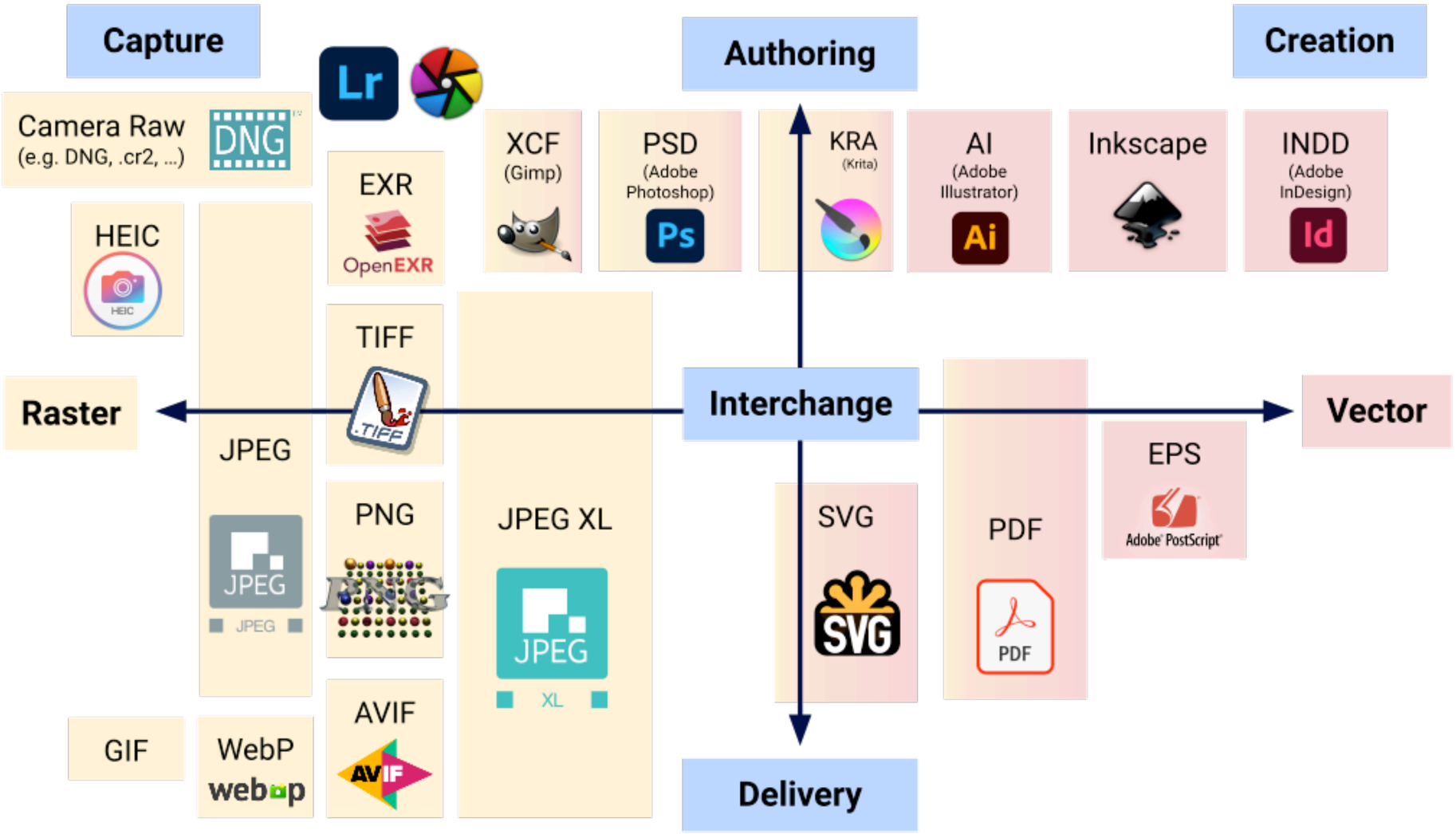
CS 5112
Algorithms for Applications

Huffman Coding Application:
JPEG Compression

How do you store an image in a file?

How do you store an image in a file?



https://en.wikipedia.org/wiki/Image_file_format

How do you store an image in a file?

What’s the simplest way?

How do you store an image in a file?

What's the simplest way?

```
3 3 3 3 3 3 3 3
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
0 5 5 5 5 5 5 0
0 5 5 5 5 5 5 0
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
2 2 2 2 2 2 2 2
```

0	000000
1	FF0000
2	00FF00
3	0000FF
4	FFFFFF
5	FFFF00
6	FF00FF
7	00FFFF
8	FF0080
9	FF8040
A	804000
B	008080
C	800000
D	800080
E	8080FF

Uncompressed bitmap

How do you store an image in a file?

What's the simplest way?

```
3 3 3 3 3 3 3 3
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
0 5 5 5 5 5 5 0
0 5 5 5 5 5 5 0
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
2 2 2 2 2 2 2 2
```

0	000000
1	FF0000
2	00FF00
3	0000FF
4	FFFFFF
5	FFFF00
6	FF00FF
7	00FFFF
8	FF0080
9	FF8040
A	804000
B	008080
C	800000
D	800080
E	8080FF

Store the values of each pixel in an array

Uncompressed bitmap

How do you store an image in a file?

What's the simplest way?

```
3 3 3 3 3 3 3 3
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
0 5 5 5 5 5 5 0
0 5 5 5 5 5 5 0
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
2 2 2 2 2 2 2 2
```

0	000000
1	FF0000
2	00FF00
3	0000FF
4	FFFFFF
5	FFFF00
6	FF00FF
7	00FFFF
8	FF0080
9	FF8040
A	804000
B	008080
C	800000
D	800080
E	8080FF

Store the values of each pixel in an array

Problem:
these are large

Uncompressed bitmap

How do you store an image in a file?

What's the simplest way?

```
3 3 3 3 3 3 3 3
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
0 5 5 5 5 5 5 0
0 5 5 5 5 5 5 0
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
2 2 2 2 2 2 2 2
```

0	000000
1	FF0000
2	00FF00
3	0000FF
4	FFFFFF
5	FFFF00
6	FF00FF
7	00FFFF
8	FF0080
9	FF8040
A	804000
B	008080
C	800000
D	800080
E	8080FF

Uncompressed bitmap

Store the values of each pixel in an array

Problem:
these are large

Solution:
use compression

Compressed Bitmaps

Take a bitmap and apply lossless compression

Lempel–Ziv–Welch (LZW)

DEFLATE
(LZ77 and Huffman coding)

Compressed Bitmaps

Take a bitmap and apply lossless compression

Lempel–Ziv–Welch (LZW)

DEFLATE
(LZ77 and Huffman coding)

JPEG Files

Joint Photographic Experts Group

Introduced in 1992

Most widely used image compression standard

“JPEG was largely responsible for the proliferation of digital images and digital photos across the Internet and later social media”

https://en.wikipedia.org/wiki/JPEG

JPEG Files

Joint Photographic Experts Group

Introduced in 1992

Most widely used image compression standard

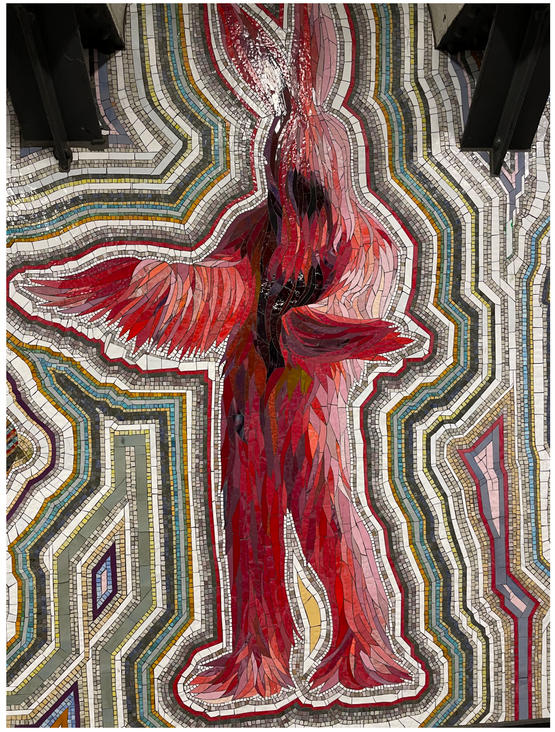
“JPEG was largely responsible for the proliferation of digital images and digital photos across the Internet and later social media”

Why?

What’s so great about JPEG?

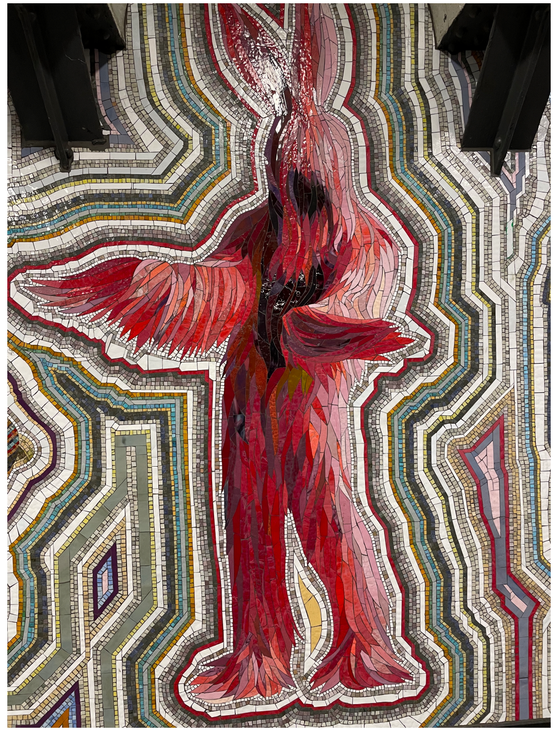
https://en.wikipedia.org/wiki/JPEG

Image File Size



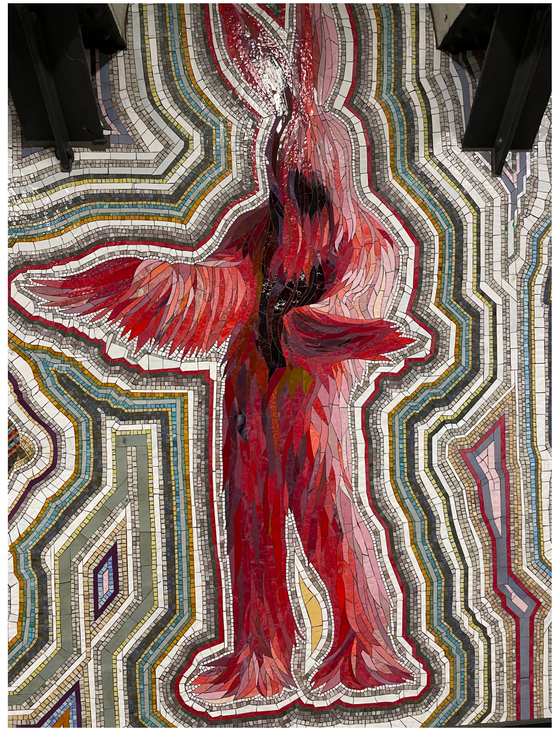
Bitmap (BMP)

37MiB



Compressed Bitmap (PNG)

