Image Compression

L15, EE274, Fall 23

Announcements

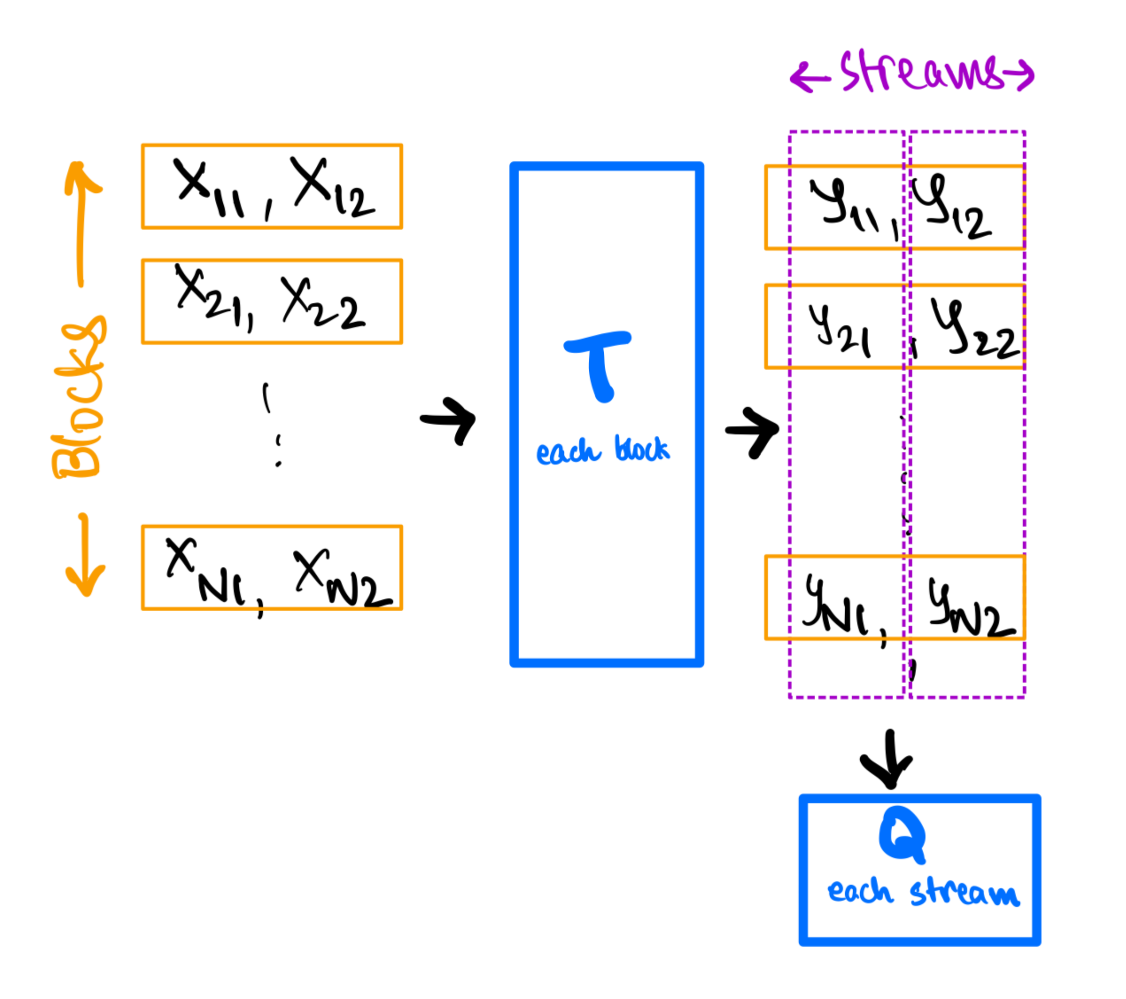
- HW3 due today!
  - Check clarifications thread on Ed if struggling with P1/P4
- Coming up this week:
  - HW4 — released this Fri, will be due Mon 12/4
- Thanksgiving break next week — no classes
- Start working on your projects!
  - Milestone due Mon right after thanksgiving break: 11/27
    - Details on website project page
  - Final presentations in last week of class: 12/6

Recap

1. Transform Coding

   KLT example

   Two knobs!



Quiz Q1

In which of the following cases do you expect vector quantization to improve the lossy compression performance?
(select all the correct options)

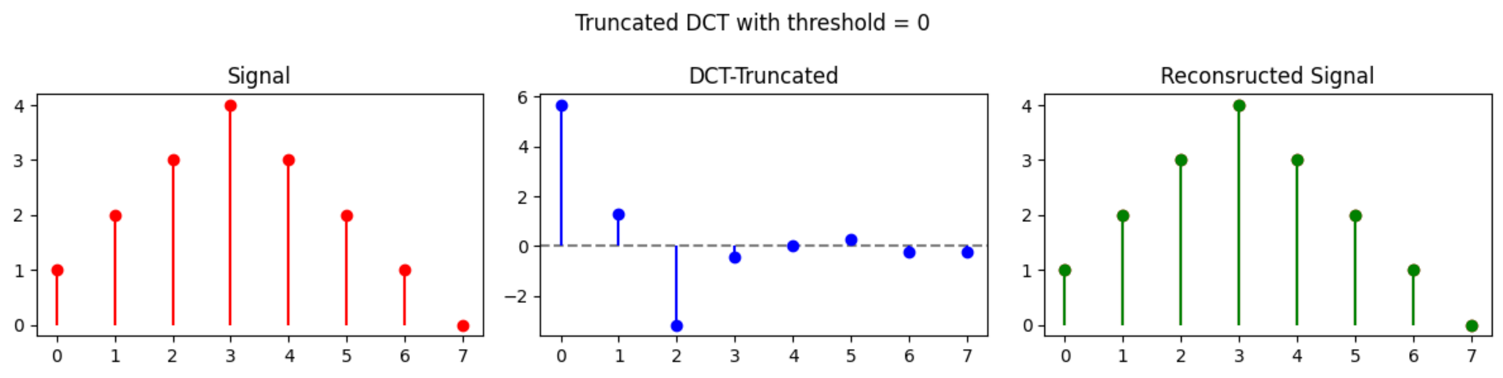
[ ] i.i.d. data compressed with scalar quantization
[ ] non-i.i.d. (correlated) data with scalar quantization

