 100 | 1001 | 4 |
| 5 | 0.06 | 101 | 10001 | 5 |
| 6 | 0.03 | 110 | 100001 | 6 |
| 7 | 0.02 | 111 | 100000 | 6 |

(4)

for this variable Length code,

$$L_{av} = 2(0.19 + 0.25 + 0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03 + 0.02)$$

$$= 2.7 \text{ bpp} \quad \leftarrow \text{used as the average Length}$$

We compressed 10% the # of bits needed to send this message.

$\frac{3}{2.7} = 1.1$ 11% better compression than if we haven't done it.

- whats the best code
- what's the fundamental limits

---

How to know what the best achievable $L_{av}$ is?

15:43

The Entropy of a Distribution $-E(\text{Log}_2(P_i))$

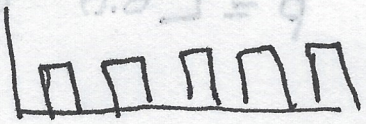
$$H = -\sum_{i=0}^{N-1} P_i \log_2 P_i$$

(negative placed so that $\log_2(N \geq 1)$ is positive)

Entropy measures how uncertain a Random variable is.

Worst case (Highest H):

Uniform Distribution (PMF)

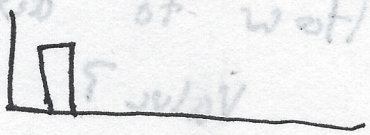


$P_i =$

$= \frac{1}{N}$

$$H = -\sum_{i=0}^{N-1} \frac{1}{N} \log_2 \frac{1}{N} = -\log_2 \frac{1}{N} = \log_2 N$$

Best case, (Lowest H) (Delta function)



$P_1 = 1$

$P_2 \ldots P_N = 0$

$H =$

$= -1 \log 1 + 0 + 0 \ldots -$

$= 0$

predictable, Entropy, H is > 0

The entropy is the lowest number of average bits per symbol That can be used to code the distribution

Our Example, $H = 2.65$

our code had $L_{AV} = 2.7$, pretty close

⑥ We find this by

$p = [0.19 \quad 0.25 \quad 0.21 \quad 0.16 \quad 0.08 \quad 0.06 \quad 0.03 \quad 0.02]$;

$\text{Sum}(p) = 1$

$H = -\text{sum}(p.*\log^2(p))$

↳ best we can possibly do

How to design a code that Approaches this Value?

Huffman Coding: optimal (minimum Lav)

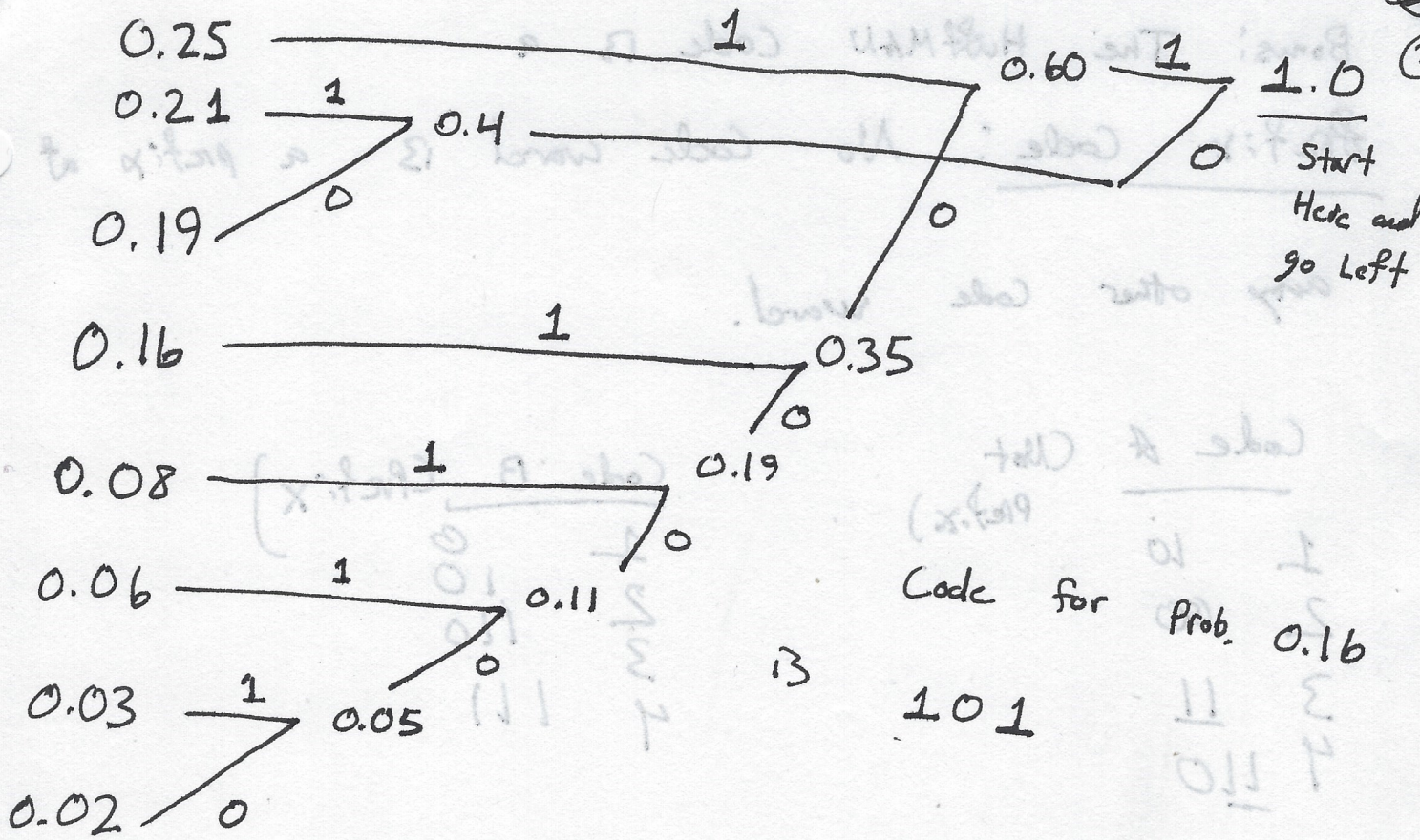
for Independently coding N symbols.