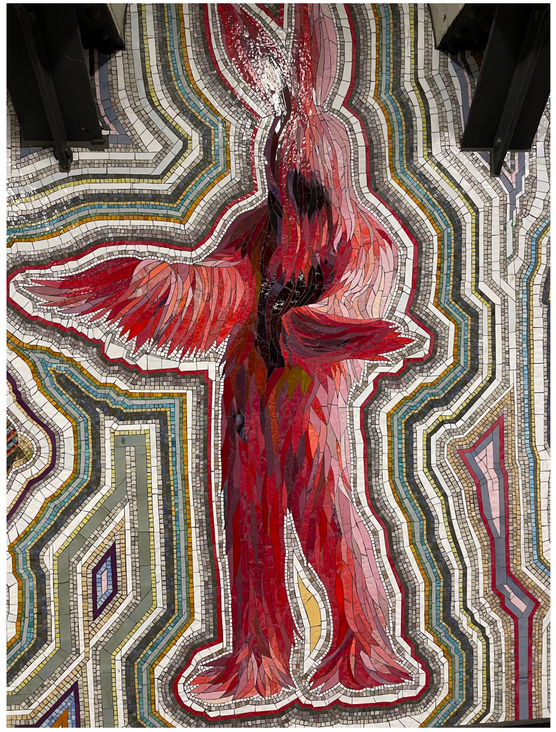
21MiB



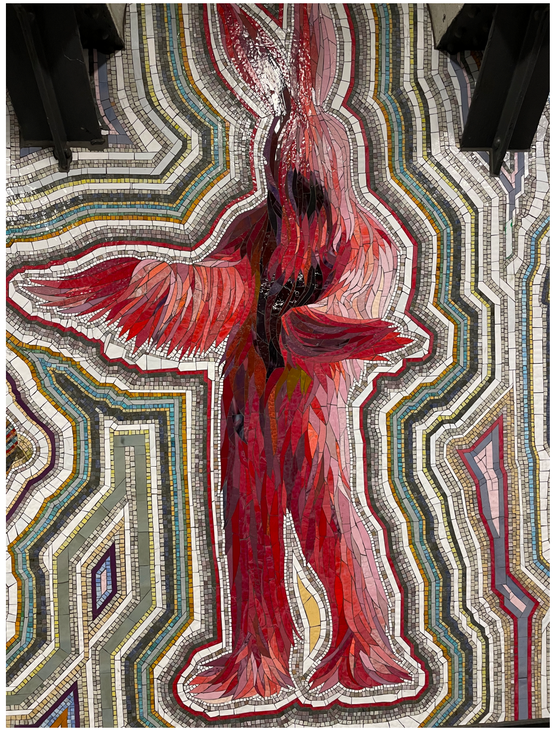
JPEG

12MiB

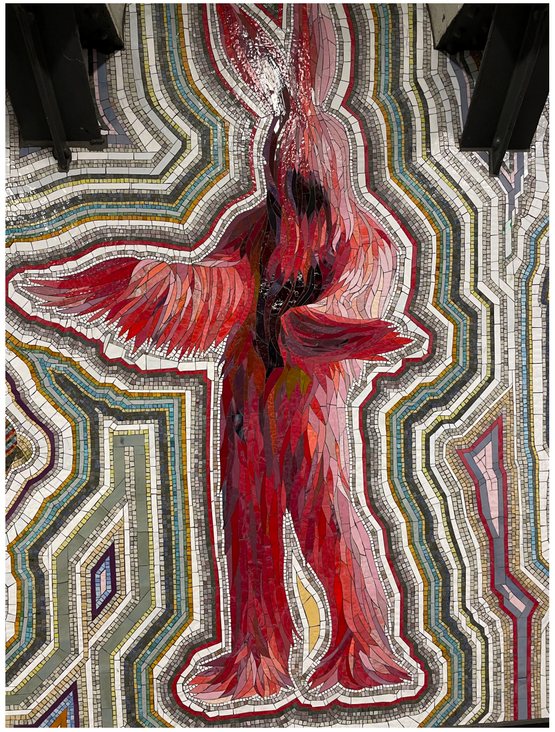
JPEG Variable Compression



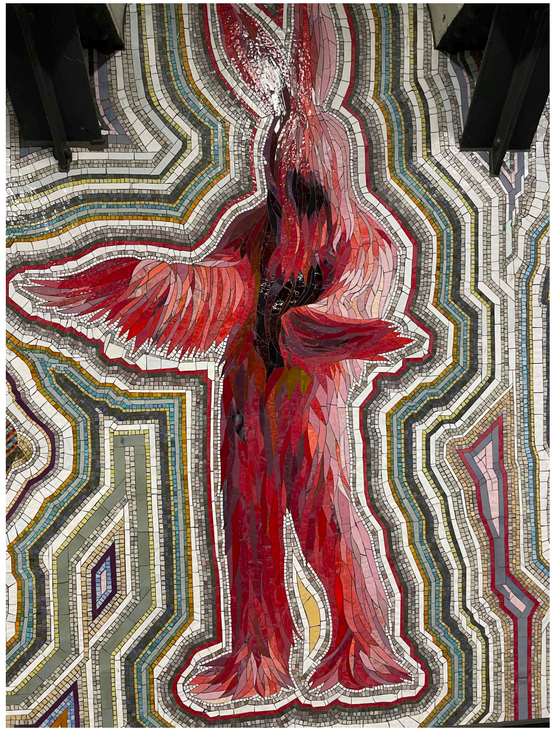
Maximum
12MiB



High
5.4MiB

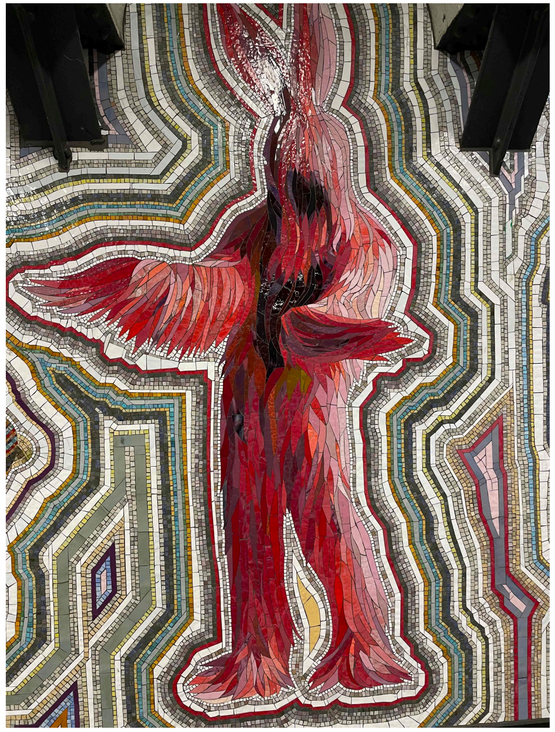


Medium
3.2MiB



Low
1.9MiB

JPEG Variable Compression



File size was/is very important
Especially in the early days of the internet

Maximum	High	Medium	Low
12MiB	5.4MiB	3.2MiB	1.9MiB

Times Square—42nd St

Each One, Every One, Equal All — Nick Cave

JPEG Compression

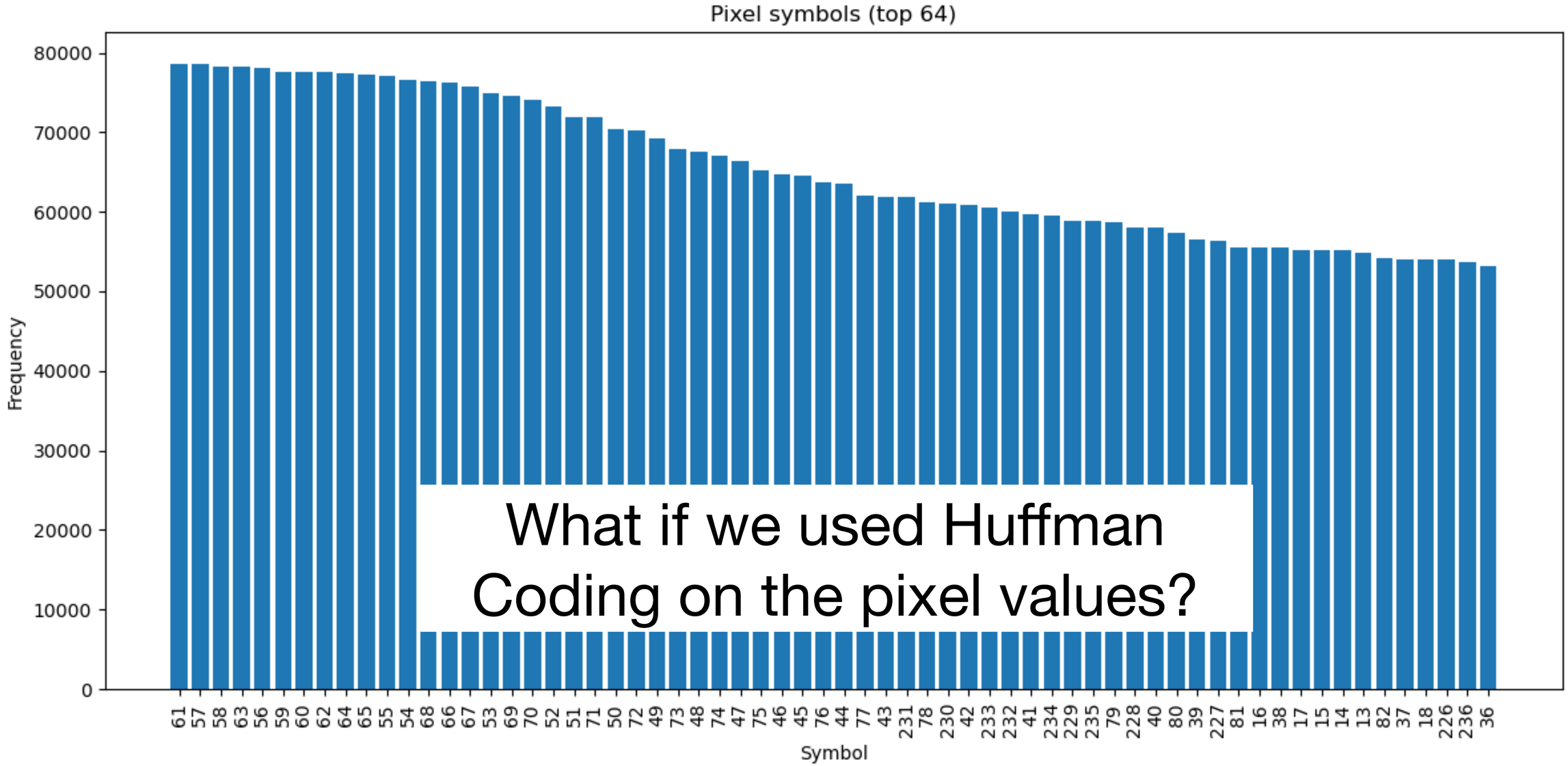
How does it work?

Why does it work so well?