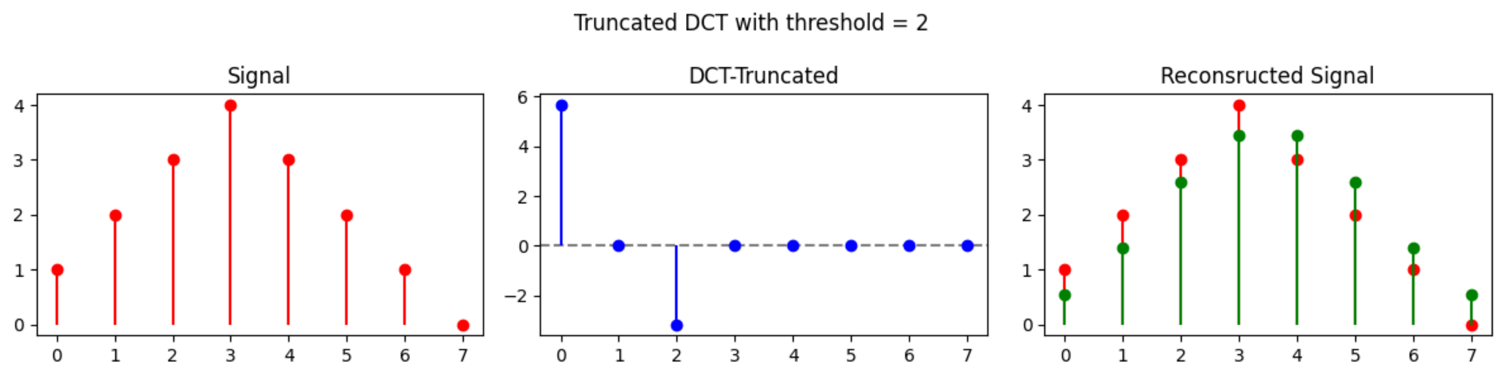
In which of the following cases do you expect transform coding to improve the lossy compression performance?
(select all the correct options)

[ ] i.i.d. data
[ ] non-i.i.d. (correlated) data

Recap

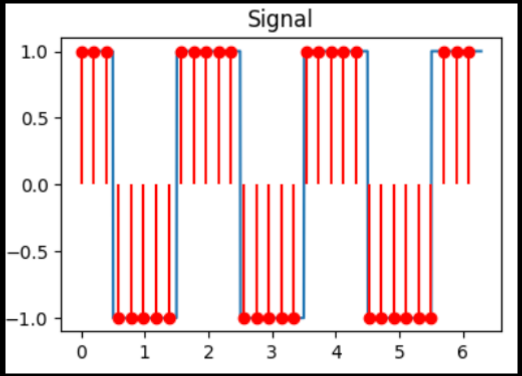
2. DCT

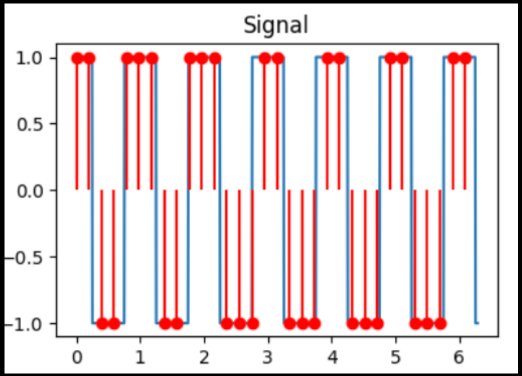

Truncated DCT with threshold = 0
Signal
DCT-Truncated
Reconsructed Signal

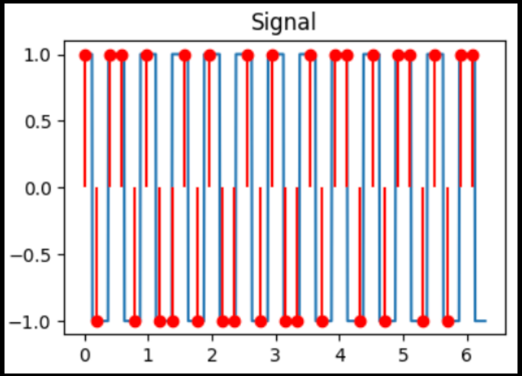

Truncated DCT with threshold = 2
Signal
DCT-Truncated
Reconsructed Signal

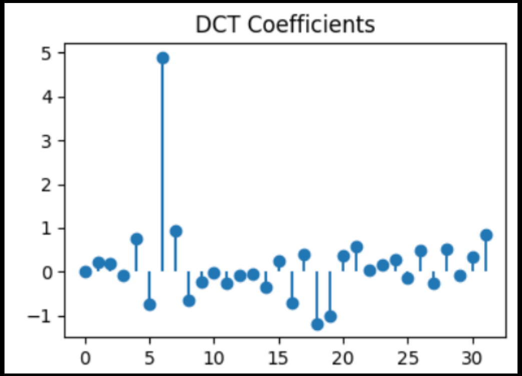
Quiz Q2

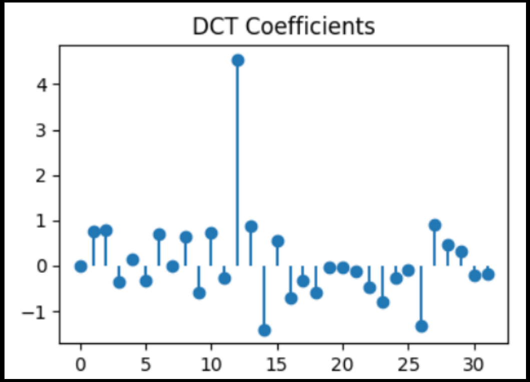
Match the signals to their DCT!