Algorithm

1) arrange Symbols in Decreasing $P_i$ (Probability)
(Think of as Leaves of a tree.)

2) Merge 2 Leaves with Lowest Prob
Assign 0 to Bottom Branch
1 to Top Branch
Repeat until finished

3) Read code from Root to Leaf.




Code for Prob. 0.16

13

101

— we had a tie at 0.19, so go either way it really doesn't matter.

| Probs | and | Code words |
|---|---|---|
| 0.25 | — | 11 |
| 0.21 | — | 01 |
| 0.19 | — | 00 |
| 0.16 | — | 101 |
| 0.08 | — | 1001 |
| 0.06 | — | 10001 |
| 0.03 | — | 100001 |
| 0.02 | — | 100000 |

the huffman code is the best we can do.

Entropy is best case scenario and Huffman

---

2710

01 100000 11 00

(8)

Bonus: The HUFFMAN Code is a

Prefix Code: No Code word is a prefix of any other Code word.

| Code A | (Not Prefix) |
|---|---|
| 1 | 10 |
| 2 | 00 |
| 3 | 11 |
| 4 | 110 |

| Code B | (Prefix) |
|---|---|
| 1 | 0 |
| 2 | 10 |
| 3 | 110 |
| 4 | 111 |

3 begins with 2 Symbols from 4

Code 1234321

A: 1000 1111 0110010

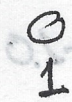
B: 0|10|110|111|10|0

↑ Prefix Code can be de-coded instantaneously.

← we need to wait when 1110 till the 0 to decode

→ only one way to decode it.

CAN Generally Do Better By Considering Pairs of Symbols (or more). —




00 | 11 | 01 | 10 | 10 | 11 | 00 | 01111000 00

| | |
|---|---|
| 00 | $P_1$ |
| 01 | $P_2$ |
| 10 | $P_3$ |
| 11 | $P_4$ |

| | |
|---|---|
| 000 | $P_1$ |
| 001 | $P_2$ |
| ⋮ | ⋮ |
| 111 | $P_8$ |

---

Truncated Huffman Coding

Lengths of codes that don't appear very often can be long.

To Avoid Extremely long codewords for infrequent Symbols

- Huffman code the most probable $k$ Symbols in the Source
- Replace the rest with a prefix + a fixed length code.

(8) (LG)

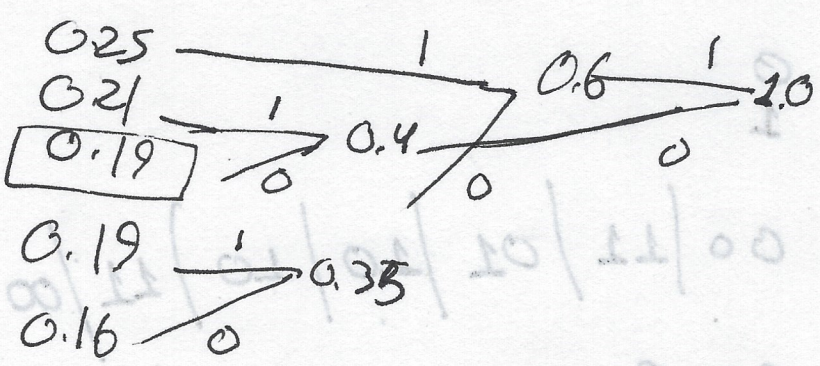
40:23

Ex

0.25
0.21
0.19
0.16
0.19

"The rest"