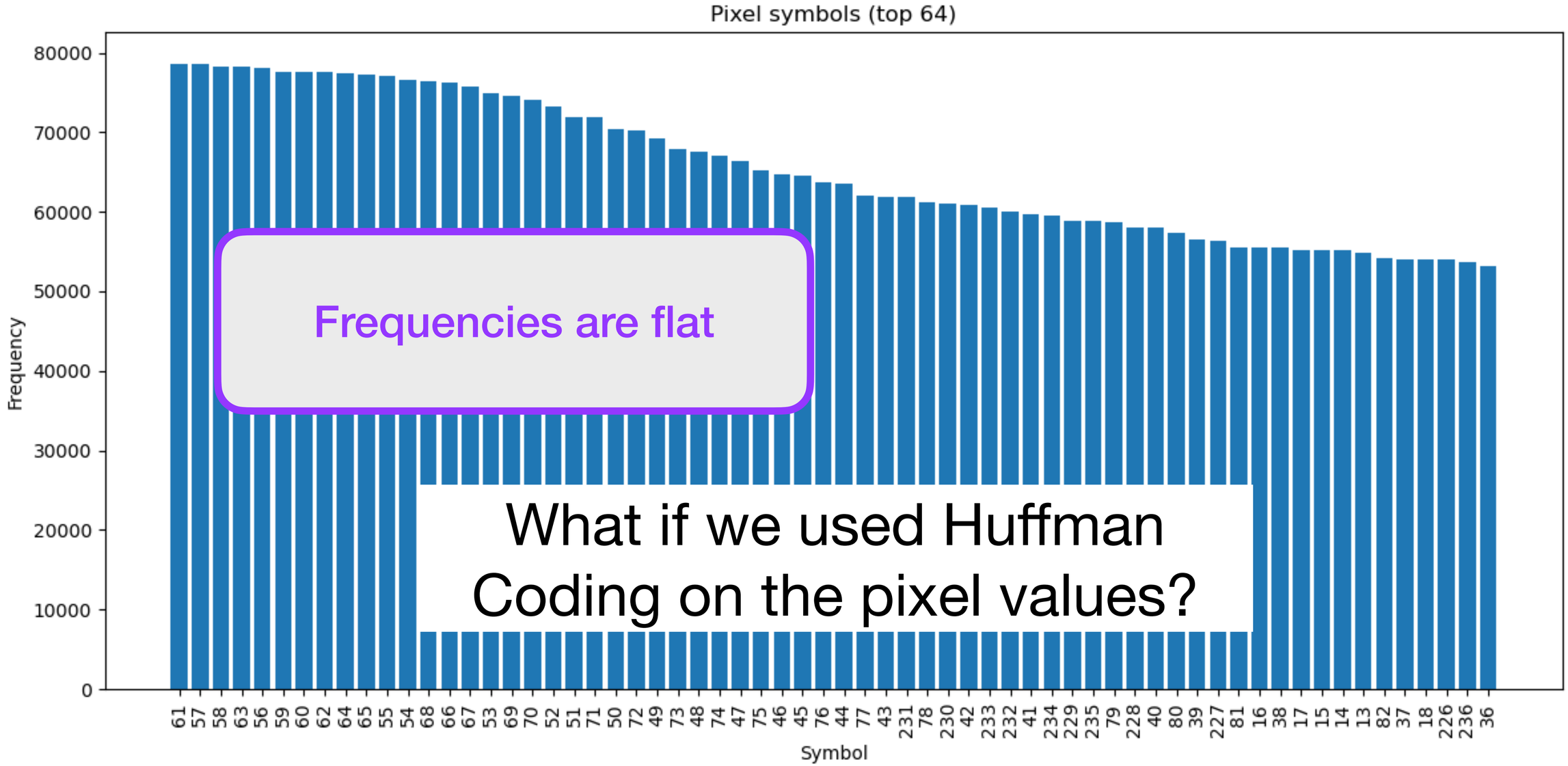
We just learned about Huffman Coding…

What if we used Huffman Coding on the pixel values?

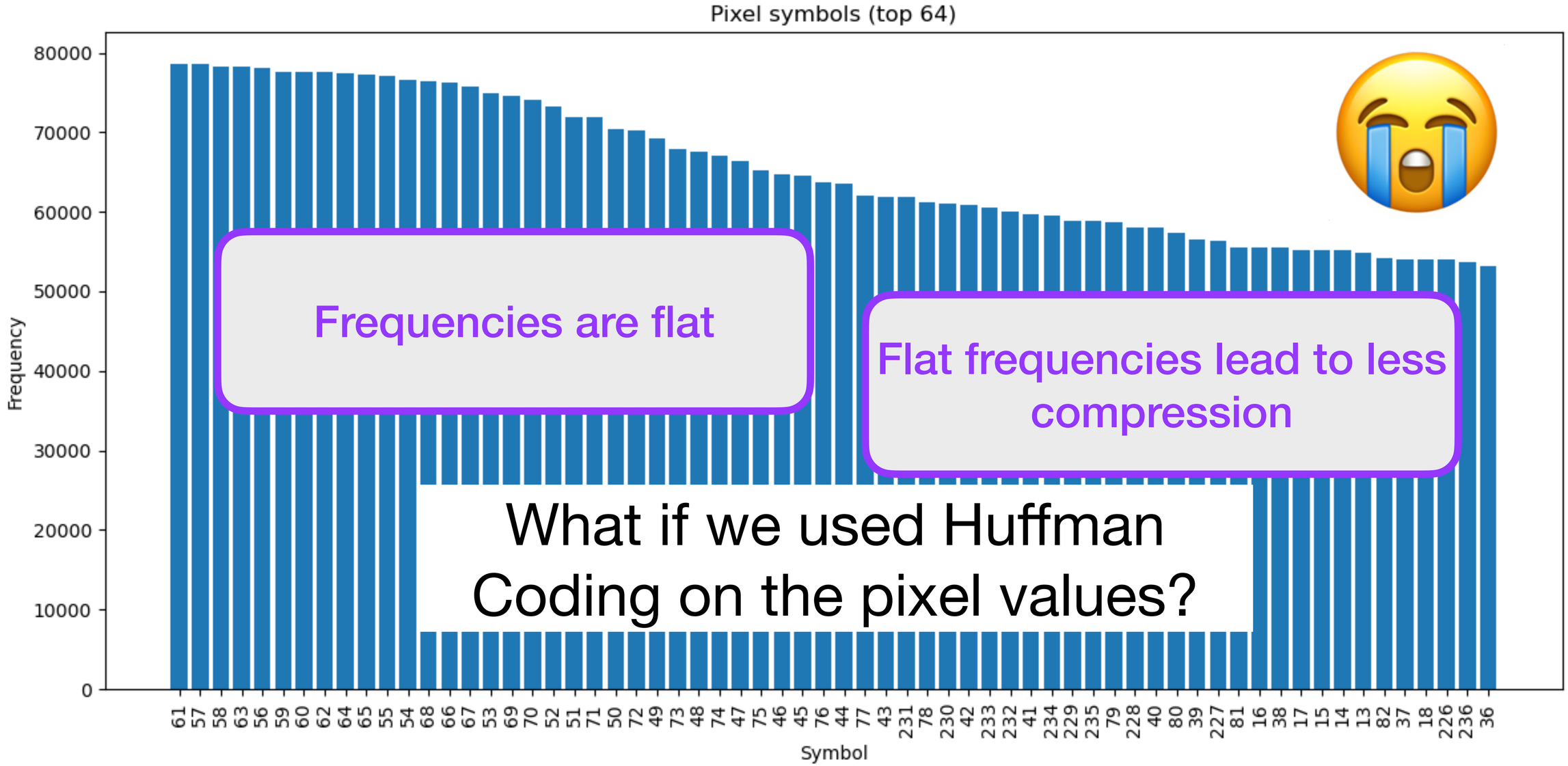
We just learned about Huffman Coding...



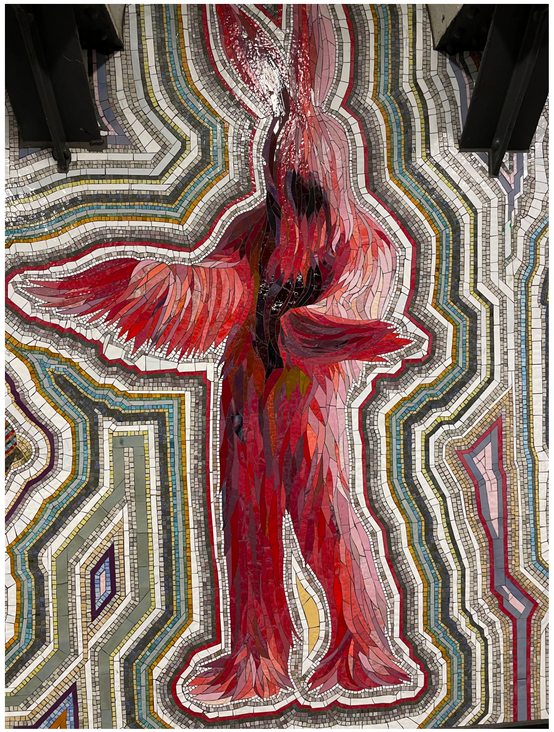
We just learned about Huffman Coding...



We just learned about Huffman Coding...



Start by separating channels



Combined

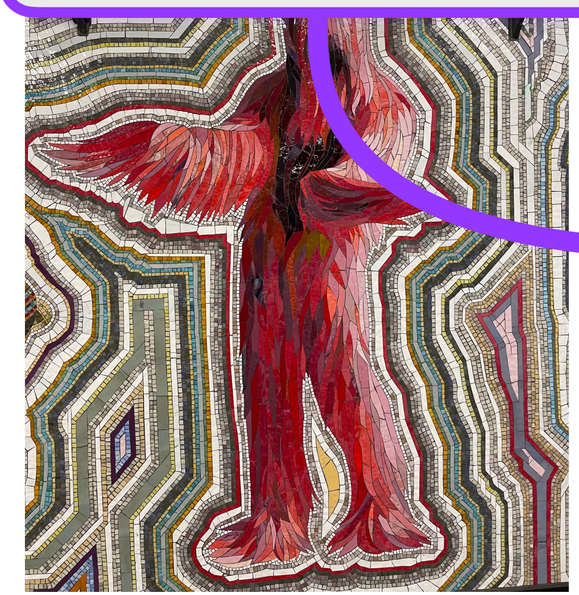
Y
Luminance

Cb
Blue difference

Cr
Red difference

Start by separating channels

Most detail in Y



Combined

Y
Luminance

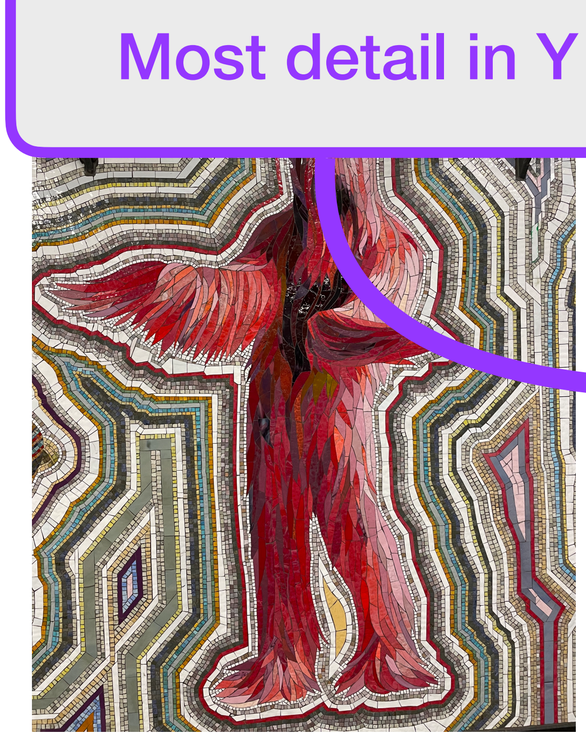
Cb
Blue difference

Cr
Red difference

Start by separating channels

Most detail in Y

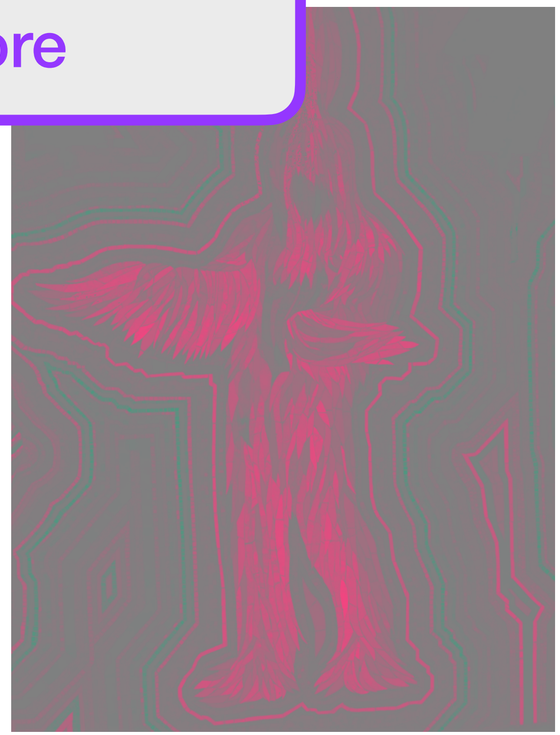
Can compress Cb and Cr more



Combined

Y
Luminance

Cb
Blue difference



Cr
Red difference

Human Perception of Color

We are much more sensitive to luminance boundaries than color boundaries

Focus on Y — Luminance

Even though this image
has lots of detail

Focus on Y — Luminance

Large areas look
“uniform”