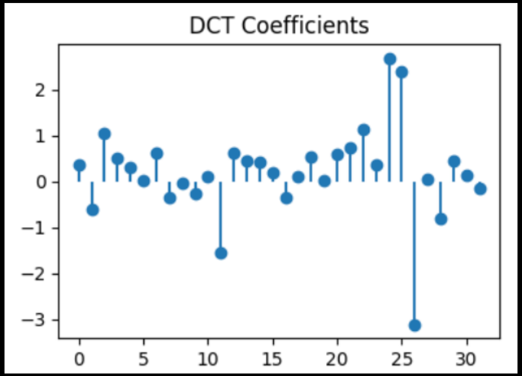






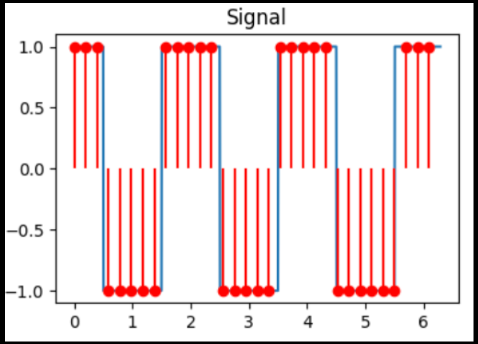


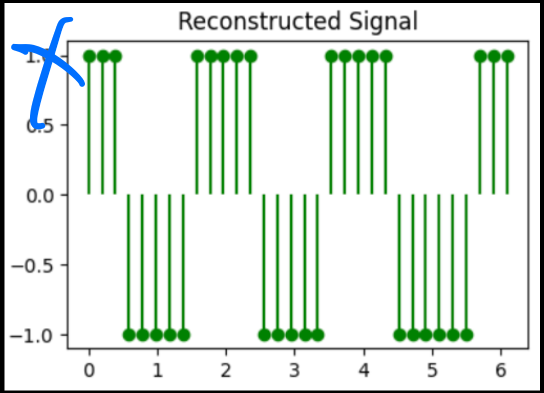


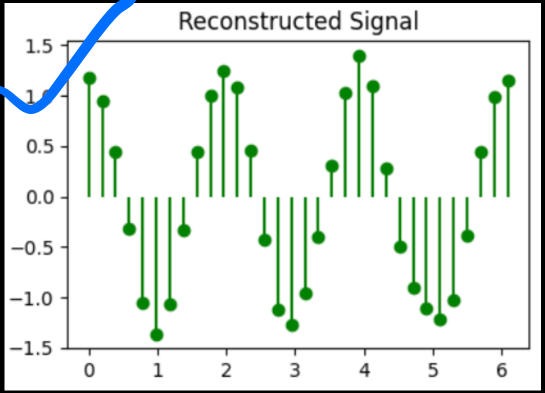


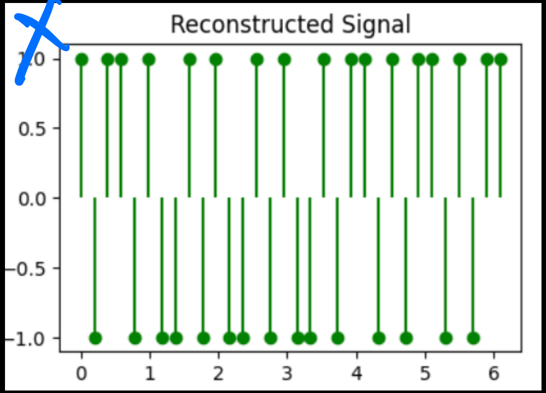
Quiz Q3

For the signal shown above, we take the DCT and truncate (zero out) the 16 highest frequencies (out of 32 total components in the DCT). Identify the reconstructed signal obtained after performing the inverse DCT.



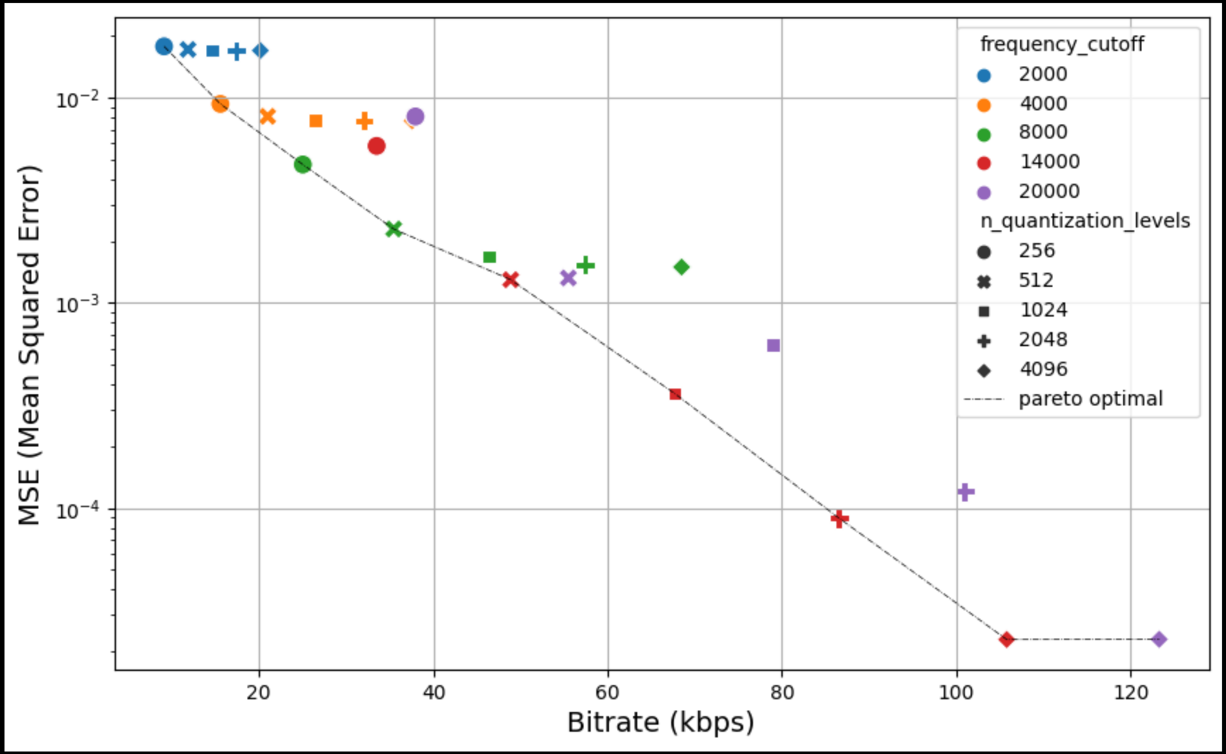






Recap

3. Audio Compression



Today’s Lecture

Image Compression!

