Image Compression

EE274, Fall22

Image Compression

764x512x3 bytes
= 1.1MB!
(Uncompressed)

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

Image from Kodak dataset

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

Image Compression -> BPG

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

Image from Kodak dataset

HiFiC -> ML-based image compression

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

Image from Kodak dataset

Lossy Compression

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

- So ubiquitous, my DSLR camera does JPEG compression by default :-| .. (difficult to find a “dataset” of non-compressed images)

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

Exploiting Spatial correlation in the data

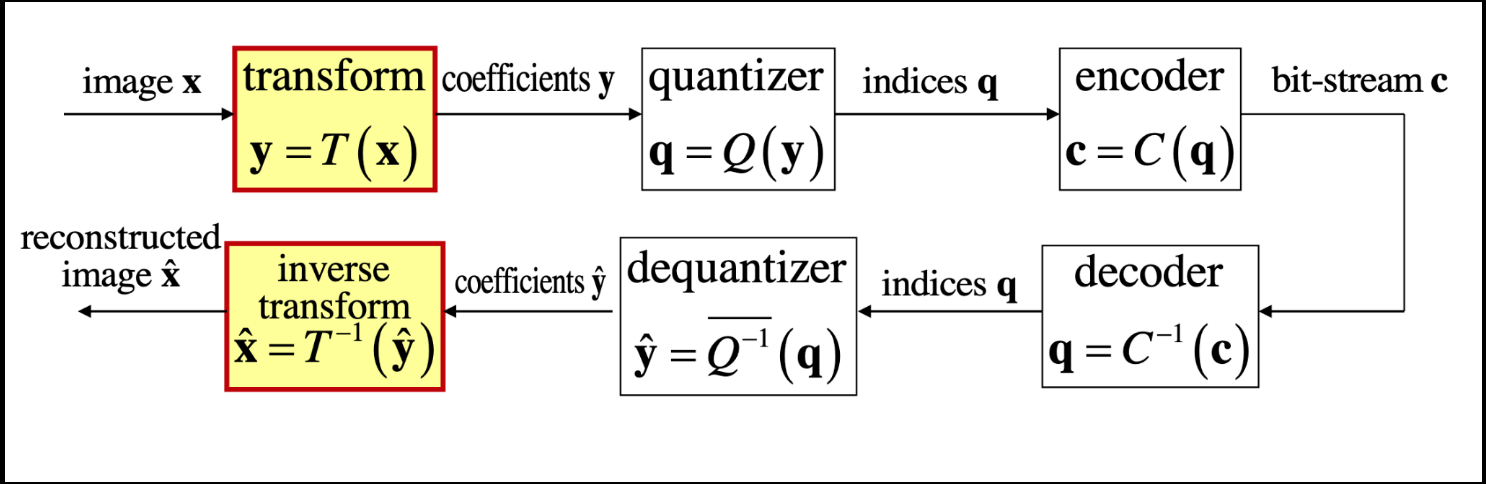
Key Idea -> We need to somehow exploit/remove the correlation between neighboring pixels.

Exploiting Spatial correlation in the data

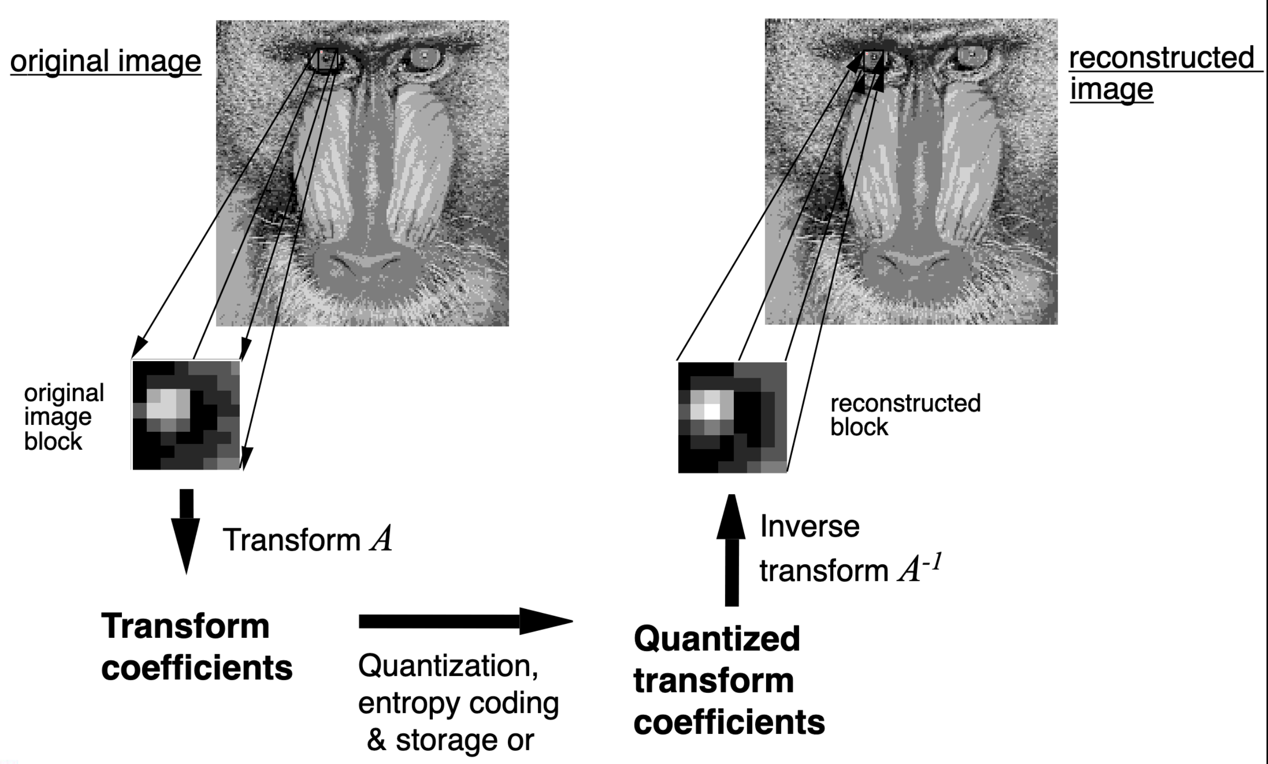
Key Idea -> We need to somehow exploit/remove the correlation between neighboring pixels.

TRANSFORM CODING!

Transform Coding -> RECAP



Block Transform Coding



Linear Transform Coding

- Forward transform

$$\boxed{\mathbf{y} = \mathbf{A}\mathbf{x}}$$

$N x N$ transform coefficients, arranged as a column vector

Transform matrix of size $N^2 x N^2$

Image block of size $N x N$, arranged as a column vector

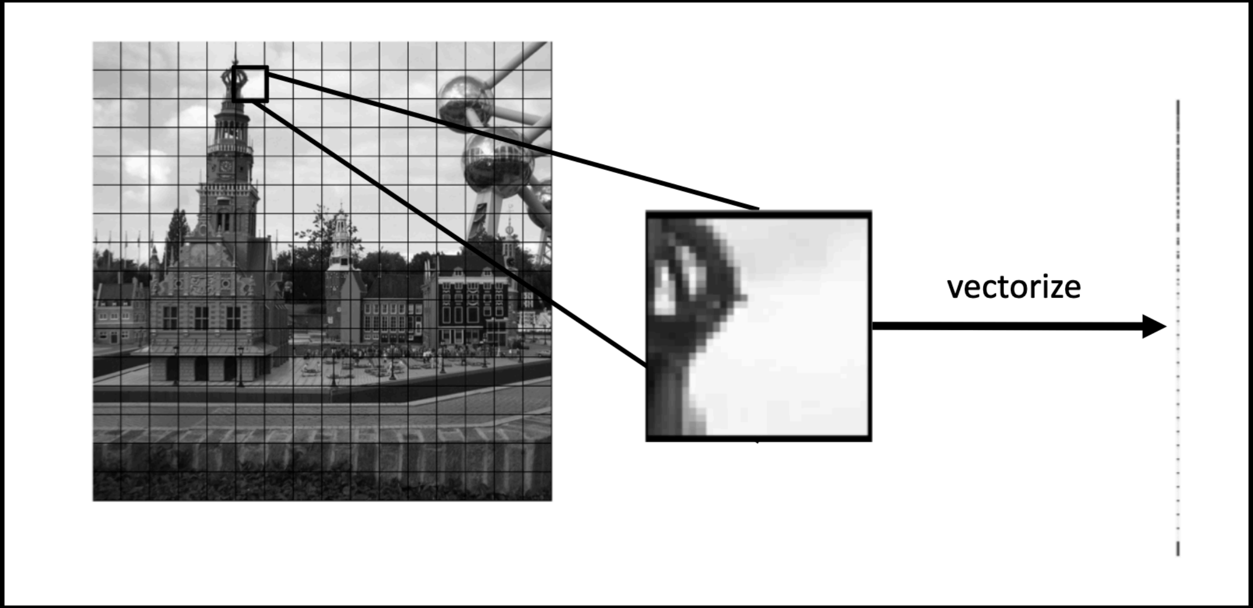
- Inverse transform

$$\boxed{\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} = \mathbf{A}^{T}\mathbf{y}}$$

Block Transform Coding

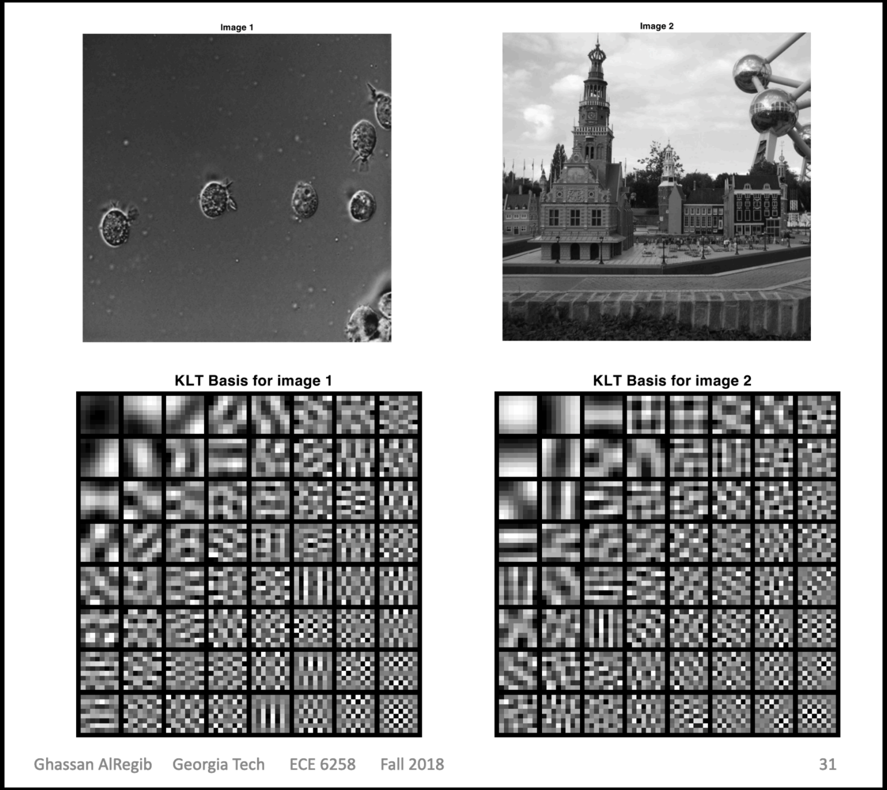
Step 1 -> Cut the image into blocks (eg 8x8), [grayscale]

X



KLT -> Transform Coding

Step 1 -> Cut the image into blocks **X**(eg 8x8)
Step 2 -> Find the transform matrix A
using Karhunen-Loeve Transform (KLT)

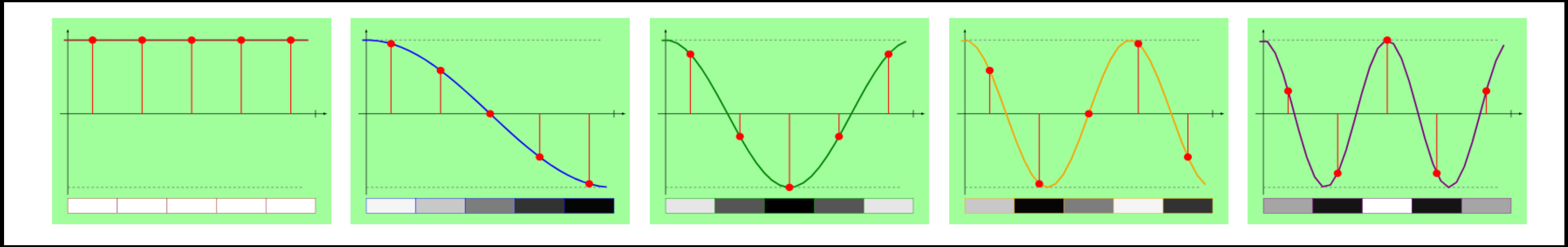


KLT -> Transform Coding

- **Decorrelation by design:** Decorrelated transform coefficients
- **Depends upon the data:** Transform depends upon the input image
- **Slow:** Non-structured matrix of size NxN = 64x64, matrix multiplication is N^2 (too slow :(), KLT construction is also slow