Even though this image
has lots of detail

Focus on Y — Luminance

Large areas look “uniform”

Even though this image has lots of detail

When we zoom in, over small blocks there is a lot of uniformity

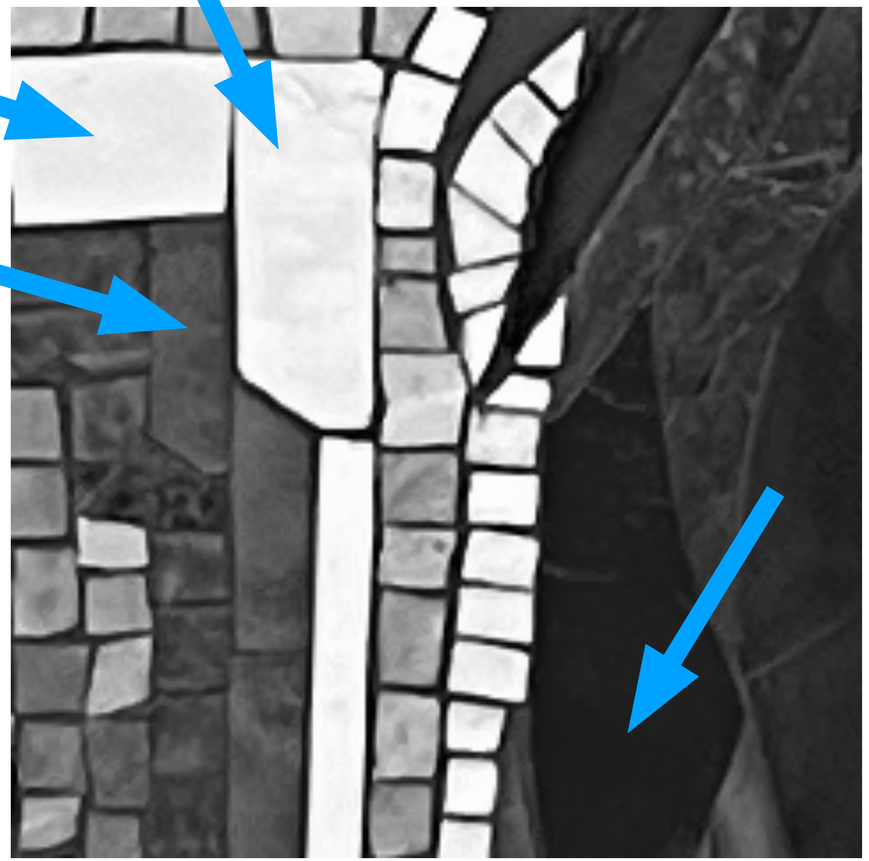
240 x 240

Focus on Y — Luminance

Large areas look “uniform”

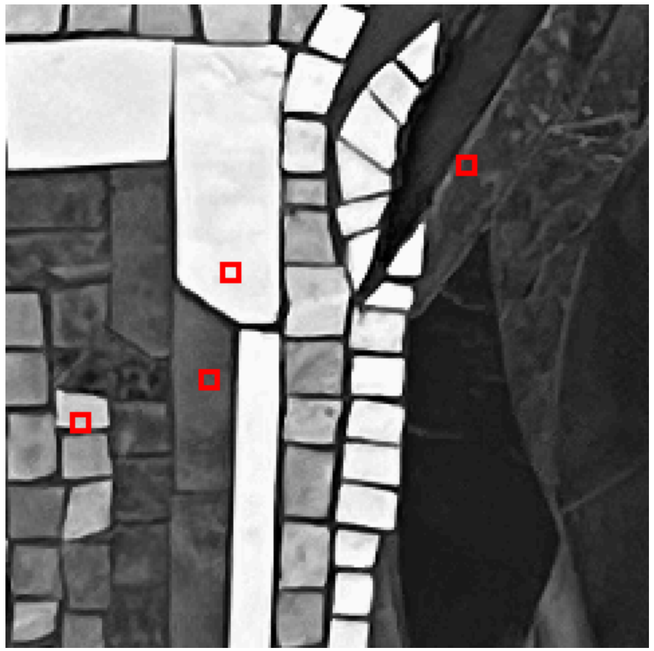
Even though this image has lots of detail

When we zoom in, over small blocks there is a lot of uniformity



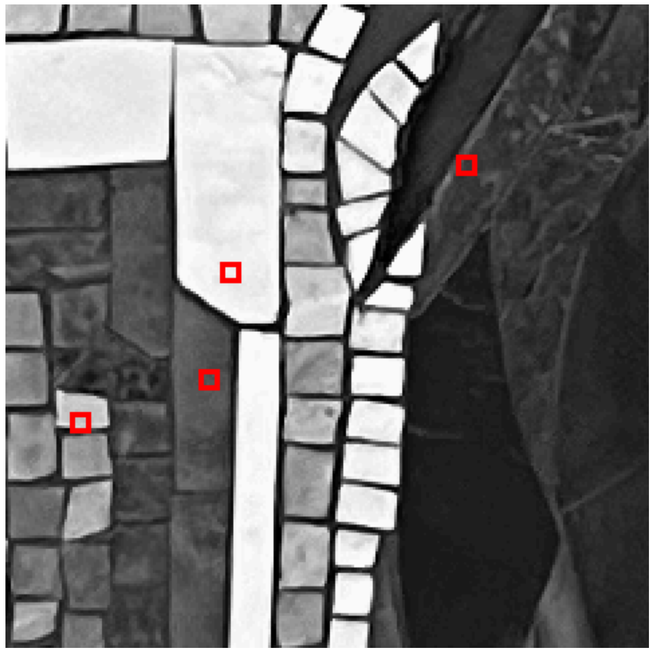
240 x 240

Exploit Uniformity over 8x8 blocks

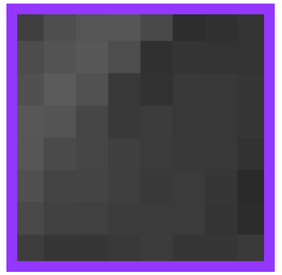


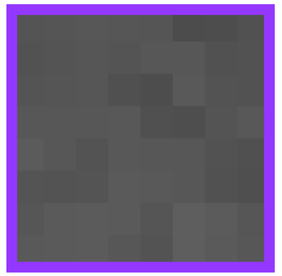
Want to keep detail, but store less

Exploit Uniformity over 8x8 blocks



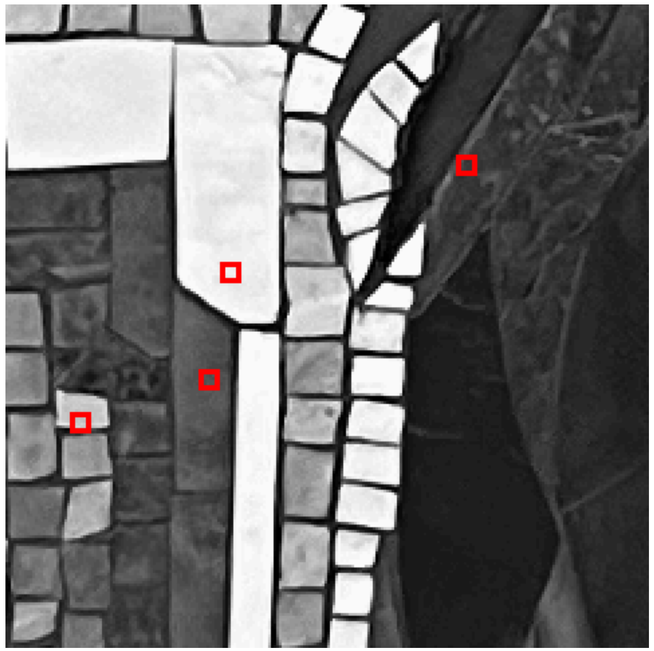
Many blocks look like these, some texture, but mostly samey



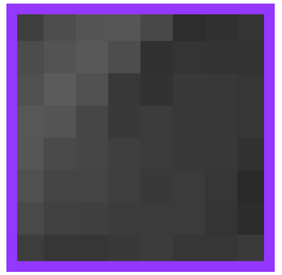


Want to keep detail, but store less

Exploit Uniformity over 8x8 blocks

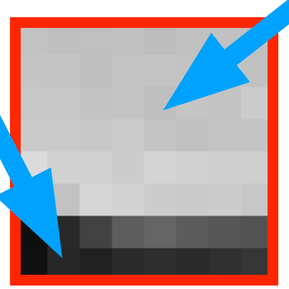


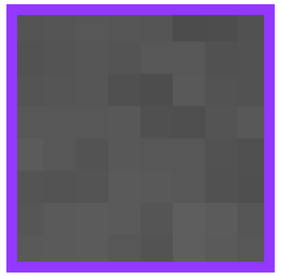
Many blocks look like these, some texture, but mostly samey