- Bring together everything we have learnt so-far

- Learn about JPEG basics

Slide resources

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

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

What is an image?

What is an image?

Array of pixels: (Height, Width, Channels)

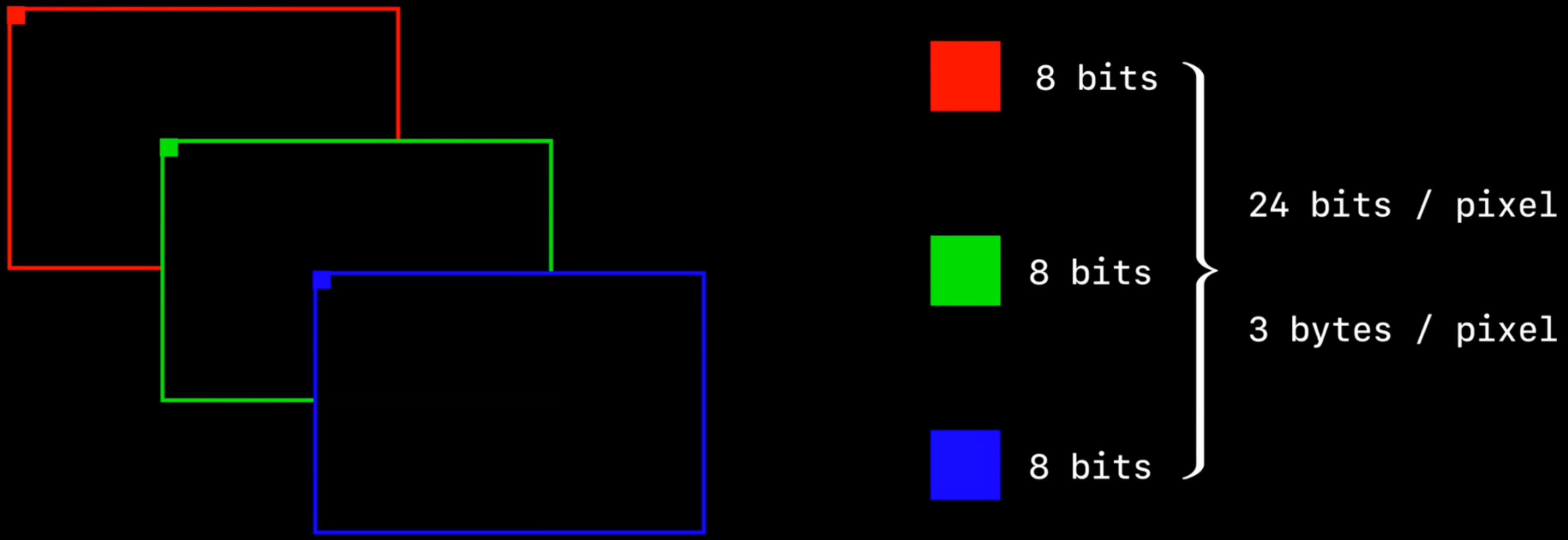


Image Compression

Image from Kodak dataset

764x512x3 bytes
= 1.1MB!
(Uncompressed)

H = 764

W = 512

C = 3

1 byte

Image Compression -> JPEG 40x

Image from Kodak dataset

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

Image Compression -> JPEG 80x

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

Image from Kodak dataset

Image Compression -> JPEG 137x

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

Image from Kodak dataset

Image Compression -> BPG

Image from Kodak dataset

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

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, e.g. 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 …

JPEG Image Compression

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

THE JPEG STILL PICTURE COMPRESSION STANDARD

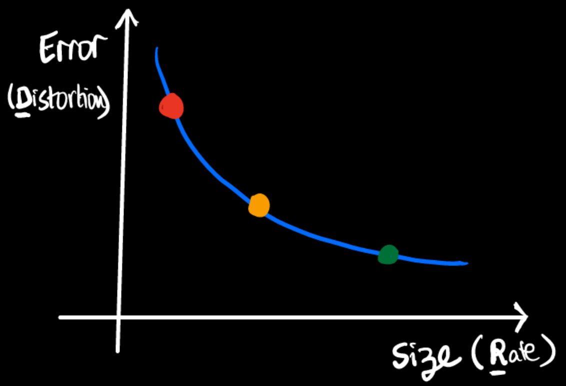
Gregory K. Wallace
Multimedia Engineering
Digital Equipment Corporation
Maynard, Massachusetts

commonplace in general-purpose computing systems the way text and geometric graphics are. The majority of modern business and consumer usage of photographs and other types of images takes place through more traditional analog means.

The key obstacle for many applications is the vast amount of data required to represent a digital image

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

Rate-Distortion Tradeoff



585 KB

280 KB

13 KB

What is the distortion metric?

Lossy Compression: Problem definition

Distortion metric -> MSE?

MSE - basis for much of rate-distortion theory!