Not actual code word




So we have

0.25 → 11
0.21 → 01
0.19 → 00
0.19 → 101
0.16 → 100

"The Rest"

0.08 00 00
0.06 00 01
0.03 00 10
0.02 00 11

Use 2 bits to describe the rest

$L_{av} = 2.75$

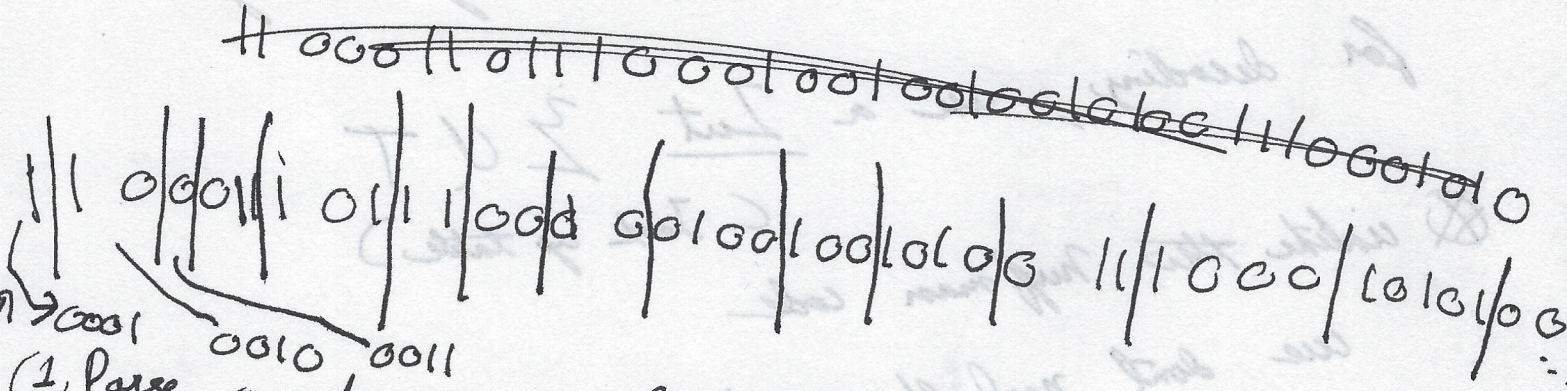
Worse than Regular Huffman but not bad

What actually happens in a zip file?
(also Tif. PNG Gif)

Lempel - Ziv - Welch (LZW) coding

- Parse source string sequentially into "~~phrases~~" phrases
  - strings that have not appeared so far
- ~~code~~ code each phrase by giving Location of prefix (fixed length code) + Extra bit

Ex




(1) Parse a phrase we haven't seen before

(2) assign each phrase as a fixed length code

our null string is $\phi$ = 000

(12) location of the prefix

Method solved around 50:25

(0000, 1)
(0001, 0)
(0008, 0)
(0013, 2)
(0010, 1)

↑ New bit

→ start at null locations, then add the 1 bit (new bit)

(See video for assistance)

not sources of the greatness of compression

→ Super effective for long binary strings

for decoding, use a Lut LUT (look up table)

⊗ unlike the huffman code we don't need the pdfs

- THICS symbol co-occurence into account

- Really pays off for very long strings

→ this algo spends time on the stuff occuring frequently rather than the infrequent stuff

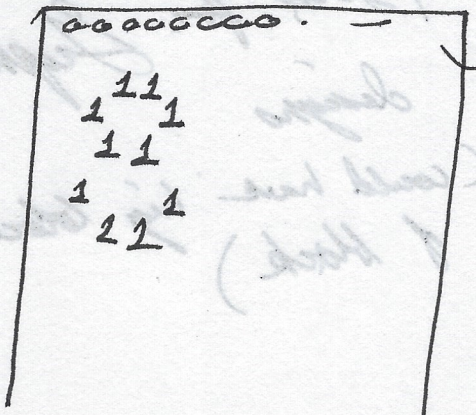
# Run-length Encoding (CRLE)

~~14~~ 13

Specific types of images are very structured

for of a page for is binary

we have zeros until we run into 1s that make up letters and numbers.



→ Image made of zeros.

IDEA: just tell us the number of zeros before a 1.

Simple either

- ~~Lengths~~ Code the number of 0's between successive 1's
- Code the lengths of continuous black / white Runs

⇒ Then we could Huffman code the run lengths

14

000 000 000 11 0000 0000 1111 1000000 10000 1

a) 108 00 00 64 → Good for drawings
(Typical)

b) 9 2 8 5 6 → Good for Graphic designs (could have big blocks of block)
W B W B W

recovery about bandwidths

J.if, Png, PDf, Z.p — Zossless