Dark down here

Light grey here

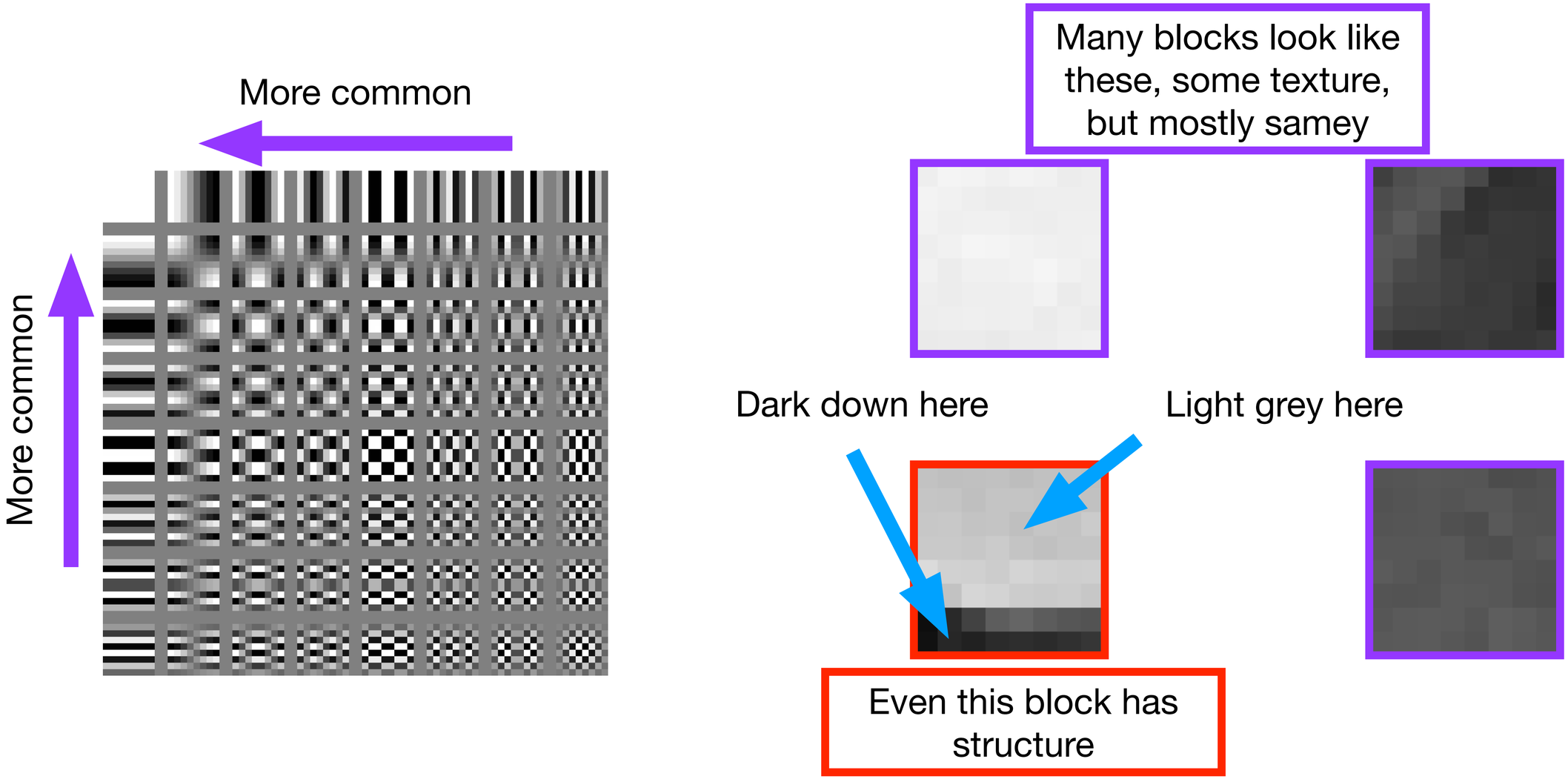




Even this block has structure

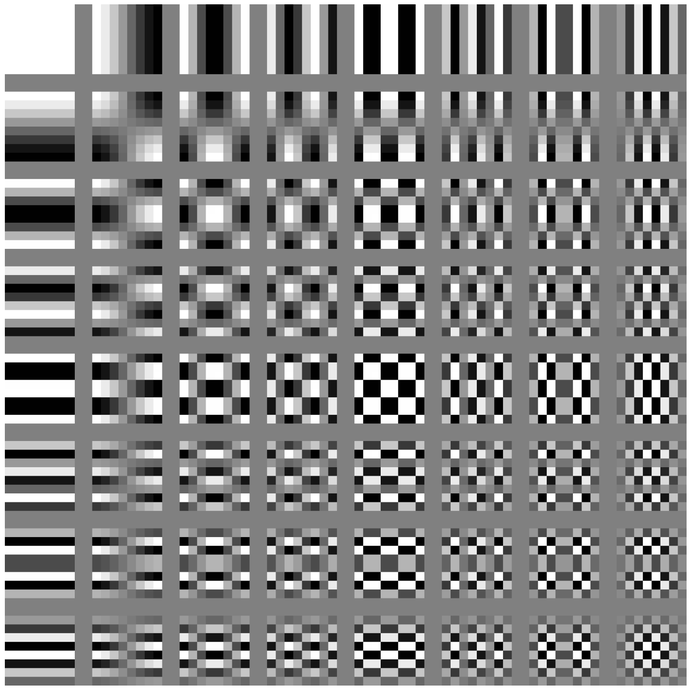
Want to keep detail, but store less

Some Patterns are more common than others



Want to keep detail, but store less

Discrete Cosine Transform (DCT)



Can think of an 8x8 grid of pixels as a vector in $[255]^{64}$

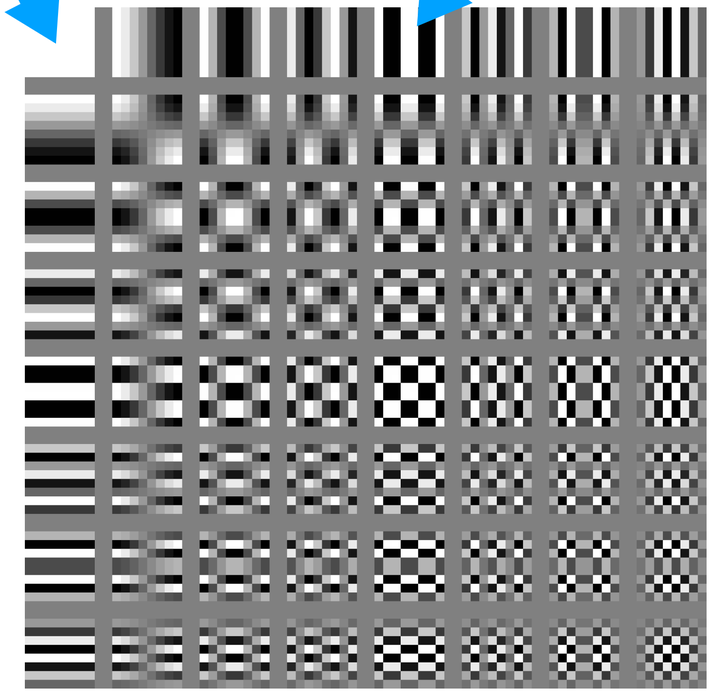
The “usual” basis for this vector space is each individual pixel

The DCT gives us a different basis

Discrete Cosine Transform (DCT)

The average value

The average difference between pixels in these regions



Can think of an 8x8 grid of pixels as a vector in $[255]^{64}$

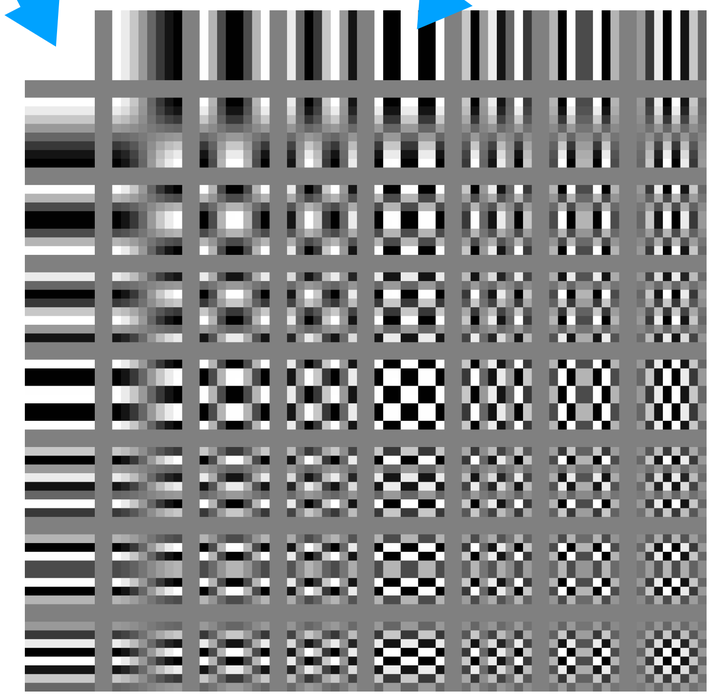
The “usual” basis for this vector space is each individual pixel

The DCT gives us a different basis

Discrete Cosine Transform (DCT)

The average value

The average difference between pixels in these regions



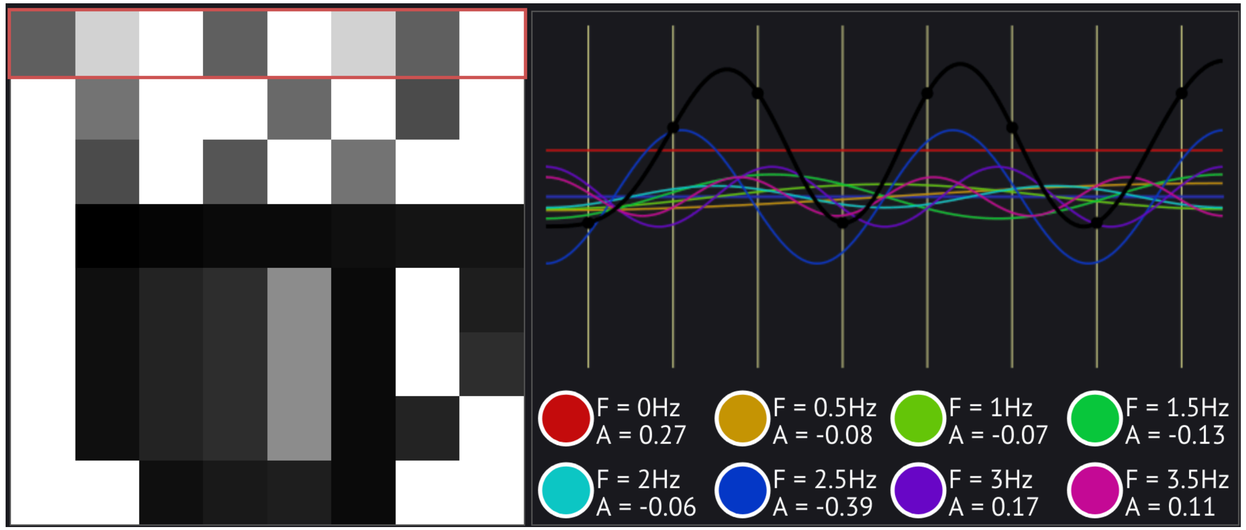
Can think of an 8x8 grid of pixels as a vector in $[255]^{64}$

The “usual” basis for this vector space is each individual pixel

The DCT gives us a different basis

We can represent the 8x8 block as a weighted average of these patterns

Discrete Cosine Transform (DCT)



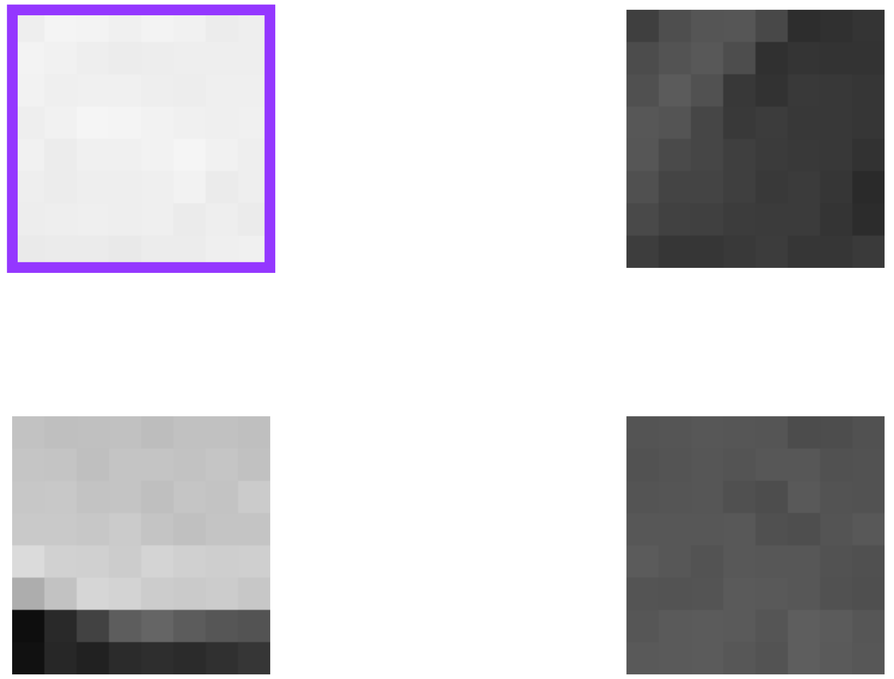
Express a given row as the integer intersections of a sum of cosine functions

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right], \quad k = 0,1,\ldots,N-1$$