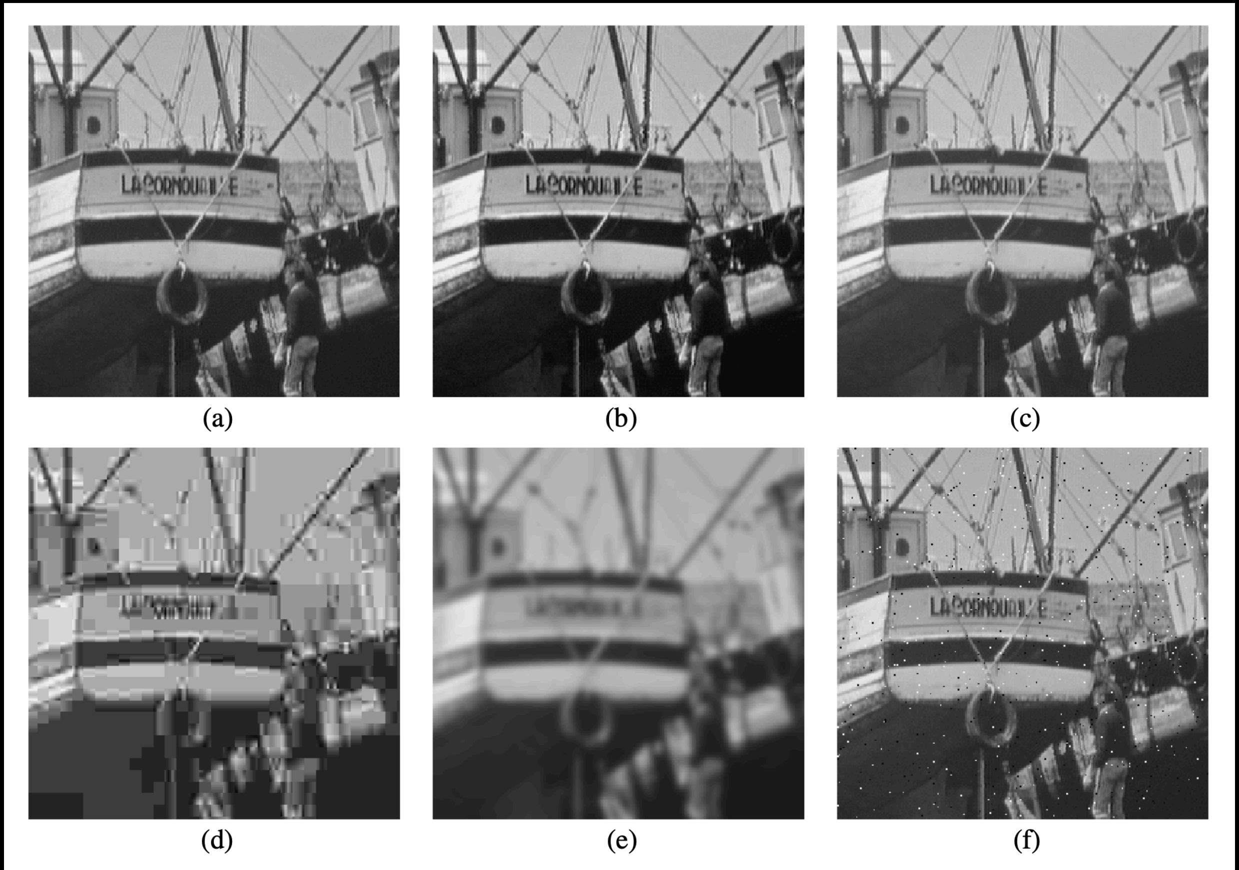
$$\mathrm{MSE} = \overbrace{\boxed{\frac{1}{n}\sum_{i=1}^{n}}}^{\text{Mean}} (\overbrace{\boxed{Y_i - \hat{Y}_i}}^{\text{Error}})^{\overbrace{\boxed{2}}^{\text{Squared}}}$$

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

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

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

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



Lossy Compression -> Problem definition

Distortion metric -> ~~MSE~~?

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

For simplicity, we will consider MSE (+heuristics)

Lossy Compression: Problem caveats

- Typically we care about compressing a single image, and not a *group* of images

- Non-asymptotic performance of various techniques is important

- Data is most likely non-stationary:
  need to convert/transform appropriately

Lossy Compression: Tools we know

- Scalar Quantization: fast, not the best
- Vector Quantization: very good, but slow as dim increases
- Transform Coding: Decorrelate data, and then use simpler (scalar) quantization
- Predictive coding: Fancier delta coding

All are useful in different contexts!

Compress Image — what's the first obvious thought?

Array of pixels:

(H, W, C)

Compress Image — what's the first obvious thought?

$(H, W, C) \mapsto (H//D, W//D, C)$

- Downsample Original Image!

  e.g. (H//2, W//2, Channels)
  compresses by **4X**!

- Typically upsampled at the decoder to recover original array of (H, W, C)

- Very common in practice!

Vector Quantization / Color Quantization

Original Image

Simple Idea:
quantize the 3-dim vector (R,G,B)

8bit

$2^8 \cdot 2^8 \cdot 2^8$

R G B

Vector Quantization / Color Quantization

Simple Idea:
quantize the 3-dim vector (R,G,B)

Number of colors = 256
(3X compression!)

Vector Quantization / Color Quantization

Simple Idea:
quantize the 3-dim vector (R,G,B)