***Q:** Can we design a structured transform, which is close to optimal?*
(i.e. to the KLT matrix)

Transform Coding -> 1D-DCT



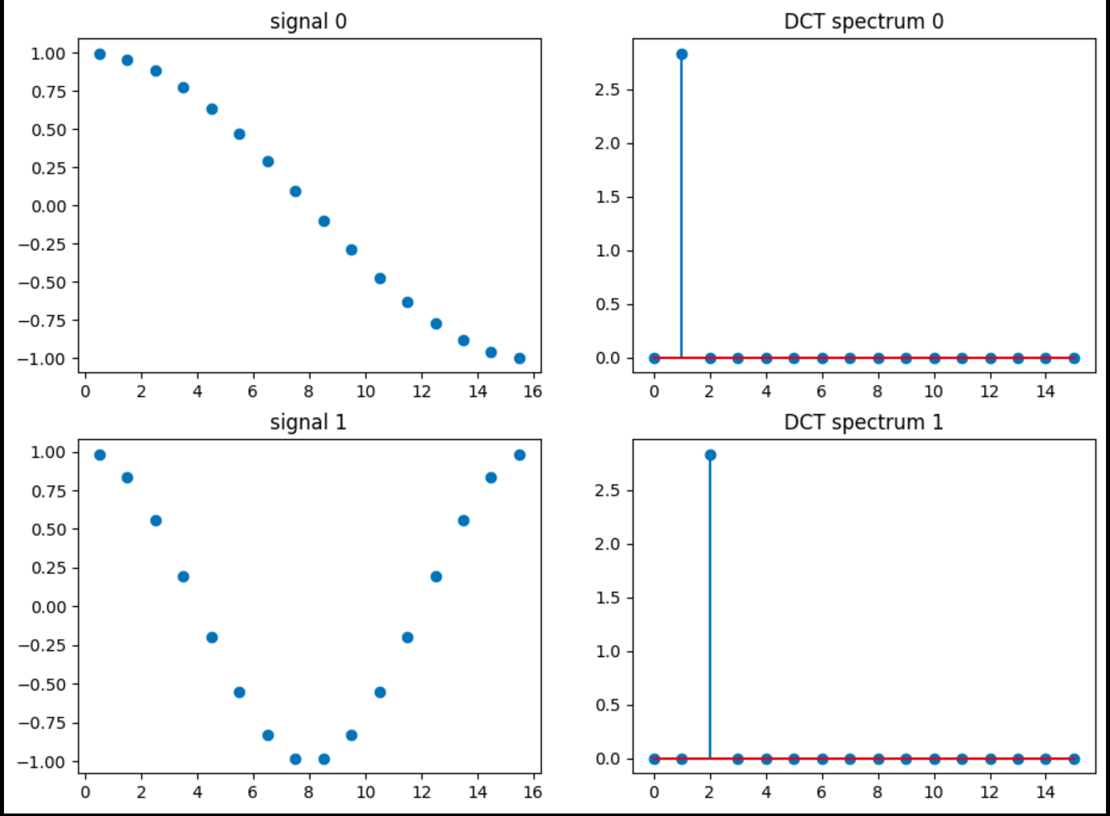
- 1D-> Discrete Cosine transform
  = values of the cosine function at different quantized values
- Forms the basis of any input of size 5
- The DCT vectors are orthonormal

Transform Coding -> 1D-DCT

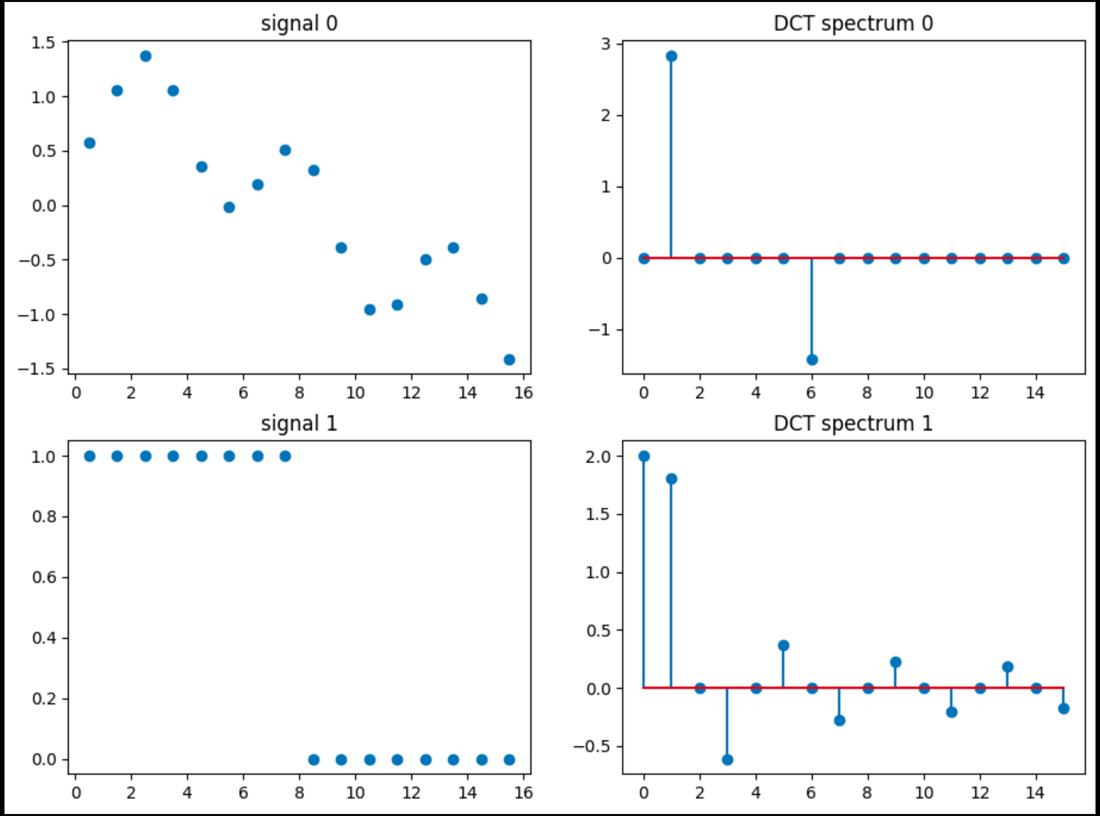
$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{(2n+1)\pi k}{2N}\right] \quad \iff \quad X_k = \vec{C}_k^T \vec{x}$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} \longleftarrow & C_0^T & \longrightarrow \\ \longleftarrow & C_1^T & \longrightarrow \\ & \vdots & \\ \longleftarrow & C_7^T & \longrightarrow \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

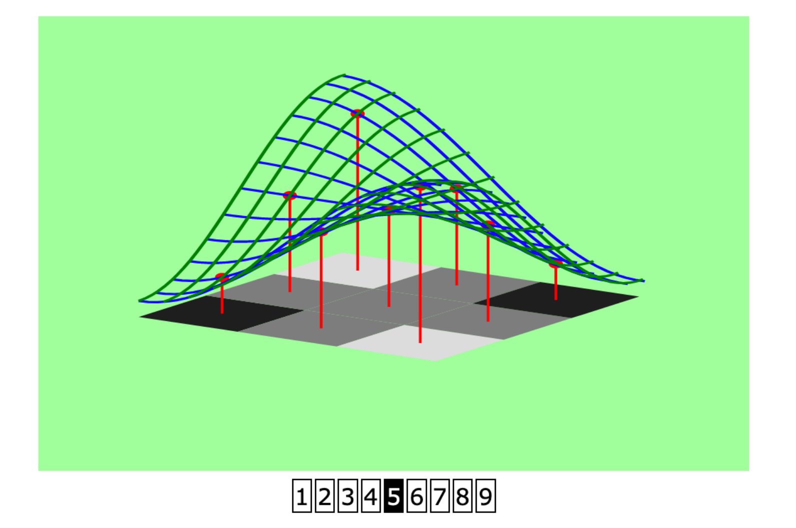
Transform Coding -> 1D-DCT- examples

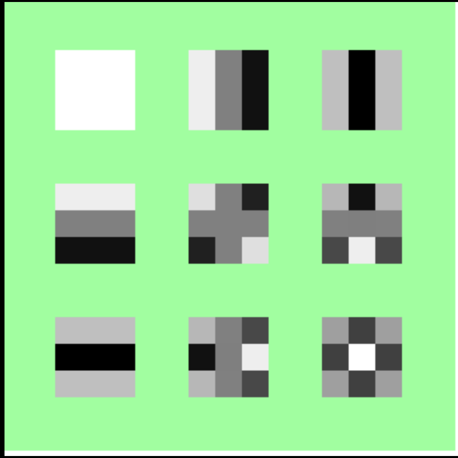


Transform Coding -> 1D-DCT- examples



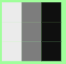
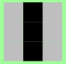
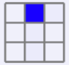
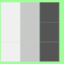
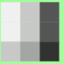
Transform Coding -> 2D-DCT



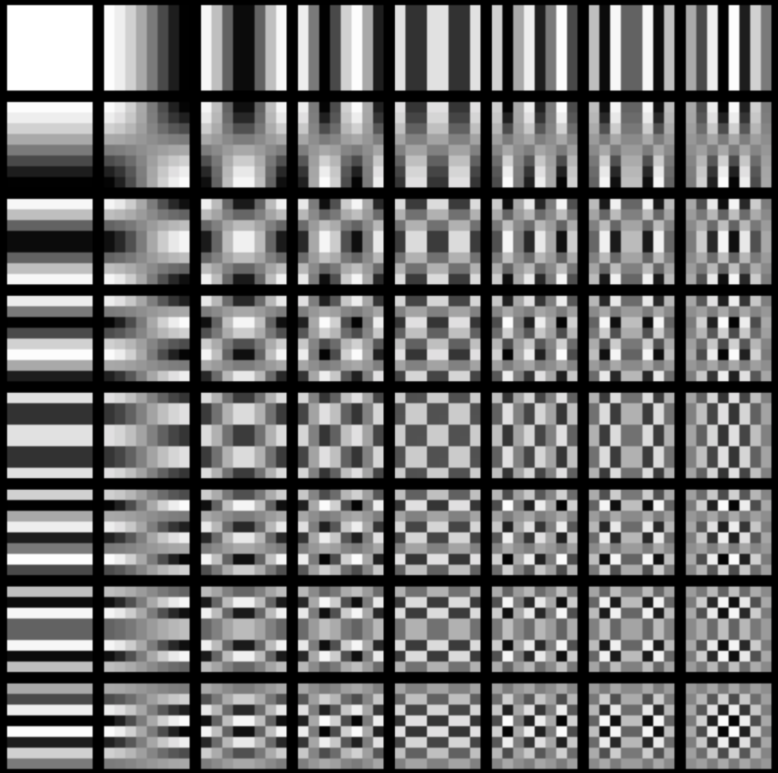


2D-DCT basis vectors
(apply 1D along x, and then y)

Transform Coding -> 2D-DCT

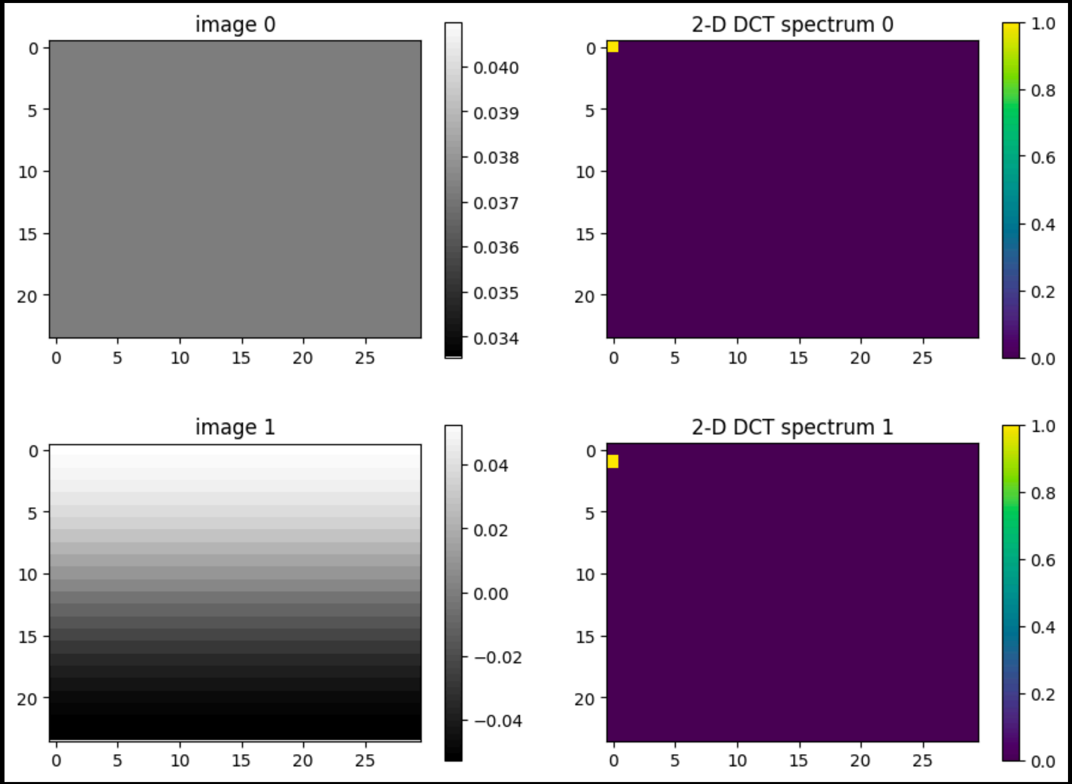
		1	2	3	4	5	6	7	8	9	
											
A		0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	
B		0.667	0.000	0.000	-0.333	0.000	0.000	0.000	0.000	0.000	
C		0.400	0.000	0.693	0.000	0.000	-0.200	0.000	0.000	0.000	
D		0.300	0.173	0.693	-0.100	0.000	-0.200	0.000	0.000	0.000	
E		0.200	-0.065	0.064	0.435	-0.527	0.041	-0.547	-0.267	-0.014	
F		0.200	-0.065	0.064	0.435	-0.527	0.041	-0.547	-0.267	0.000	