A type of Fourier analysis

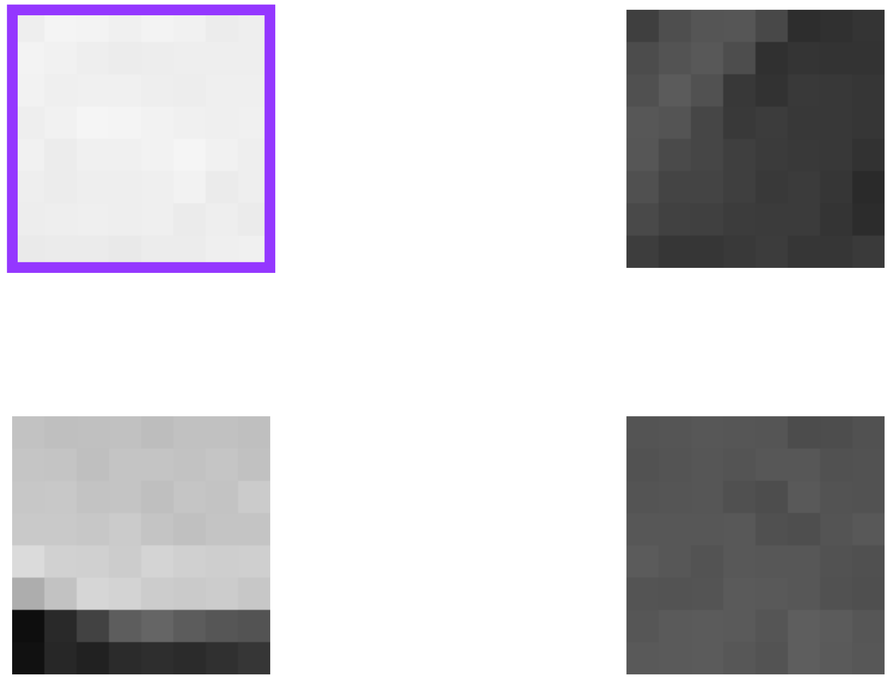
https://alexdowad.github.io/visualizing-the-idct/

Discrete Cosine Transform (DCT)



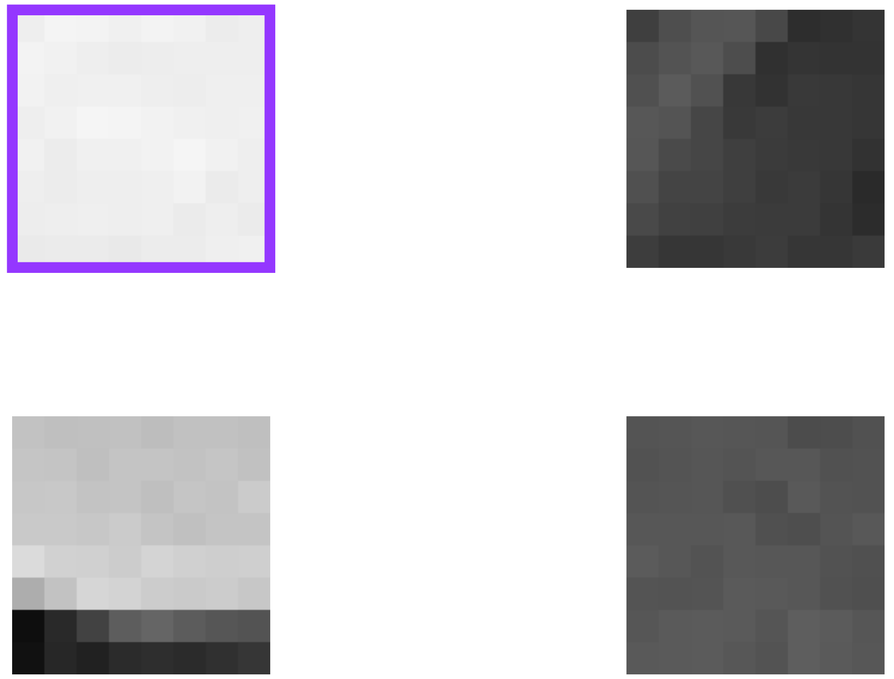
Discrete Cosine Transform (DCT)

887	2	-2	1	-1	-1	2	1
9	6	0	-1	0	-2	0	-1
-7	0	4	-1	-1	-4	-3	1
2	1	-5	4	-1	-3	3	-1
5	-3	-4	-1	-2	-2	1	0
3	3	-5	-5	0	-1	-2	1
-1	-2	0	0	1	-1	2	-1
1	0	0	2	0	2	-1	1



Discrete Cosine Transform (DCT)

887	2	-2	1	-1	-1	2	1
9	6	0	-1	0	-2	0	-1
-7	0	4	-1	-1	-4	-3	1
2	1	-5	4	-1	-3	3	-1
5	-3	-4	-1	-2	-2	1	0
3	3	-5	-5	0	-1	-2	1
-1	-2	0	0	1	-1	2	-1
1	0	0	2	0	2	-1	1



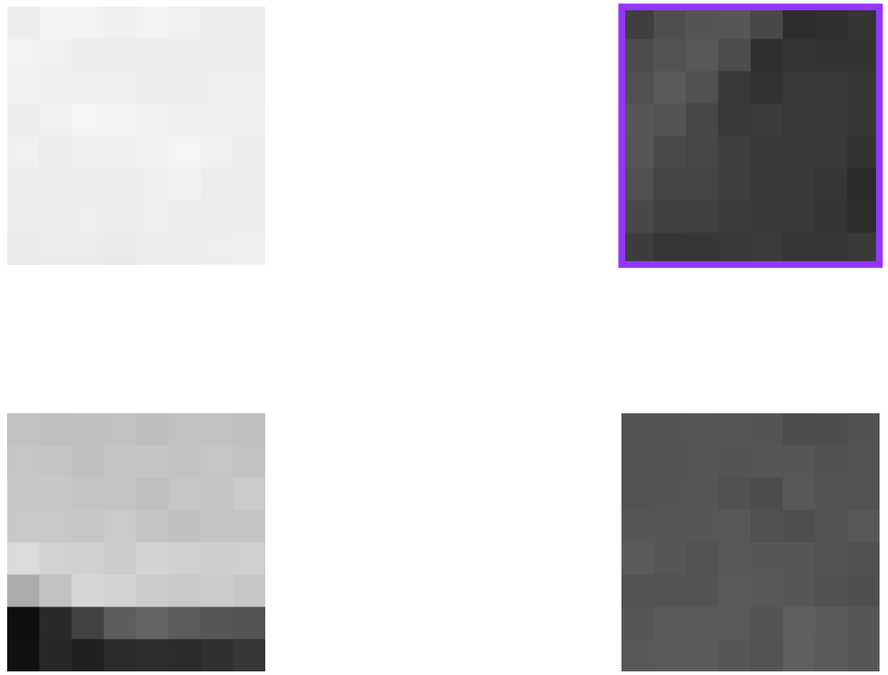
Most of the coeffs are very small (except top left)

Discrete Cosine Transform (DCT)

-519	76	5	-6	-6	3	1	2
24	24	-5	-27	-7	-3	-4	-2
-10	-14	-22	-16	9	0	-2	-1
3	6	-18	0	8	7	-1	1
0	-10	-4	0	13	-5	-6	5
1	1	-3	5	0	-7	-4	4
-2	-4	0	0	-1	-4	-2	2
1	0	0	-1	-2	-2	0	1

Discrete Cosine Transform (DCT)

-519	76	5	-6	-6	3	1	2
24	24	-5	-27	-7	-3	-4	-2
-10	-14	-22	-16	9	0	-2	-1
3	6	-18	0	8	7	-1	1
0	-10	-4	0	13	-5	-6	5
1	1	-3	5	0	-7	-4	4
-2	-4	0	0	-1	-4	-2	2
1	0	0	-1	-2	-2	0	1



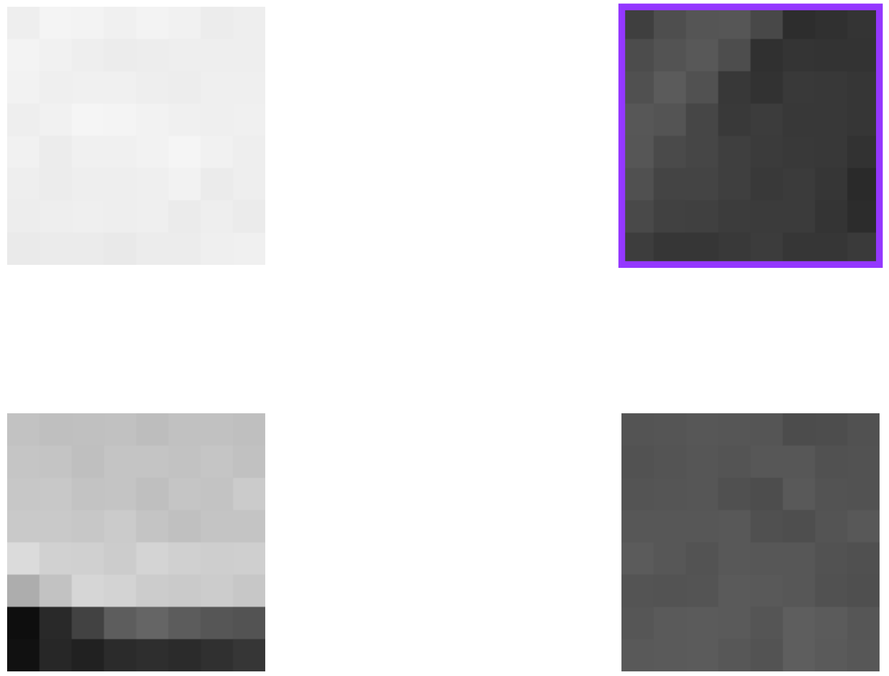
Larger coeffs in general, but still concentrated in the top left

Discrete Cosine Transform (DCT)

-519	76	5	-6	-6	3	1	2
24	24	-5	-27	-7	-3	-4	-2
-10	-14	-22	-16	9	0	-2	-1
3	6	-18	0	8	7	-1	1
0	-10	-4	0	13	-5	-6	5
1	1	-3	5	0	-7	-4	4
-2	-4	0	0	-1	-4	-2	2
1	0	0	-1	-2	-2	0	1

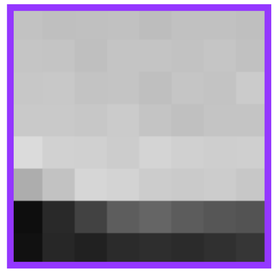
Larger coeffs in general, but still concentrated in the top left

This block has more “texture”



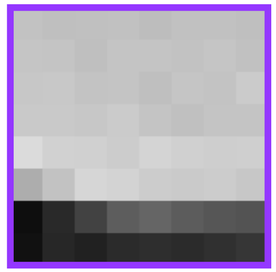
Discrete Cosine Transform (DCT)

280	-29	-23	-11	0	-2	-5	-3
349	42	33	13	3	7	5	-1
-299	-26	-11	-2	-2	-3	-5	-3
157	-2	-16	-2	-5	5	5	6
-25	22	23	8	4	-1	-1	1
-65	-21	-22	-6	1	2	0	-3
75	14	14	-9	-7	0	3	-2
-43	-10	1	9	5	-3	4	0