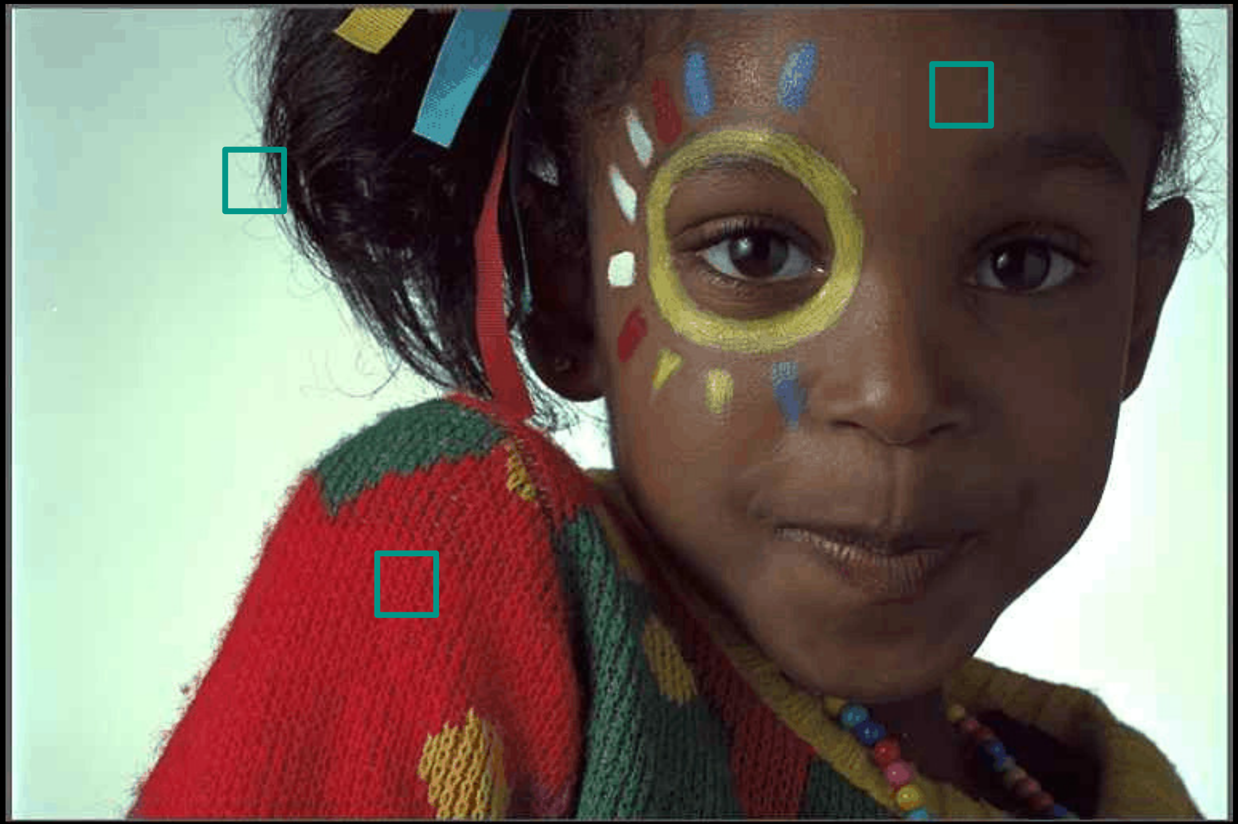
Number of colors = 16
(6X compression!)

Vector Quantization / Color Quantization

Number of colors = 256
(3x compression!)

Q: *How can we further improve compression?*

Vector Quantization / Color Quantization



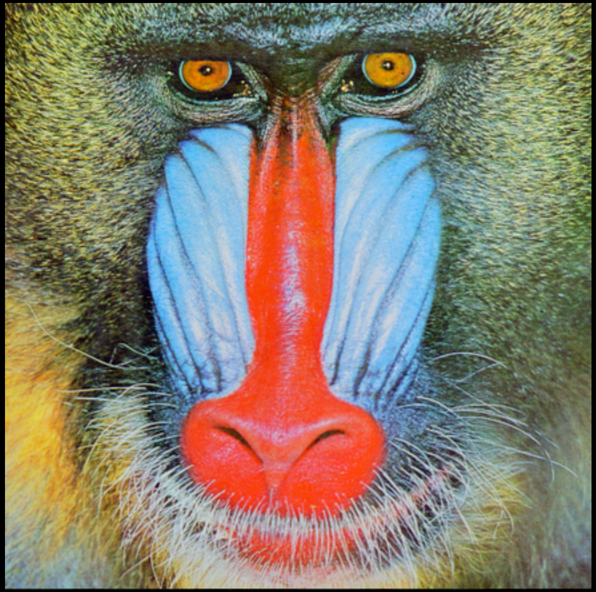
Vector Quantization / Color Quantization

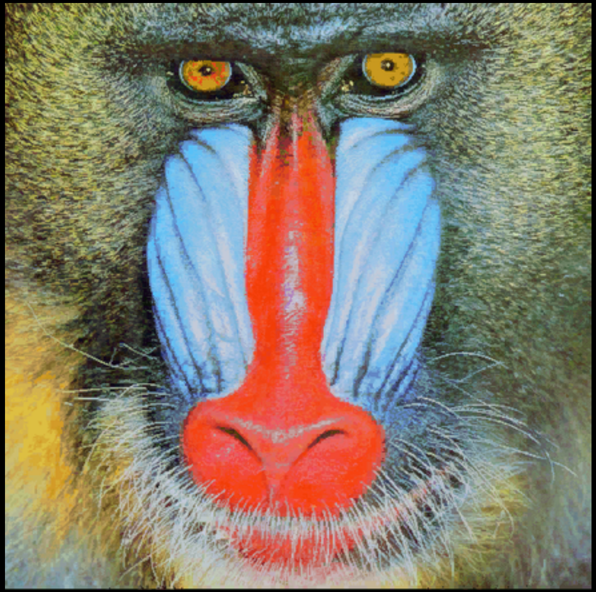
Number of colors = 256
(3X compression!)

***Q:** How can we further improve compression?*

Ans: Exploit “correlation” between neighboring pixels

Color Cell Compression





Use *Correlation* between neighboring pixels

32b/(4x4)

= 2b/pixel

Color Cell Compression (1984): use 2 colors (among 256) for each 4x4 block

Uses 16 (=8+8) bits/block for storing colors and a bit/pixel to decide which color to use for that pixel
Effectively 2 bits/pixel = 12X compression!