Transform Coding -> 2D-DCT vectors

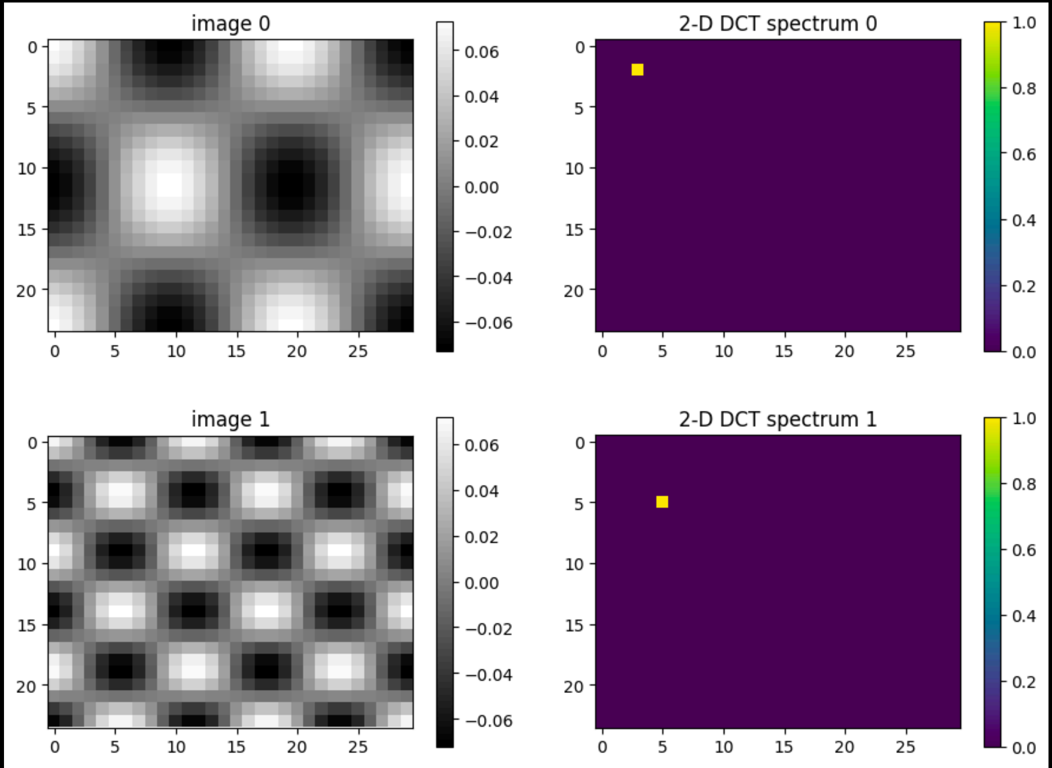


2D-DCT basis vectors for 8x8 blocks

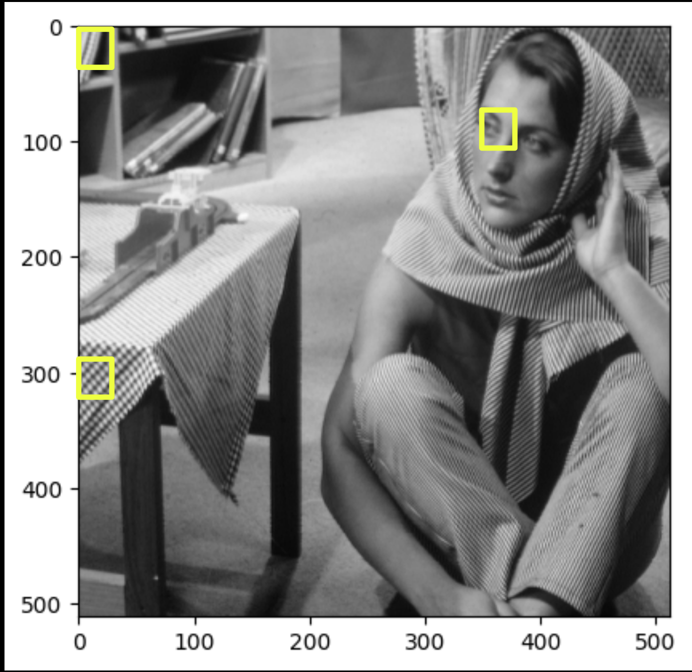
Transform Coding -> DCT

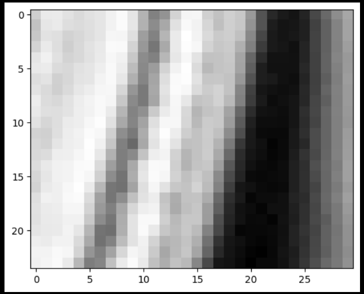


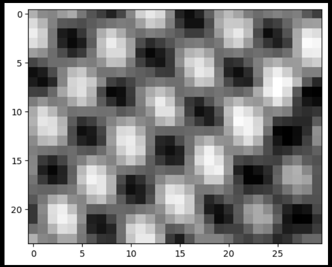
Transform Coding -> DCT

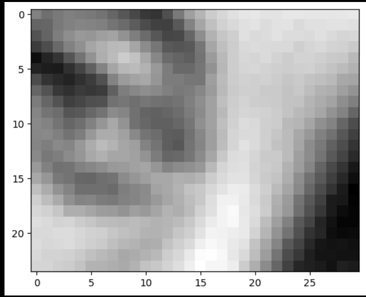


Transform Coding -> DCT

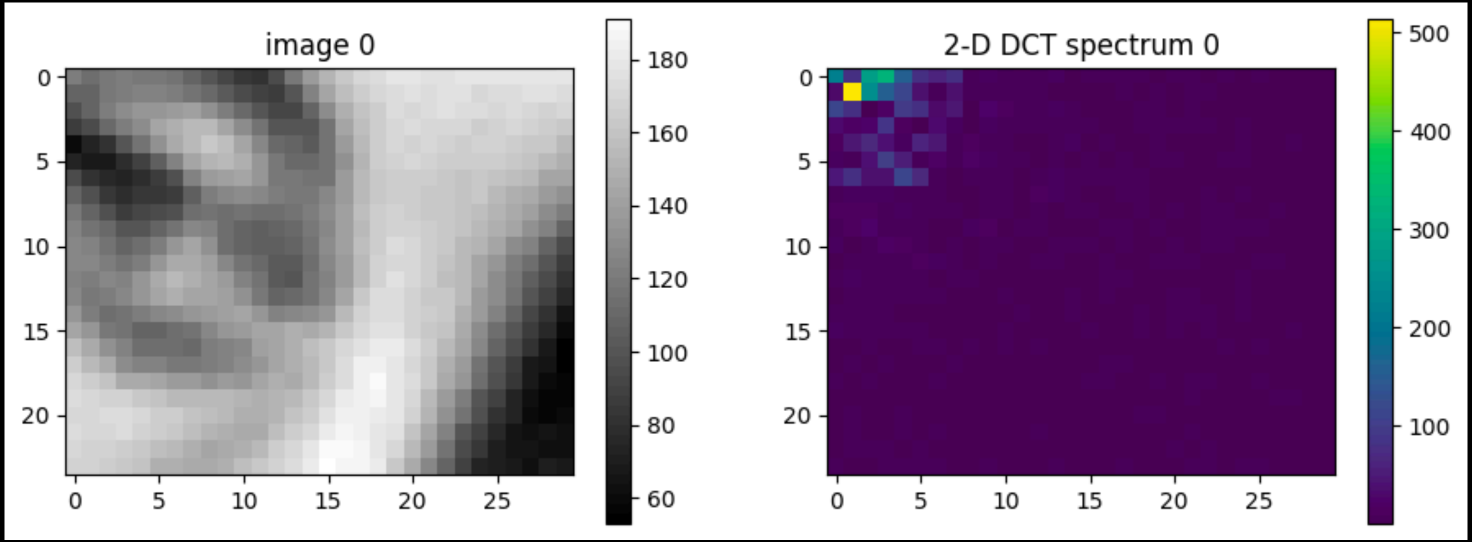






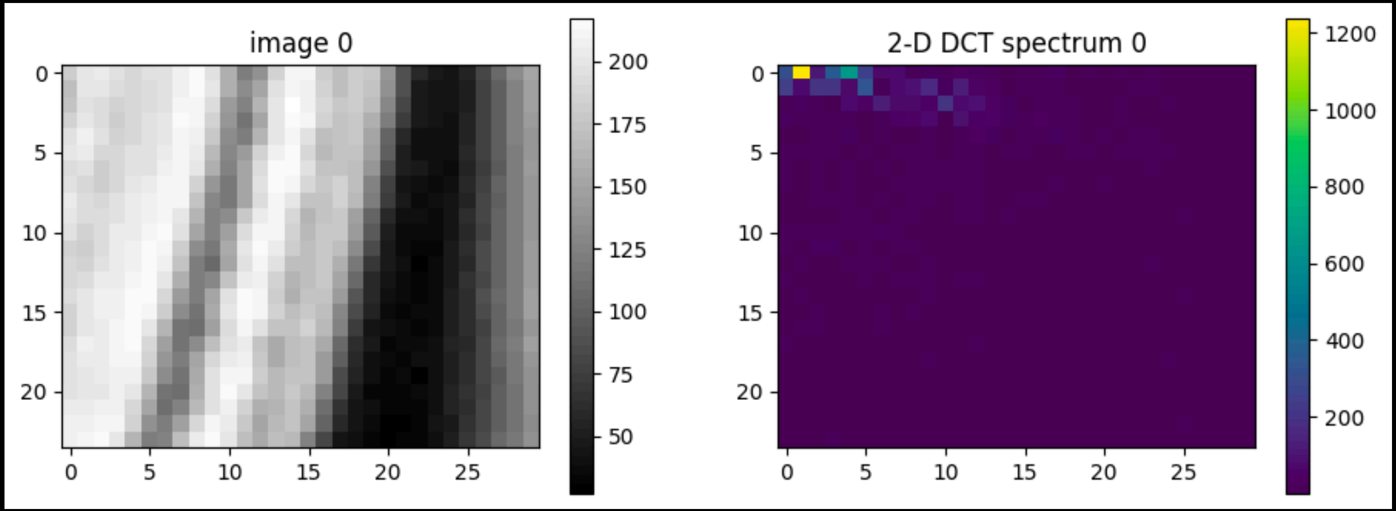


Transform Coding -> DCT



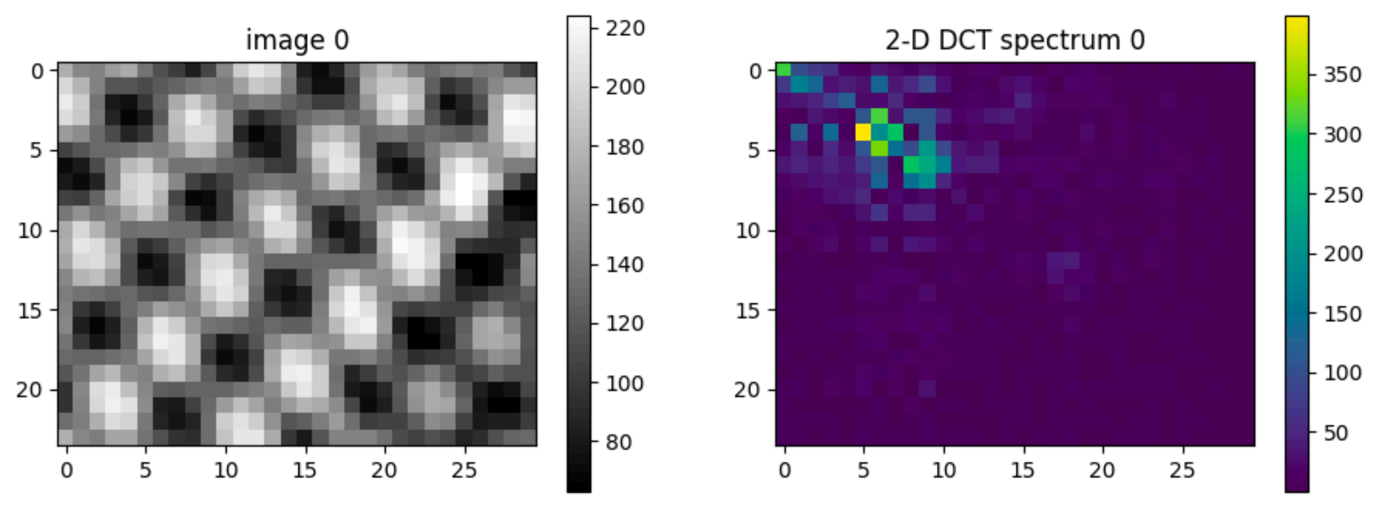
DCT -> Sparse

Transform Coding -> DCT



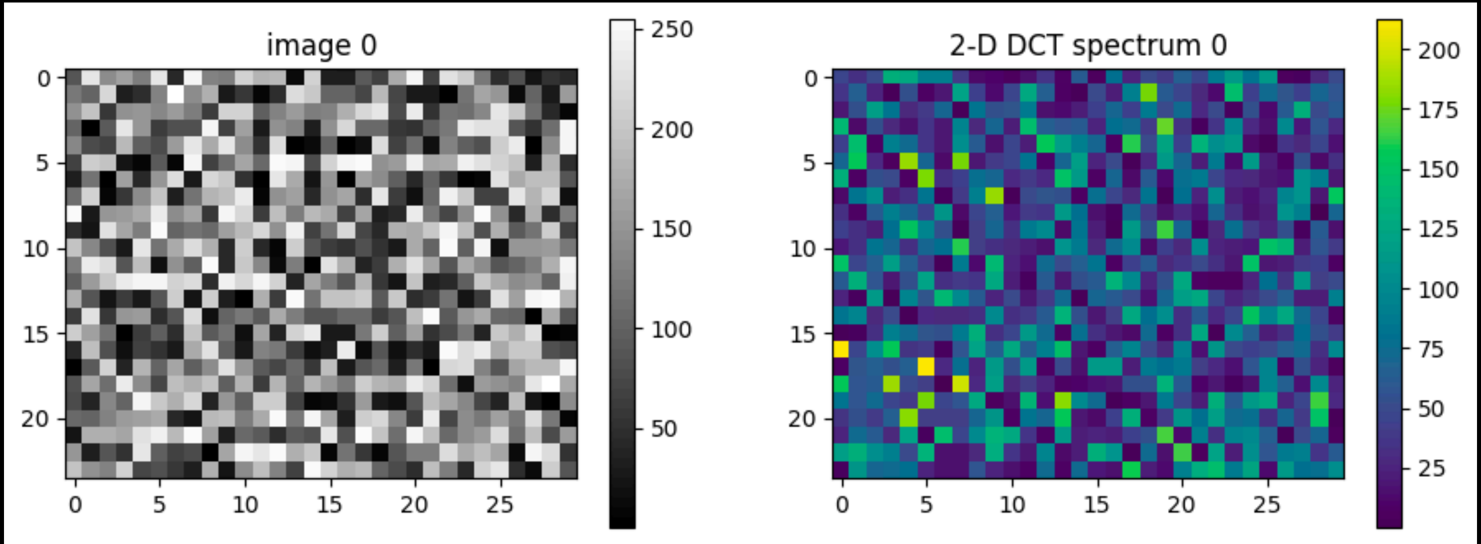
DCT -> Sparse

Transform Coding -> DCT



DCT -> Sparse (but higher frequencies)

Transform Coding -> DCT of noise

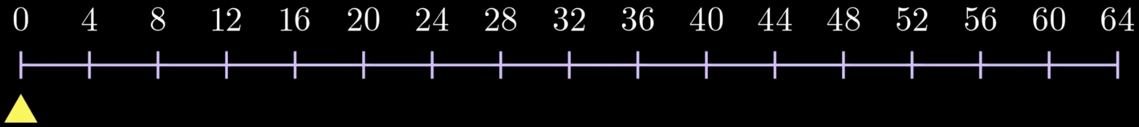


DCT -> Not-so sparse

Transform Coding -> DCT

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

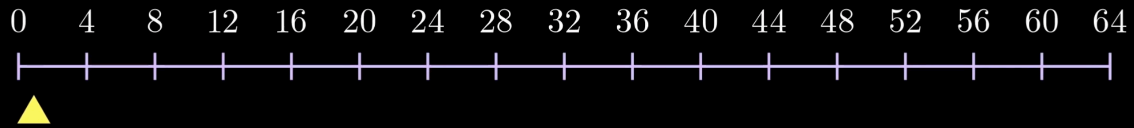
DCT Components



Transform Coding -> DCT