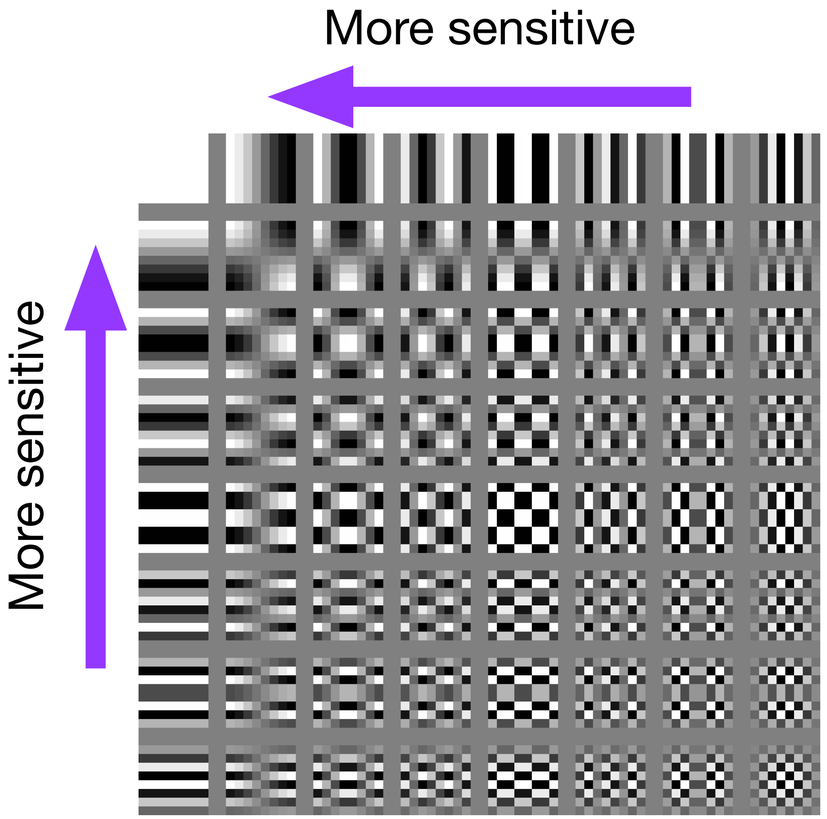
Discrete Cosine Transform (DCT)

280	-29	-23	-11	0	-2	-5	-3
349	42	33	13	3	7	5	-1
-299	-26	-11	-2	-2	-3	-5	-3
157	-2	-16	-2	-5	5	5	6
-25	22	23	8	4	-1	-1	1
-65	-21	-22	-6	1	2	0	-3
75	14	14	-9	-7	0	3	-2
-43	-10	1	9	5	-3	4	0



The pattern is still large-scale and so is mostly represented in the upper left

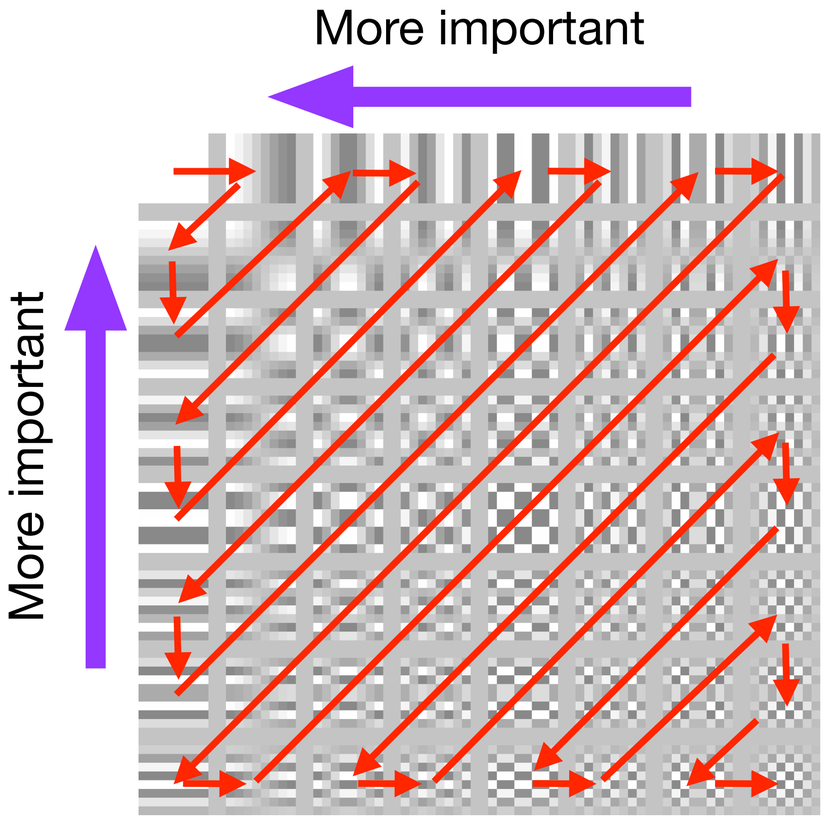
Human Perception and DCT



The human eye is more sensitive to “blockier” signal

So we perceive the upper left coeffs much more than the lower right

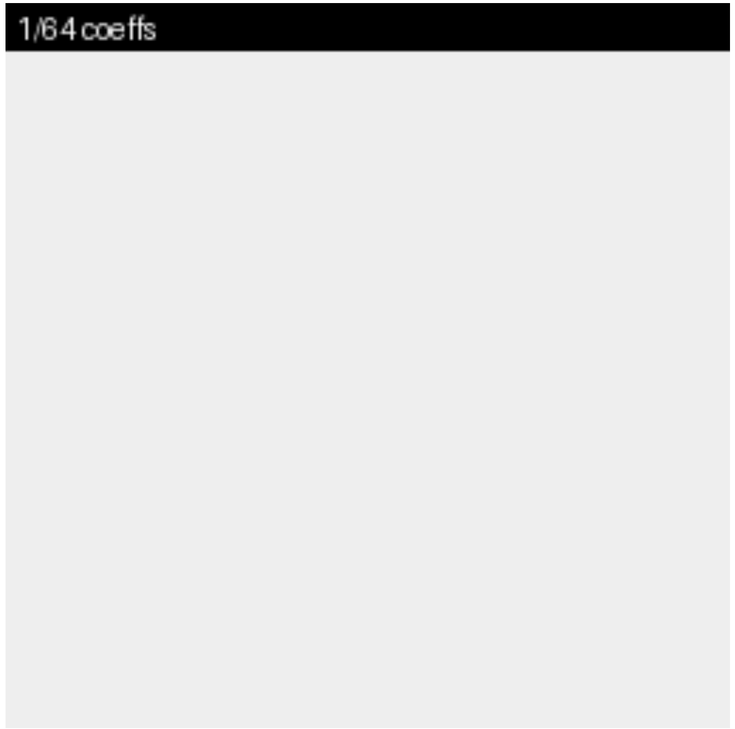
Zig-Zag Path

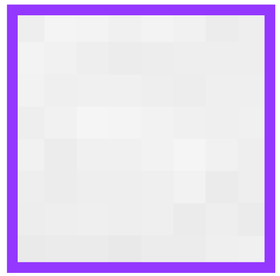


Write out the coefficients in zig-zag order:

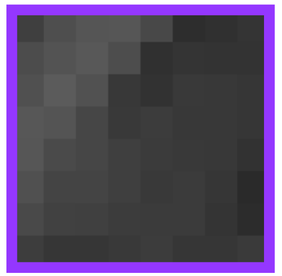
More important coefficients first

Zig-Zag Path

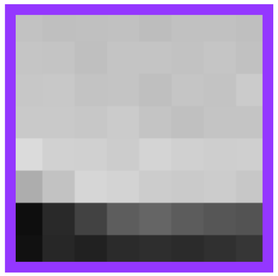




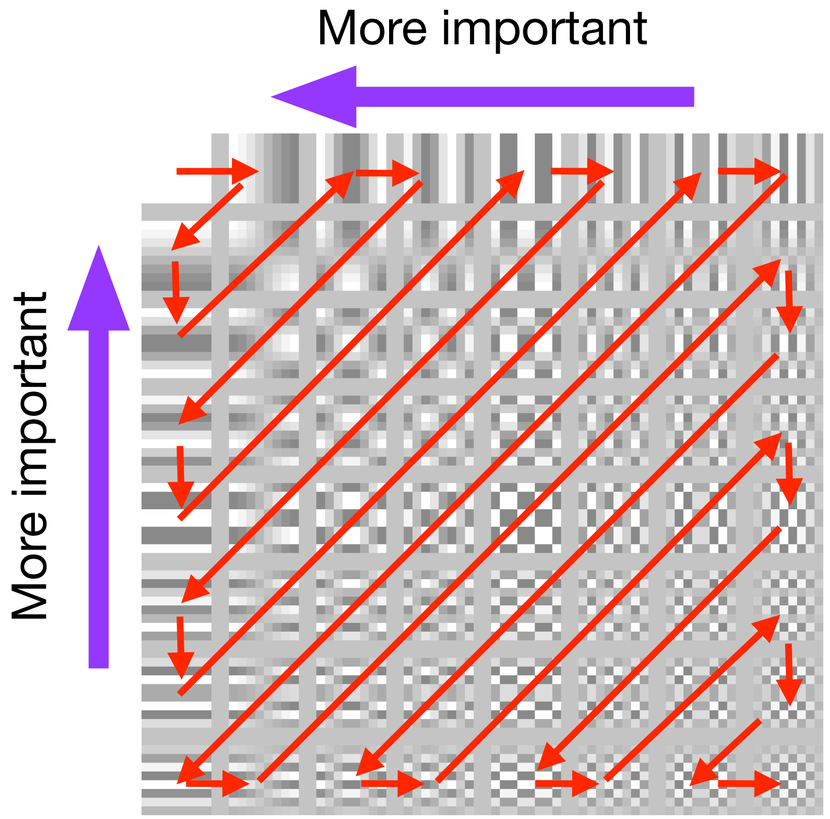
Zig-Zag Path



Zig-Zag Path



Zig-Zag and Quantization



Use more bits to store values in upper left, less in lower right

Important side effect:
Many coeffs round to 0

Zig-Zag Quantization

No quantization

-519	76	5	-6	-6	3	1	2
24	24	-5	-27	-7	-3	-4	-2
-10	-14	-22	-16	9	0	-2	-1
3	6	-18	0	8	7	-1	1
0	-10	-4	0	13	-5	-6	5
1	1	-3	5	0	-7	-4	4
-2	-4	0	0	-1	-4	-2	2
1	0	0	-1	-2	-2	0	1

“50%” quantization

-512	77	0	0	0	0	0	0
24	24	0	-19	0	0	0	0
-14	-13	-16	-24	0	0	0	0
0	0	-22	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

Zig-Zag Quantization

No quantization

-519	76	5	-6	-6	3	1	2
24	24	-5	-27	-7	-3	-4	-2
-10	-14	-22	-16	9	0	-2	-1
3	6	-18	0	8	7	-1	1
0	-10	-4	0	13	-5	-6	5
1	1	-3	5	0	-7	-4	4
-2	-4	0	0	-1	-4	-2	2
1	0	0	-1	-2	-2	0	1

75%

-520	78	5	-8	0	0	0	0
24	24	-7	-30	-13	0	0	0
-7	-14	-24	-12	0	0	0	0
0	9	-22	0	0	0	0	0
0	-11	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

50%

-512	77	0	0	0	0	0	0
24	24	0	-19	0	0	0	0
-14	-13	-16	-24	0	0	0	0
0	0	-22	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

25%

-512	66	0	0	0	0	0	0
24	24	0	-38	0	0	0	0
0	-26	-32	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	0	0