Color Cell Compression

Use *Correlation* between neighboring pixels

Color Cell Compression (1984): use 2 colors (among 256) for each 4x4 block

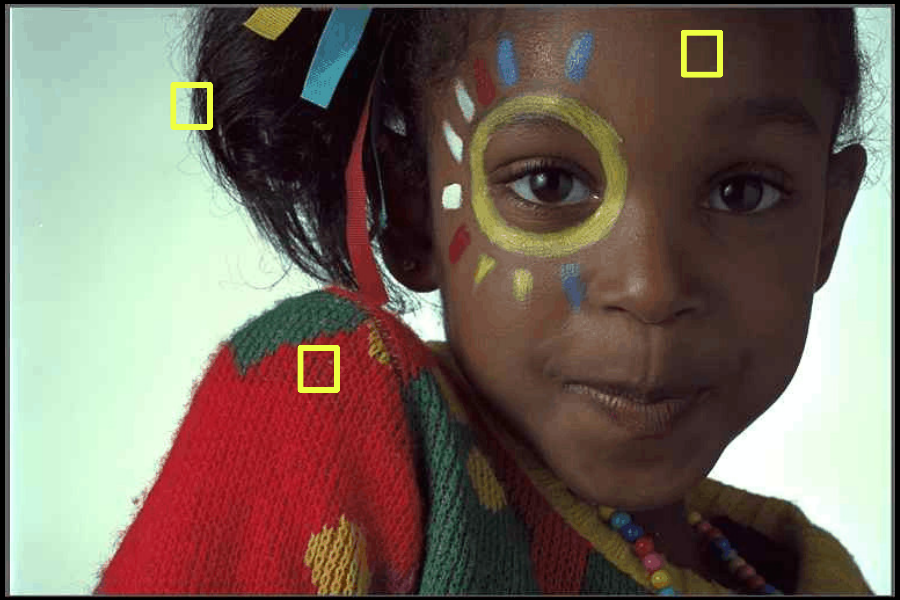
Uses 16 (=8+8) bits/block for storing colors and a bit/pixel to decide which color to use for that pixel
Effectively 2 bits/pixel = 12X compression!

Exploiting Spatial correlation in the data

Key Idea:
We need to somehow exploit/remove the correlation between neighboring pixels

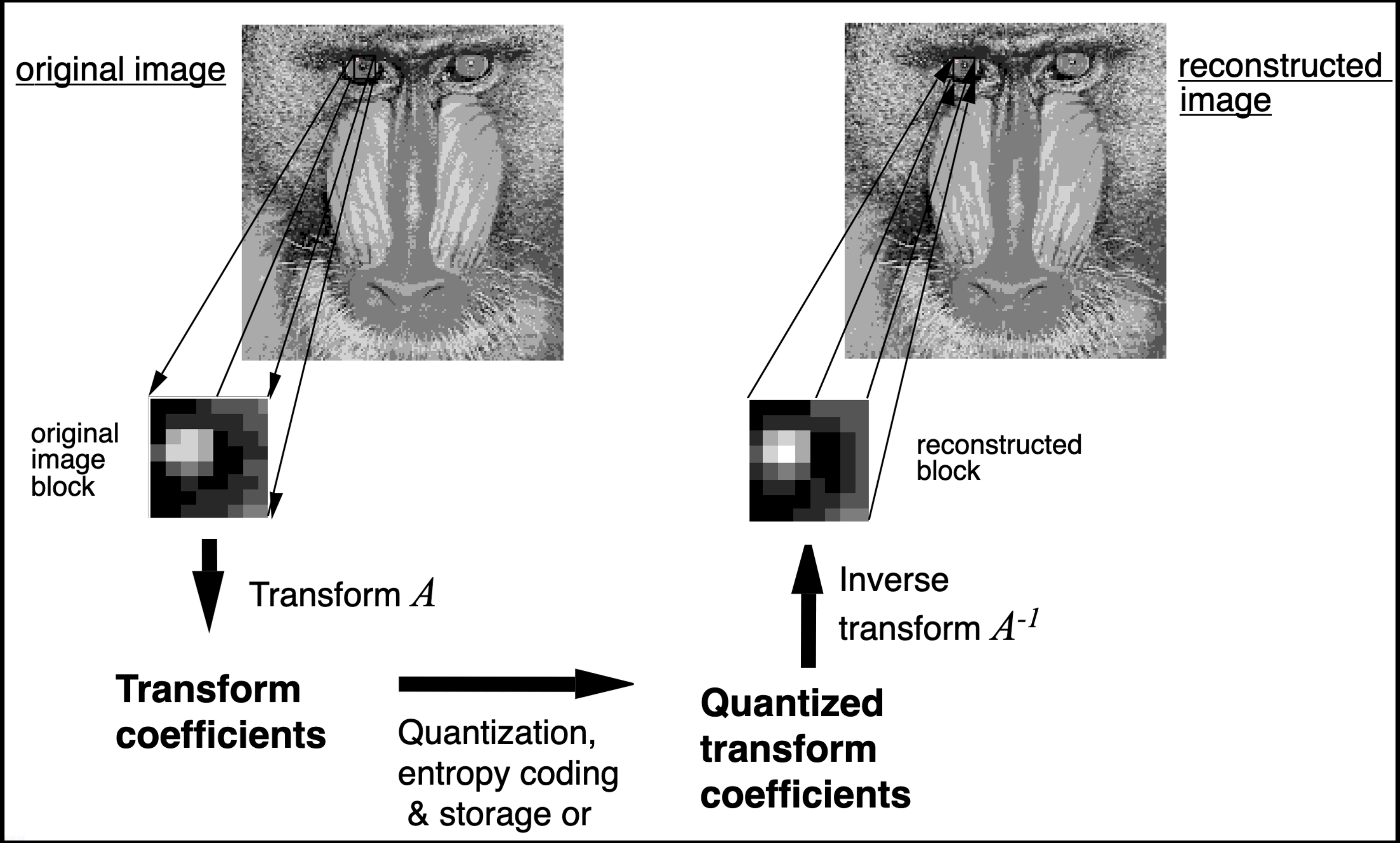
Exploiting Spatial correlation in the data

Key Idea:
We need to somehow exploit/remove the correlation between neighboring pixels



TRANSFORM CODING!