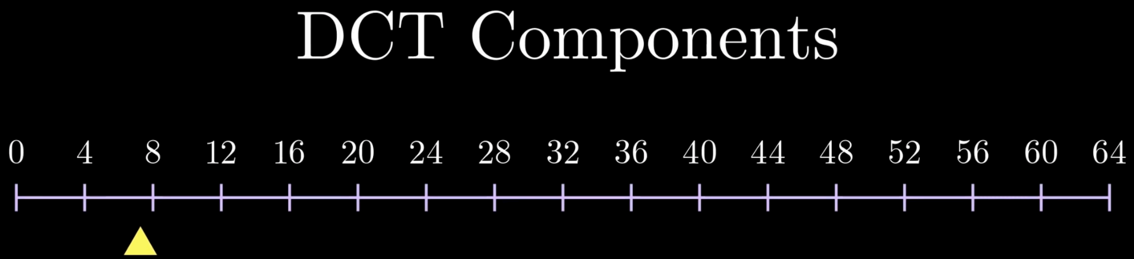
- Observation: For most of the “natural” image blocks, the DCT is sparse, and concentrated in the lower frequencies

DCT Components



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

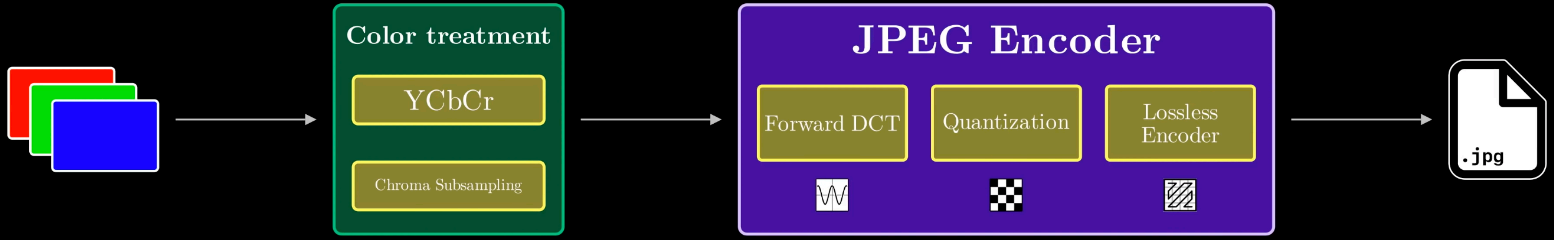
JPEG Image Compression

JPEG

A photo of a European wildcat with the compression rate decreasing and hence quality increasing, from left to right

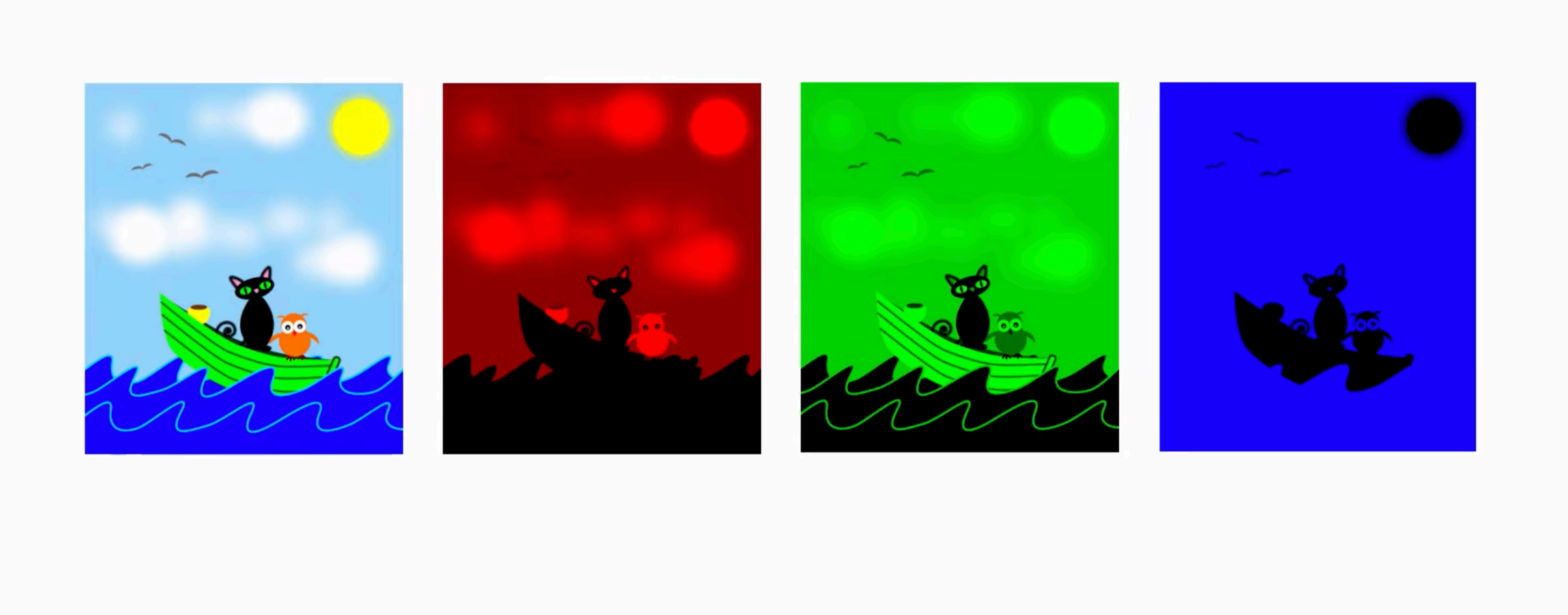
Filename extension	`.jpg`, `.jpeg`, `.jpe` `.jif`, `.jfif`, `.jfi`
Internet media type	`image/jpeg`
Type code	`JPEG`
Uniform Type Identifier (UTI)	public.jpeg
Magic number	`ff d8 ff`
Developed by	Joint Photographic Experts Group, IBM, Mitsubishi Electric, AT&T, Canon Inc.[1]
Initial release	September 18, 1992; 30 years ago
Type of format	Lossy image compression format
Standard	ISO/IEC 10918, ITU-T T.81, ITU-T T.83, ITU-T T.84, ITU-T T.86
Website	www.jpeg.org/jpeg/