Zig-Zag Quantization

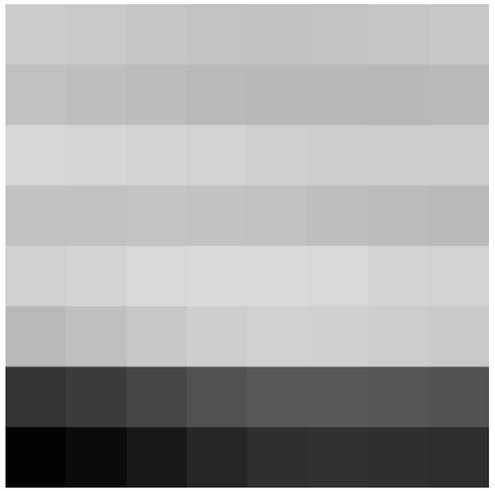
No quantization

280	-29	-23	-11	0	-2	-5	-3
349	42	33	13	3	7	5	-1
-299	-26	-11	-2	-2	-3	-5	-3
157	-2	-16	-2	-5	5	5	6
-25	22	23	8	4	-1	-1	1
-65	-21	-22	-6	1	2	0	-3
75	14	14	-9	-7	0	3	-2
-43	-10	1	9	5	-3	4	0

75%

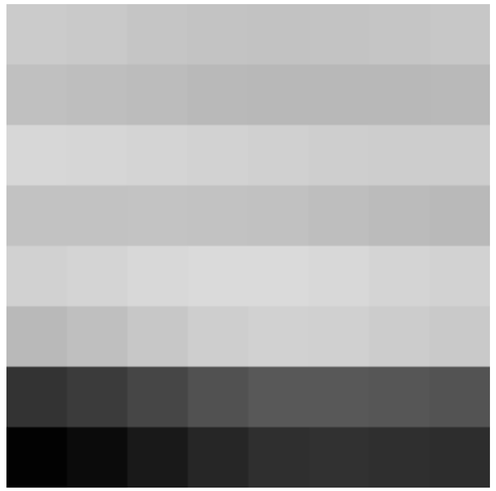
280	-30	-25	-8	0	0	0	0
348	42	35	10	0	0	0	0
-301	-28	-8	0	0	0	0	0
154	0	-11	0	0	0	0	0
-27	22	19	0	0	0	0	0
-60	-18	-28	0	0	0	0	0
75	0	0	0	0	0	0	0
-36	0	0	0	0	0	0	0

50%



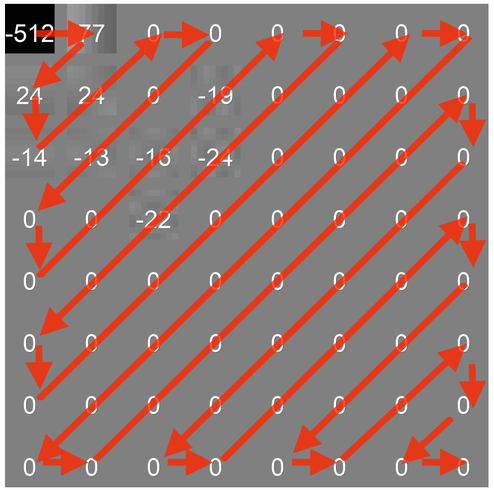
272	-33	-20	-16	0	0	0	0
348	36	28	19	0	0	0	0
-294	-26	-16	0	0	0	0	0
154	0	-22	0	0	0	0	0
-18	22	37	0	0	0	0	0
-72	-35	0	0	0	0	0	0
98	0	0	0	0	0	0	0
-72	0	0	0	0	0	0	0

25%



288	-22	-20	0	0	0	0	0
360	48	28	0	0	0	0	0
-308	-26	0	0	0	0	0	0
168	0	0	0	0	0	0	0
-36	0	0	0	0	0	0	0
-48	0	0	0	0	0	0	0
98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Run-Length Encoding



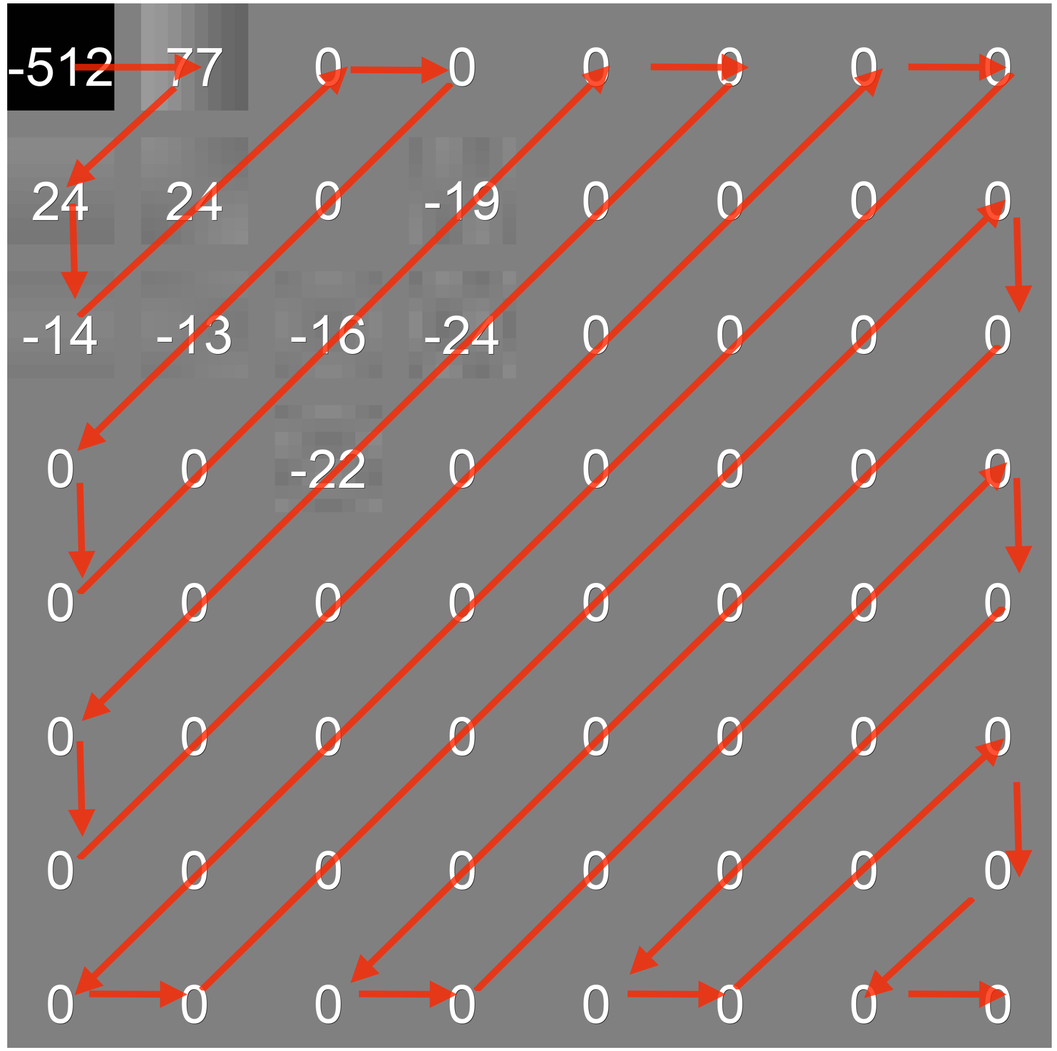
Efficiently store all these zeros

In each entry store
(# zeros, next non-zero value)

(0,-512) (0,77) (0,24) (0,14) (0,24) (3,-13)
(3,-16) (0,19) (3,-24) (0,-22) EOB

EOB (end of block) means no
more non-zero values

Run-Length Encoding



Efficiently store all these zeros

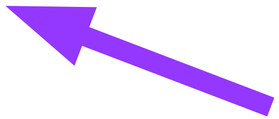
In each entry store
(# zeros, next non-zero value)

(0,-512) (0,77) (0,24) (0,14) (0,24) (3,-13)
(3,-16) (0,19) (3,-24) (0,-22) EOB

EOB (end of block) means no
more non-zero values

Huffman Code (Finally!)

(0,-512) (0,77) (0,24) (0,14) (0,24) (3,-13)
(3,-16) (0,19) (3,-24) (0,-22) EOB

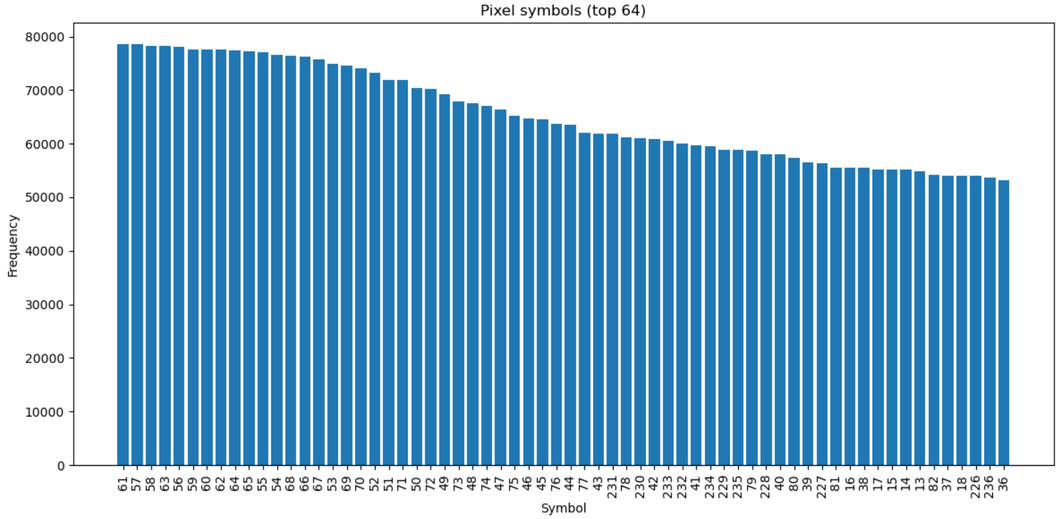


These RLE values are symbols

Build a Huffman code
across the image

Huffman Code (Finally!)

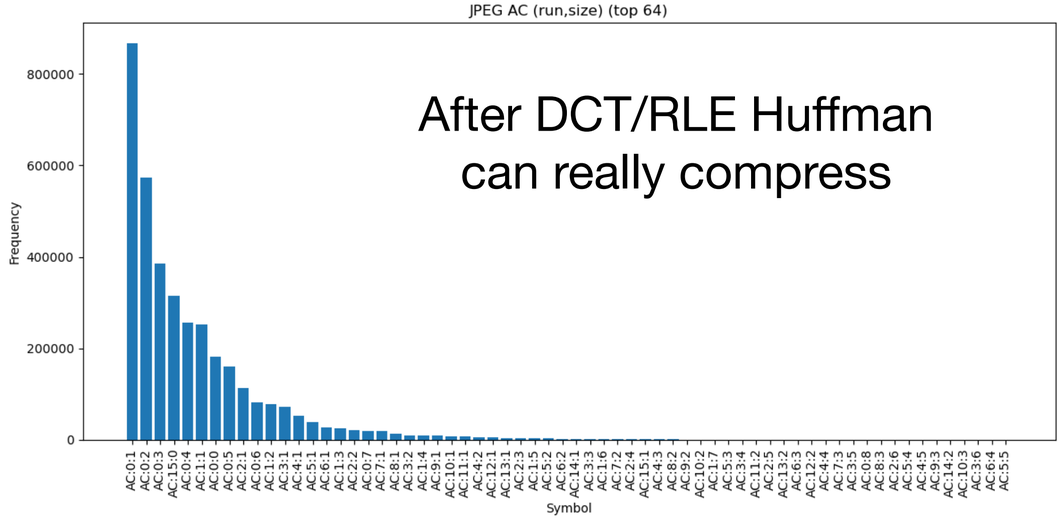
Pixels



Pixel values and frequencies

Size (Y channel):
Original: 12,192,768B
Huffman: 12,156,576B

DCT/RLE Symbols



After DCT/RLE Huffman
can really compress

RLE values and frequencies

Size (Y channel):
Original: 12,192,768B
100%: 8,628,739B
75%: 2,684,420B
50%: 1,894,744B
25%: 1,305,467B