JPEG Image Compression

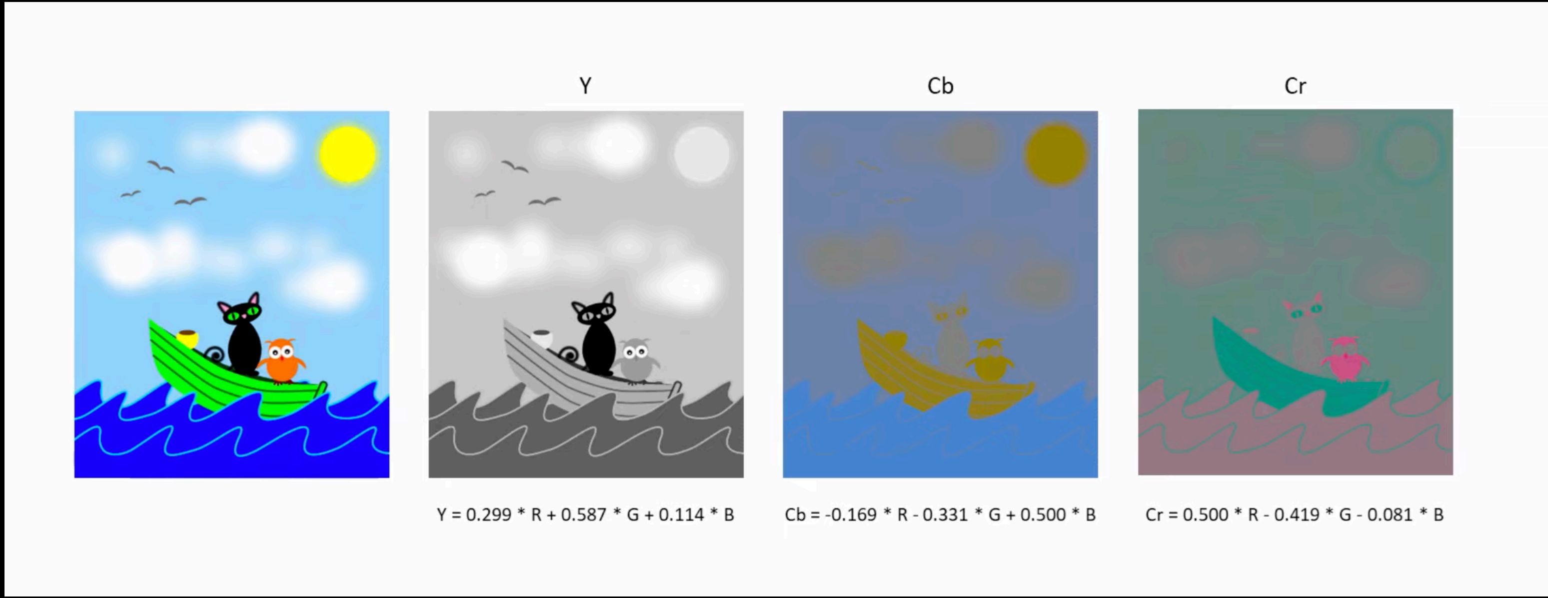


Optional color transform
+ color sub-sampling

RGB colorspace

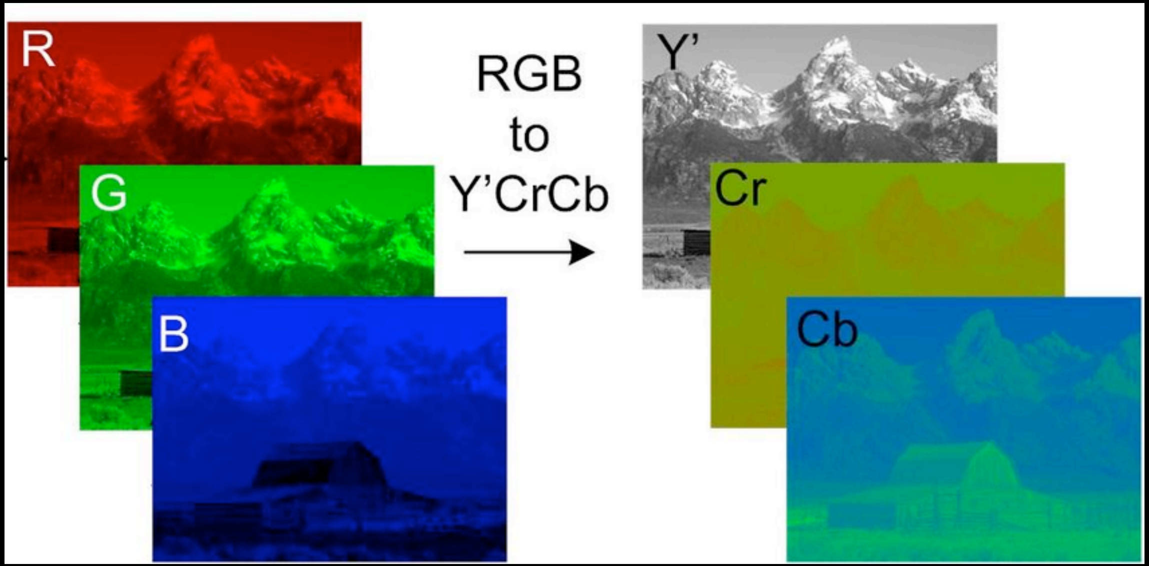


YCbCr Color space

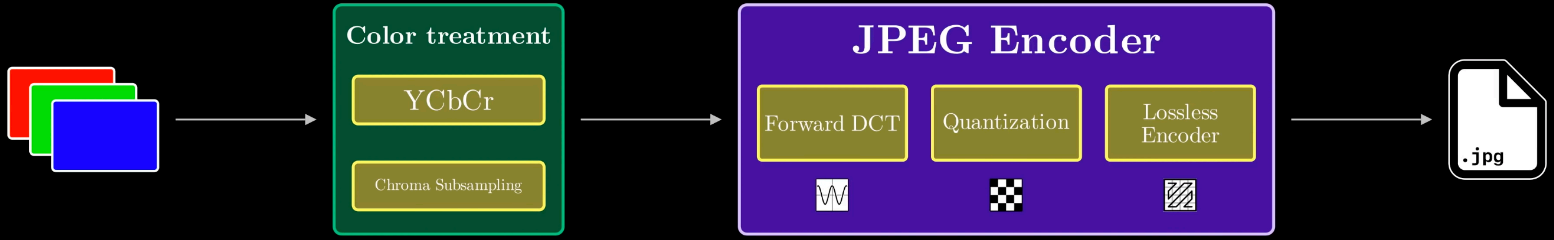


JPEG Image Compression

Optional color transform
+ color sub-sampling

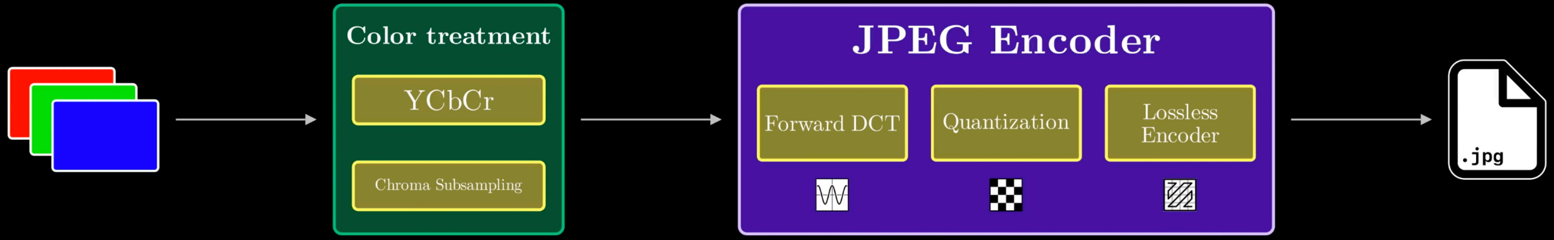


JPEG Image Compression (Baseline Encoding)



Optional color transform
+ color sub-sampling

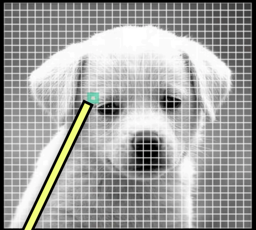
JPEG Image Compression

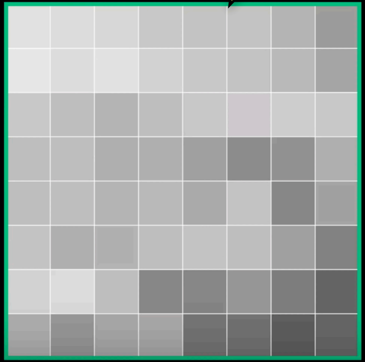


Encoding done per channel (independently)

JPEG Image Compression -> 2D-Block DCT

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



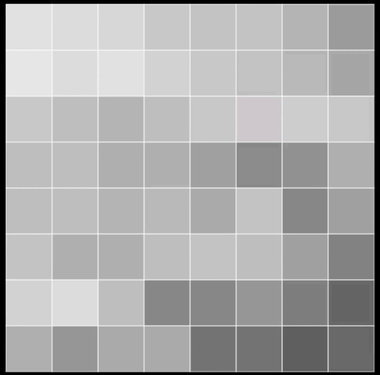


Input 8x8 block

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



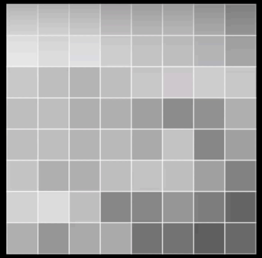
93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	-2	26
64	41	42	55	60	57	25	-8
77	87	58	-2	-5	14	-10	-35
38	14	33	33	-21	-23	-43	-34

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

Transform Coding -> DCT



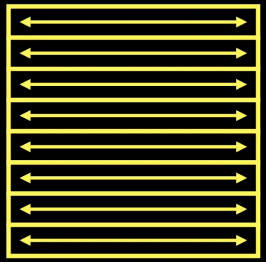
93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	-2	26
64	41	42	55	60	57	25	-8
77	87	58	-2	-5	14	-10	-35
38	14	33	33	-21	-23	-43	-34

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)

$8 \times$ DCT 1D

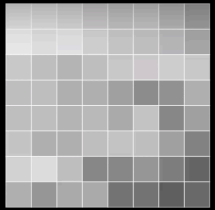
Rows



184	60	-8	12	-13	3	0	0
202	63	-6	4	-7	6	4	3
176	-12	7	16	4	3	-2	0
93	39	17	-21	18	-5	-6	-3
116	37	-12	7	2	-14	22	-27
118	37	-33	37	2	8	3	-1
65	107	23	14	-40	-10	7	0
-1	79	-11	-18	12	19	16	-12

1D DCT (along x)

Transform Coding -> DCT

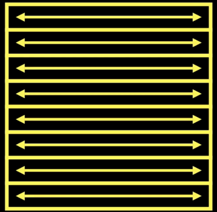


93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	-2	26
64	41	42	55	60	57	25	-8
77	87	58	-2	-5	14	-10	-35
38	14	33	33	-21	-23	-43	-34

Input 8x8 block (zero centered)

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

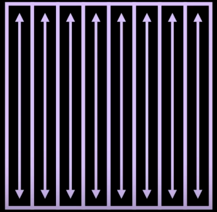
$8 \times$ DCT 1D $\rightarrow$ Rows



184	60	-8	12	-13	3	0	0
202	63	-6	4	-7	6	4	3
176	-12	7	16	4	3	-2	0
93	39	17	-21	18	-5	-6	-3
116	37	-12	7	2	-14	22	-27
118	37	-33	37	2	8	3	-1
65	107	23	14	-40	-10	7	0
-1	79	-11	-18	12	19	16	-12

184	60	-8	12	-13	3	0	0
202	63	-6	4	-7	6	4	3
176	-12	7	16	4	3	-2	0
93	39	17	-21	18	-5	-6	-3
116	37	-12	7	2	-14	22	-27
118	37	-33	37	2	8	3	-1
65	107	23	14	-40	-10	7	0
-1	79	-11	-18	12	19	16	-12

$8 \times$ DCT 1D $\rightarrow$ Columns

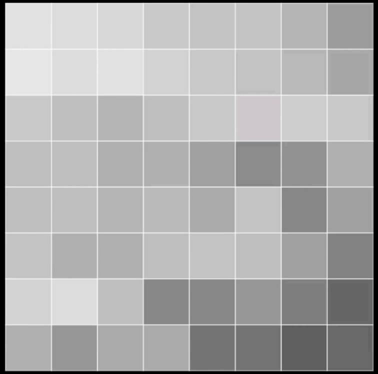


338	145	-8	18	-7	4	16	-14
162	-41	3	2	3	-1	-13	9
-17	57	-2	-2	-20	16	2	10
41	19	-24	31	-19	-8	4	-1
-59	7	-2	-32	21	-1	6	-15
-19	12	32	0	-16	-9	-15	12
7	-55	-24	17	20	15	-4	0
15	-11	10	11	-18	-13	10	-10

2D DCT

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

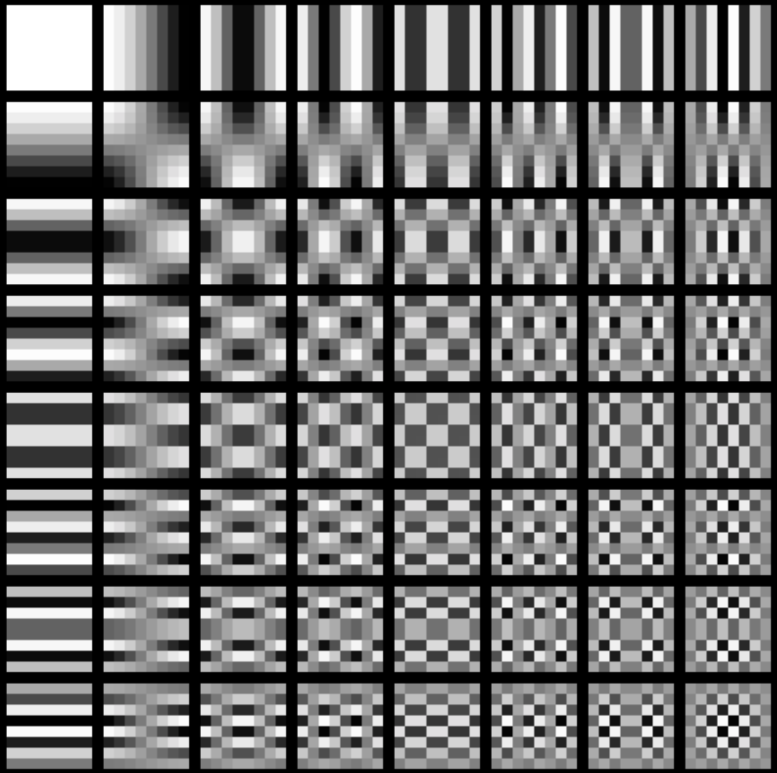
2D DCT

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

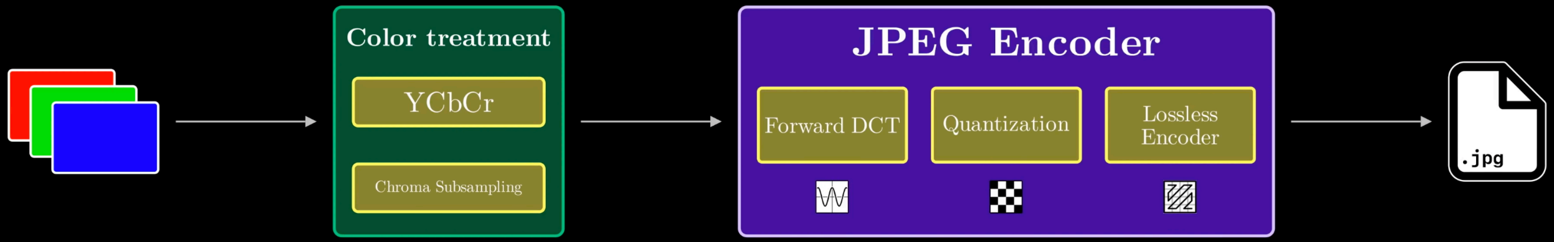
338	145	-8	18	-7	4	16	-14
162	-41	3	2	3	-1	-13	9
-17	57	-2	-2	-20	16	2	10
41	19	-24	31	-19	-8	4	-1
-59	7	-2	-32	21	-1	6	-15
-19	12	32	0	-16	-9	-15	12
7	-55	-24	17	20	15	-4	0
15	-11	10	11	-18	-13	10	-10

2D DCT



2D basis vectors

JPEG Image Compression



Efficient separable DCT implementation

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.

2D-DCT

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

$\div$ $\longrightarrow$

Quantization Table

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

$\longrightarrow$

Quantized-DCT coefficients

21	13	-1	1	0	0	0	0
14	-3	0	0	0	0	0	0
-1	4	0	0	-1	0	0	0
3	1	-1	1	0	0	0	0
-3	0	0	-1	0	0	0	0
-1	0	1	0	0	0	0	0
0	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

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.

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

$\xrightarrow{\div}$

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

Quantization Table

$\longrightarrow$

21	13	-1	1	0	0	0	0
14	-3	0	0	0	0	0	0
-1	4	0	0	-1	0	0	0
3	1	-1	1	0	0	0	0
-3	0	0	-1	0	0	0	0
-1	0	1	0	0	0	0	0
0	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantized-DCT coefficients

JPEG Image Compression -> Quantization

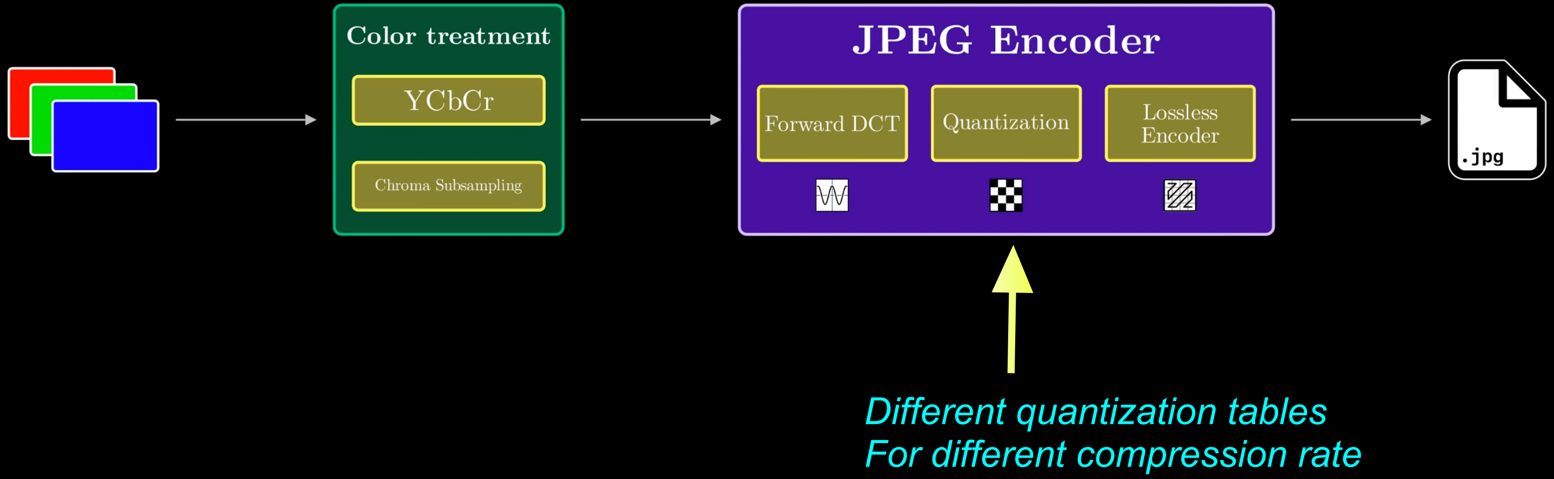
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

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

JPEG Image Compression



JPEG Image Compression

90	-40	0	4	0	0	0	0
0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
11	-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

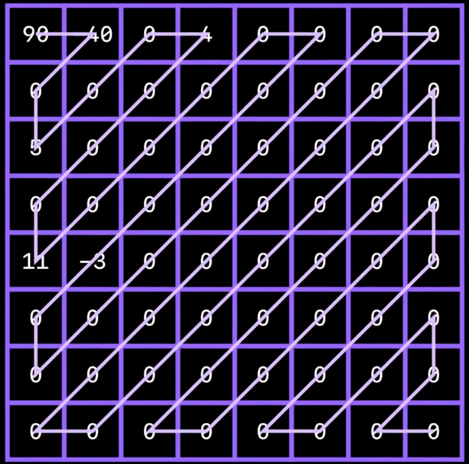
Quantized, transformed Coefficients for one 8x8 block

Q: How would you go ahead with lossless compression of these coefficients?

JPEG Image Compression

90	-40	0	4	0	0	0	0
0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
11	-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

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 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 0 0 }

Run-length + Huffman

JPEG Image Compression -> Entropy coding

90	-40	0	4	0	0	0	0
0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
11	-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

JPEG Specific Huffman Encoding

It's quite complicated . . .

- Signs of coefficients
- All 8×8 blocks
- Top left (DC) coefficients encoded separately from other (AC) coefficients

{[(0, 7), 90], [(0, 6), -40], [(1, 3), 5], [(2, 3), 4], [(3, 4), 11], [(8, 2), -3], [(0, 0)]}

→ Huffman Encoder

JPEG Compression:

- Color Channels: For Each color channel is encoded independently of each other

- Block Coding: JPEG encodes each 8x8 almost independently (except the DC coefficient).

- Huffman/Arithmetic: JPEG also has support for using Arithmetic coding, but is rarely used.

Linear Transform Coding

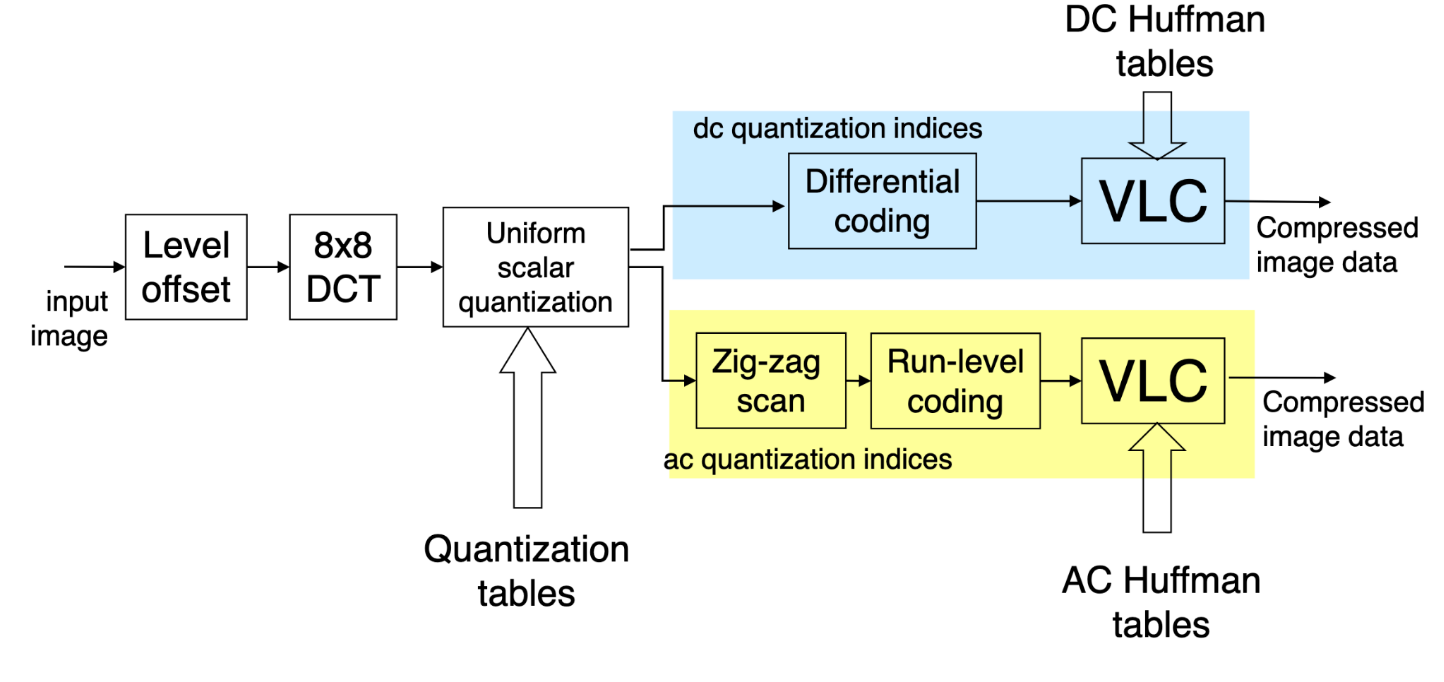
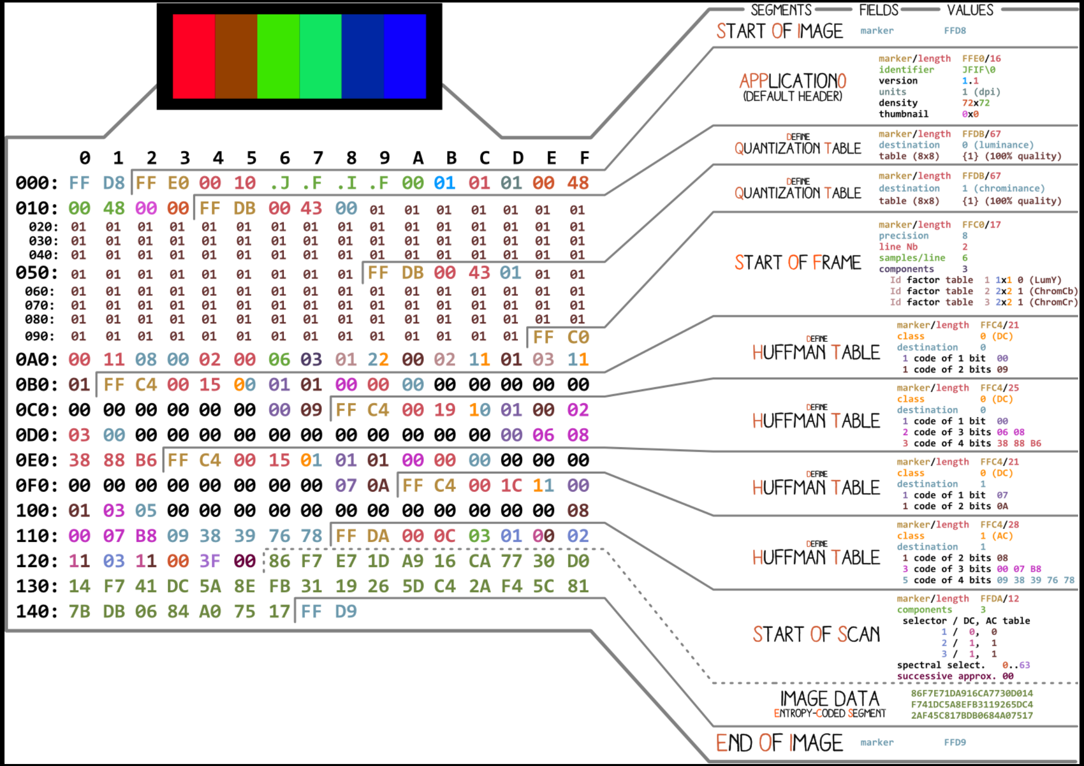


Image Compression -> Analysis

JPEG Decoder specification



JPEG Decoder specification

Common JPEG markers[48]

Short name	Bytes	Payload	Name	Comments
SOI	0xFF, 0xD8	*none*	Start Of Image	
SOF0	0xFF, 0xC0	*variable size*	Start Of Frame (baseline DCT)	Indicates that this is a baseline DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
SOF2	0xFF, 0xC2	*variable size*	Start Of Frame (progressive DCT)	Indicates that this is a progressive DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
DHT	0xFF, 0xC4	*variable size*	Define Huffman Table(s)	Specifies one or more Huffman tables.
DQT	0xFF, 0xDB	*variable size*	Define Quantization Table(s)	Specifies one or more quantization tables.
DRI	0xFF, 0xDD	4 bytes	Define Restart Interval	Specifies the interval between RST*n* markers, in Minimum Coded Units (MCUs). This marker is followed by two bytes indicating the fixed size so it can be treated like any other variable size segment.
SOS	0xFF, 0xDA	*variable size*	Start Of Scan	Begins a top-to-bottom scan of the image. In baseline DCT JPEG images, there is generally a single scan. Progressive DCT JPEG images usually contain multiple scans. This marker specifies which slice of data it will contain, and is immediately followed by entropy-coded data.
RST*n*	0xFF, 0xD*n* (*n*=0..7)	*none*	Restart	Inserted every *r* macroblocks, where *r* is the restart interval set by a DRI marker. Not used if there was no DRI marker. The low three bits of the marker code cycle in value from 0 to 7.
APP*n*	0xFF, 0xE*n*	*variable size*	Application-specific	For example, an Exif JPEG file uses an APP1 marker to store metadata, laid out in a structure based closely on TIFF.
COM	0xFF, 0xFE	*variable size*	Comment	Contains a text comment.
EOI	0xFF, 0xD9	*none*	End Of Image	

What are the issues with JPEG?

- Block size 8x8

What are the issues with JPEG?

- Block size 8x8
- Blocks processed independently

What are the issues with JPEG?

- Block size 8x8
- Blocks processed independently
- lossless coding can be improved

Image Compression -> JPEG 137x

Image from Kodak dataset

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

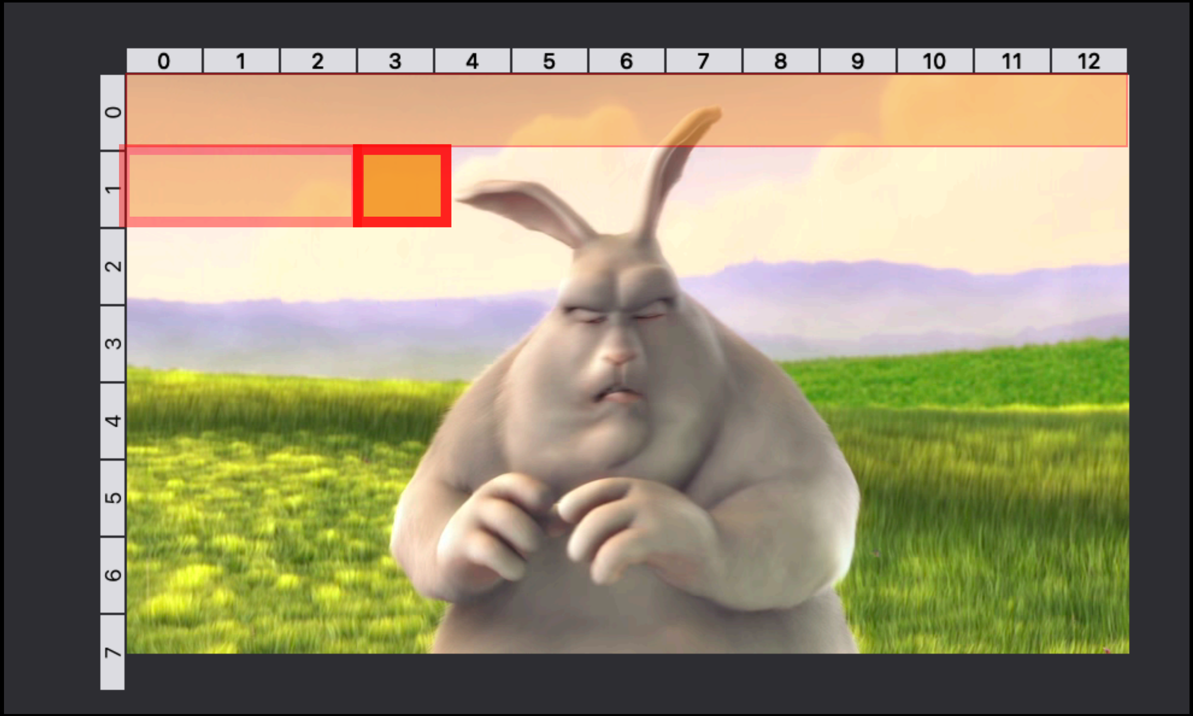
Image Compression -> BPG

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

Image from Kodak dataset

BPG/H.265-Iframe

Larger blocks are allowed (64x64), (32x32)



BPG/H.265-Iframe

BPG/H.265-Iframe

Larger blocks are allowed (64x64), (32x32)

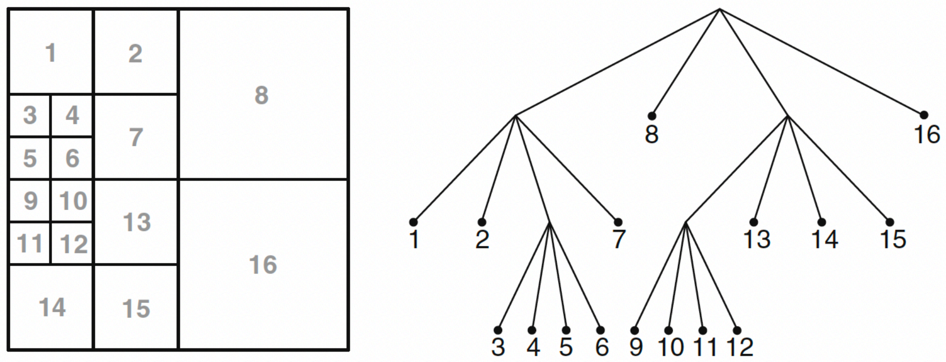
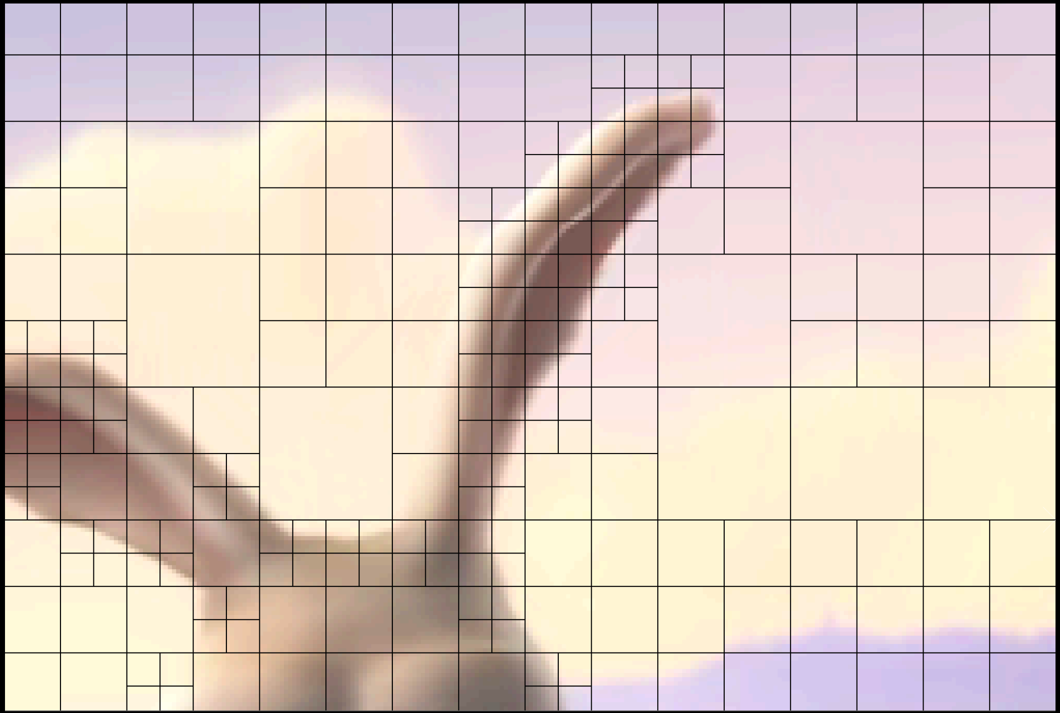


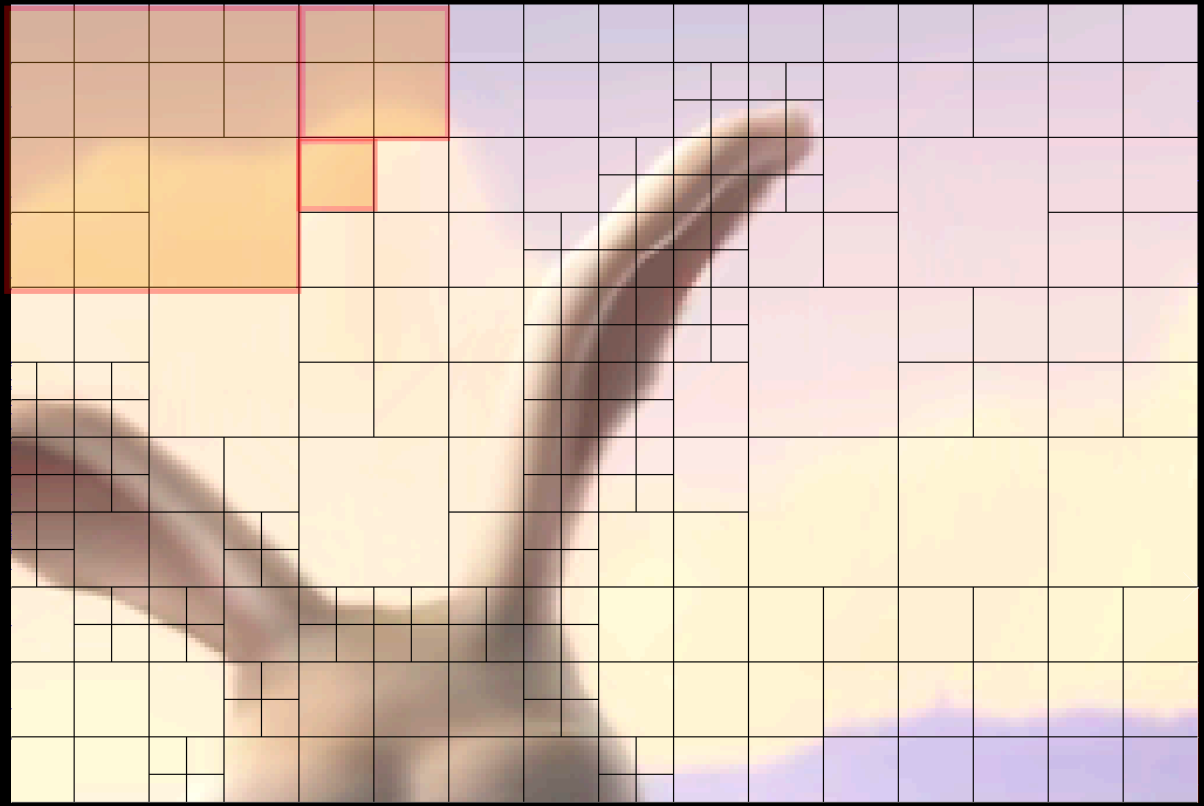
Fig. 3.4 Example for the partitioning of a 64 × 64 coding tree unit (CTU) into coding units (CUs) of 8 × 8 to 32 × 32 luma samples. The partitioning can be described by a quadtree, also referred to as coding tree, which is shown on the *right*. The numbers indicate the coding order of the CUs

BPG/H.265-Iframe

Larger blocks are allowed

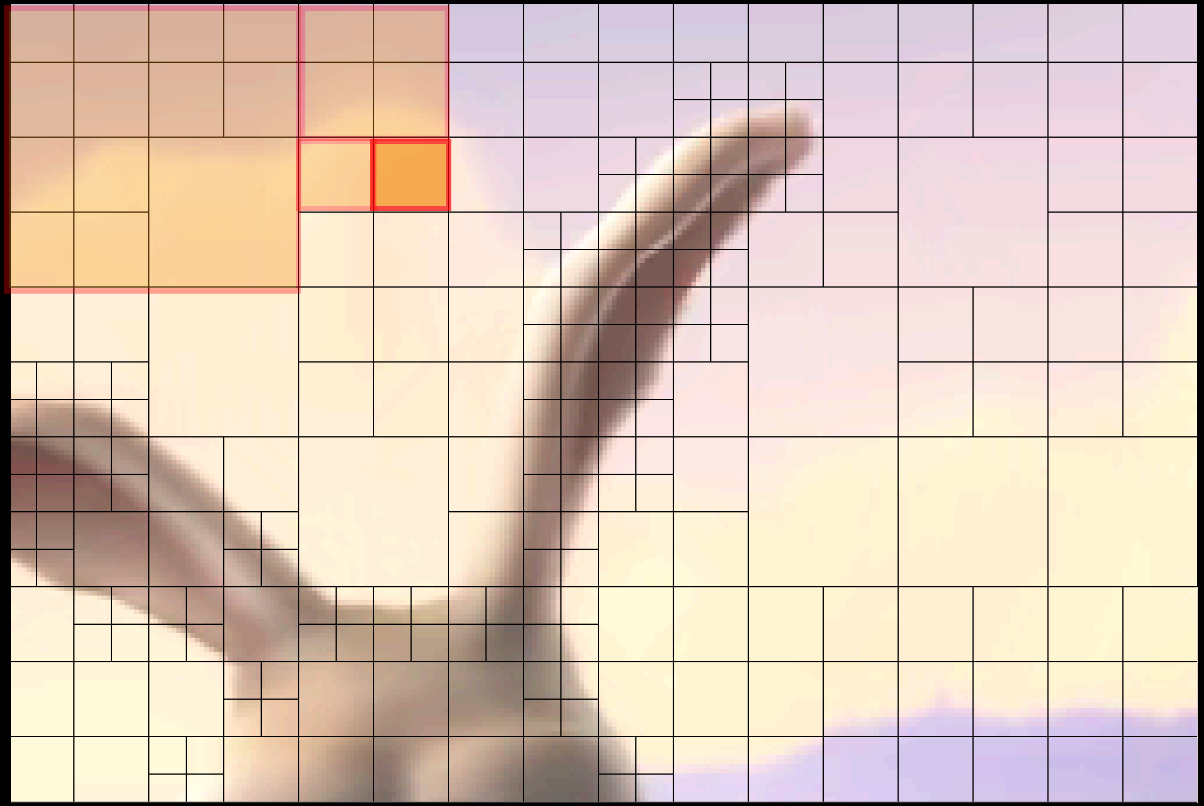


Predictive coding -> BPG/H.265



Predictive coding -> BPG/H.265

Predict next block, based on previously encoded blocks



BPG Prediction modes

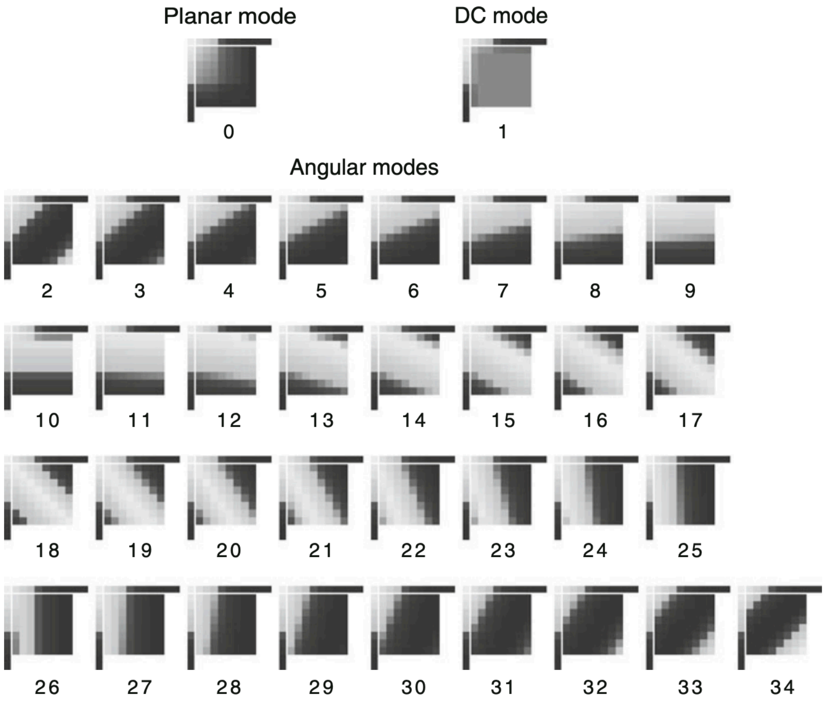
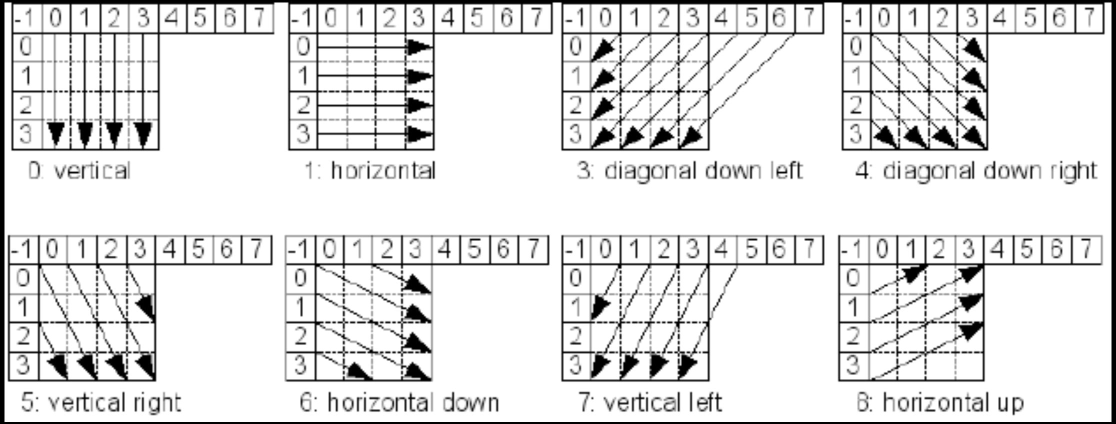


Fig. 4.2 Examples of 8 × 8 luma prediction blocks generated with all the HEVC intra prediction modes. Effects of the prediction post-processing can be seen on the top and left borders of the DC prediction (mode 1), top border of horizontal mode 10 and left border of vertical mode 26

- For simplicity (and speed) you only use the border pixels of the encoded blocks to predict the next block.
- Try multiple models, and use whichever works best

BPG Prediction modes

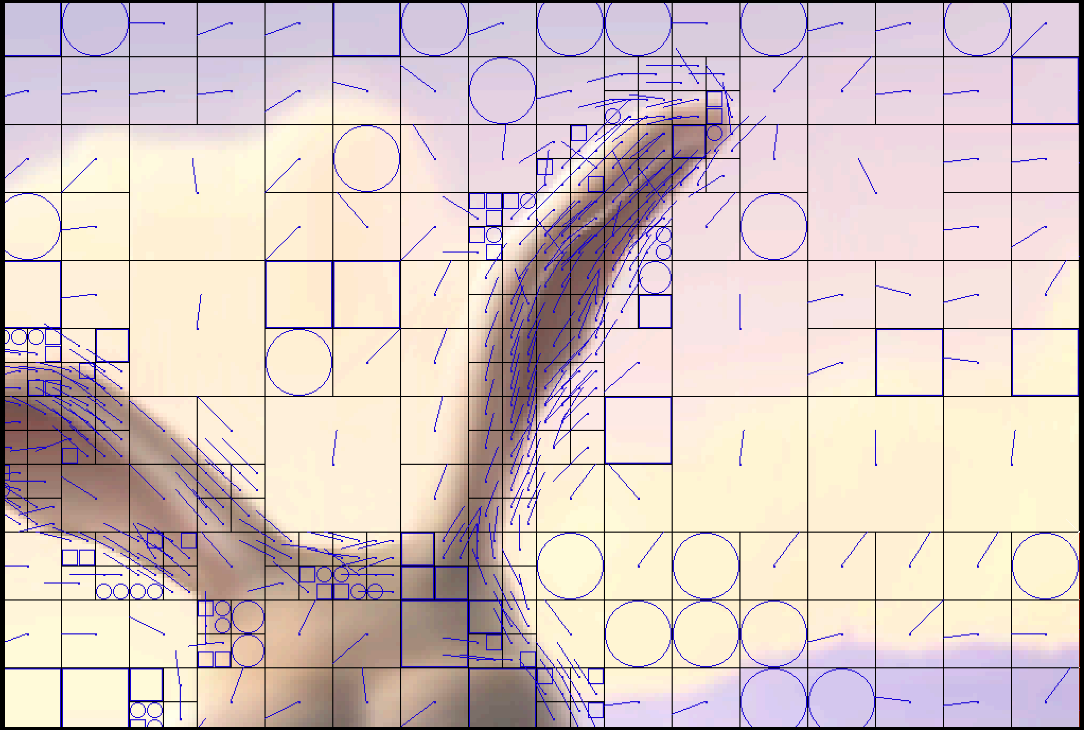


- For simplicity (and speed) you only use the border pixels of the encoded blocks to predict the next block.

- Try multiple models, and use whichever works best

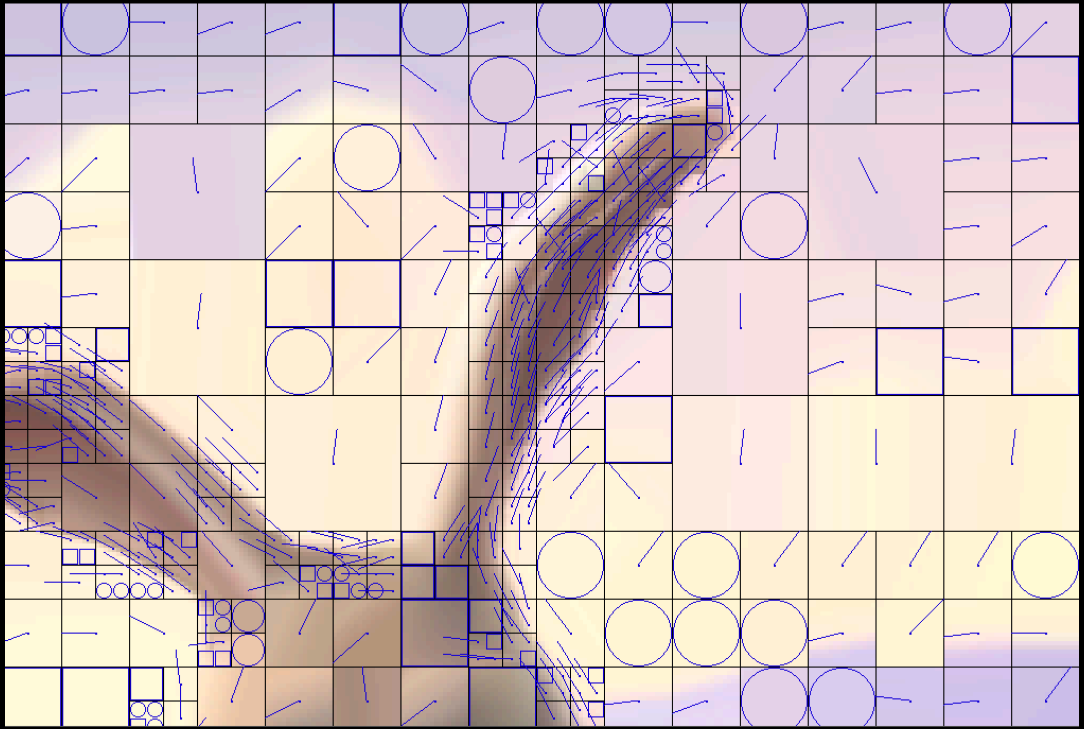
BPG Predictive coding

Blocks are not independent anymore!
Predictive coding



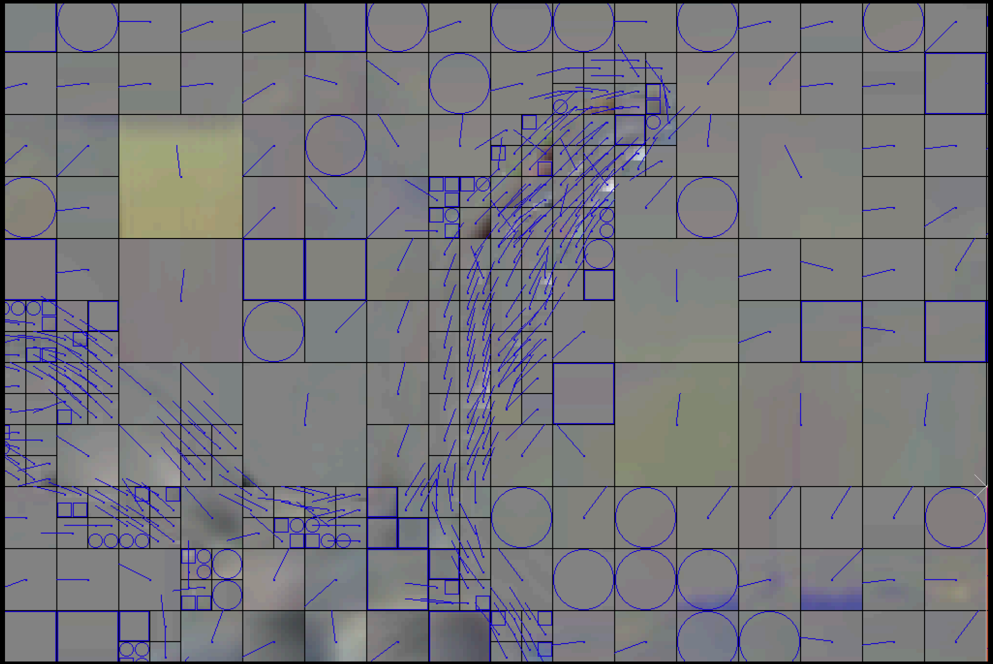
BPG Predictive coding

Blocks are not independent anymore!
Predictive coding



BPG Predictive coding

Blocks are not independent anymore!
Predictive coding



BPG

- 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.

BPG -> CABAC

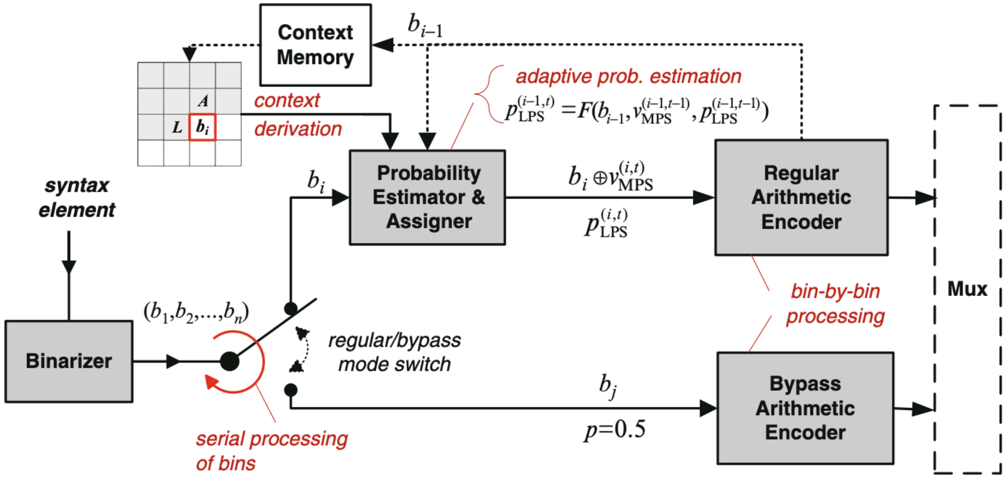


Fig. 8.1 CABAC block diagram (from the encoder perspective): Binarization, context modeling (including probability estimation and assignment), and binary arithmetic coding. In *red*: Potential throughput bottlenecks, as further discussed from the decoder perspective in Sect. 8.3.2

Image Compression -> JPEG 137x

Image from Kodak dataset

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

Image Compression -> BPG

Image from Kodak dataset

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

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 the R-D optimization is not accurate. Rate needs to be shared between different channels etc. Can we make that end-to-end?

https://wave-one.github.io/iframe_comparisons/

Questions?