Block Transform Coding



Linear Transform Coding

- Forward transform

T

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

NxN transform coefficients, arranged as a column vector

Transform matrix of size N^2xN^2

Image block of size NxN, arranged as a column vector

N^2

- Inverse transform

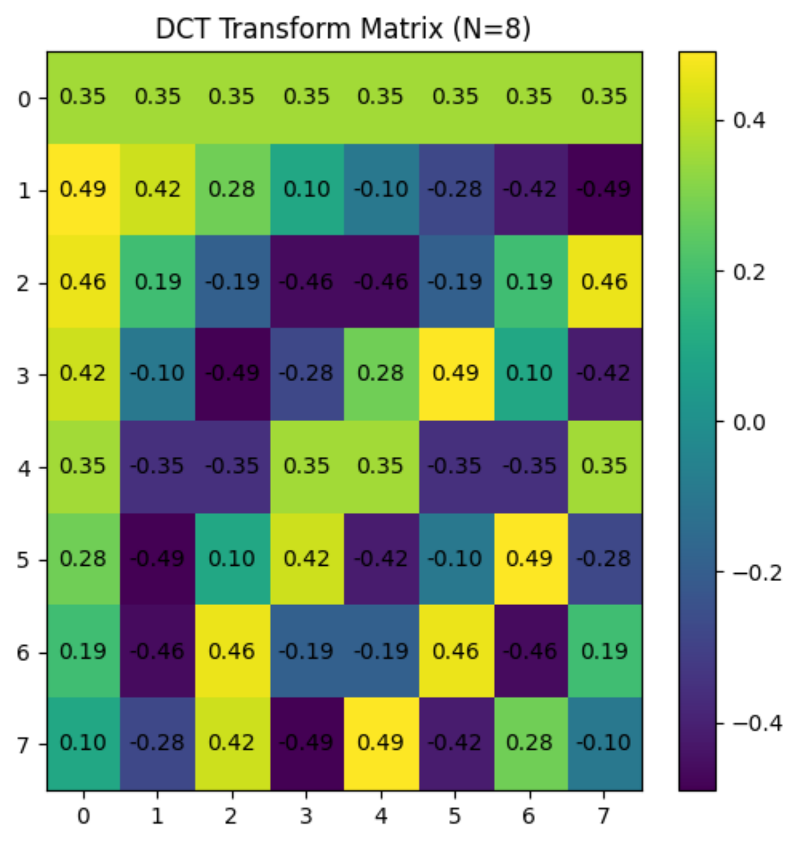
$$\boxed{\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} = \mathbf{A}^{T}\mathbf{y}}$$

Recall 1D-DCT vectors

Average value
(DC component)

Higher
frequency
components

(AC components)

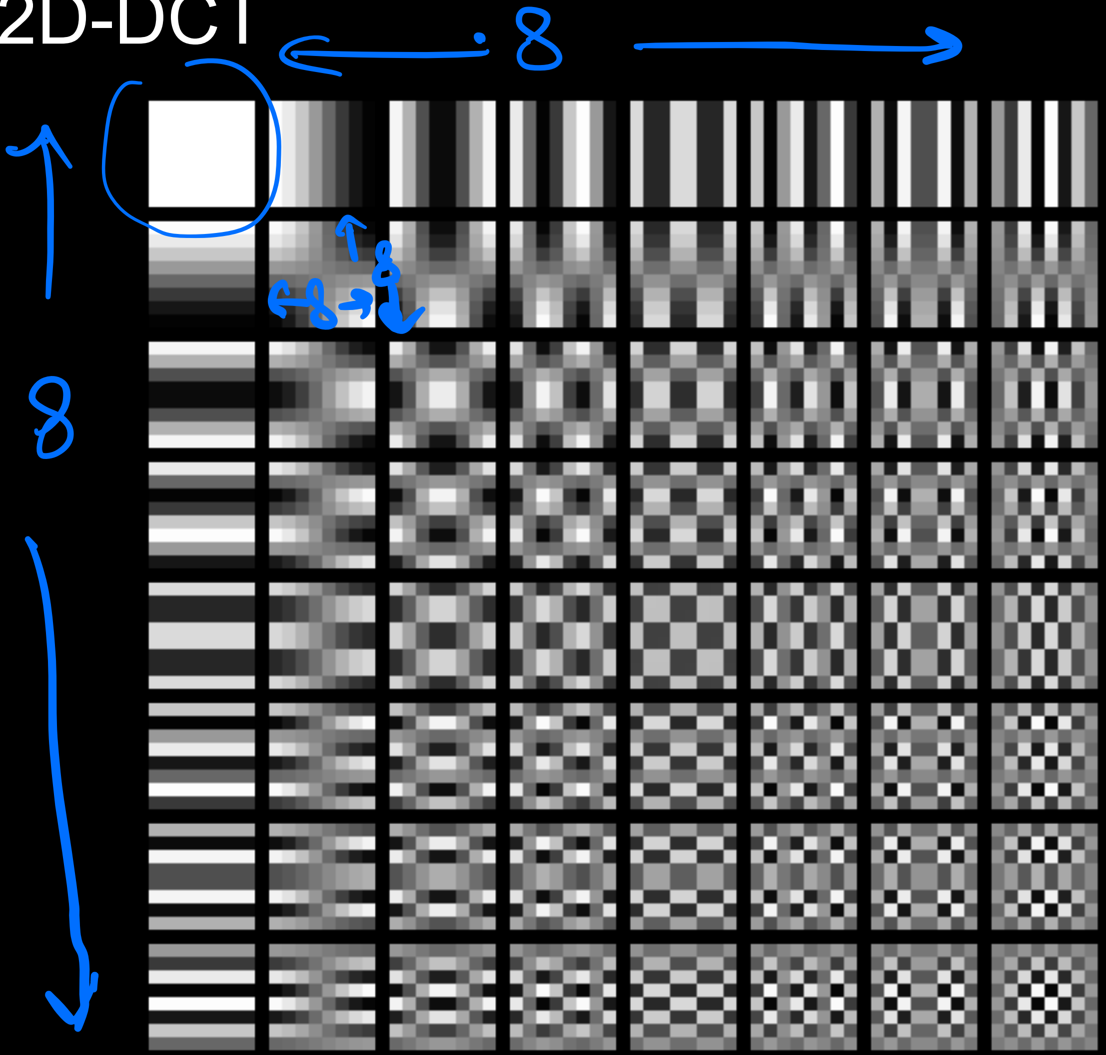


Block-Size (X) = N
DCT-Size = NxN

Transform Coding: 2D-DCT

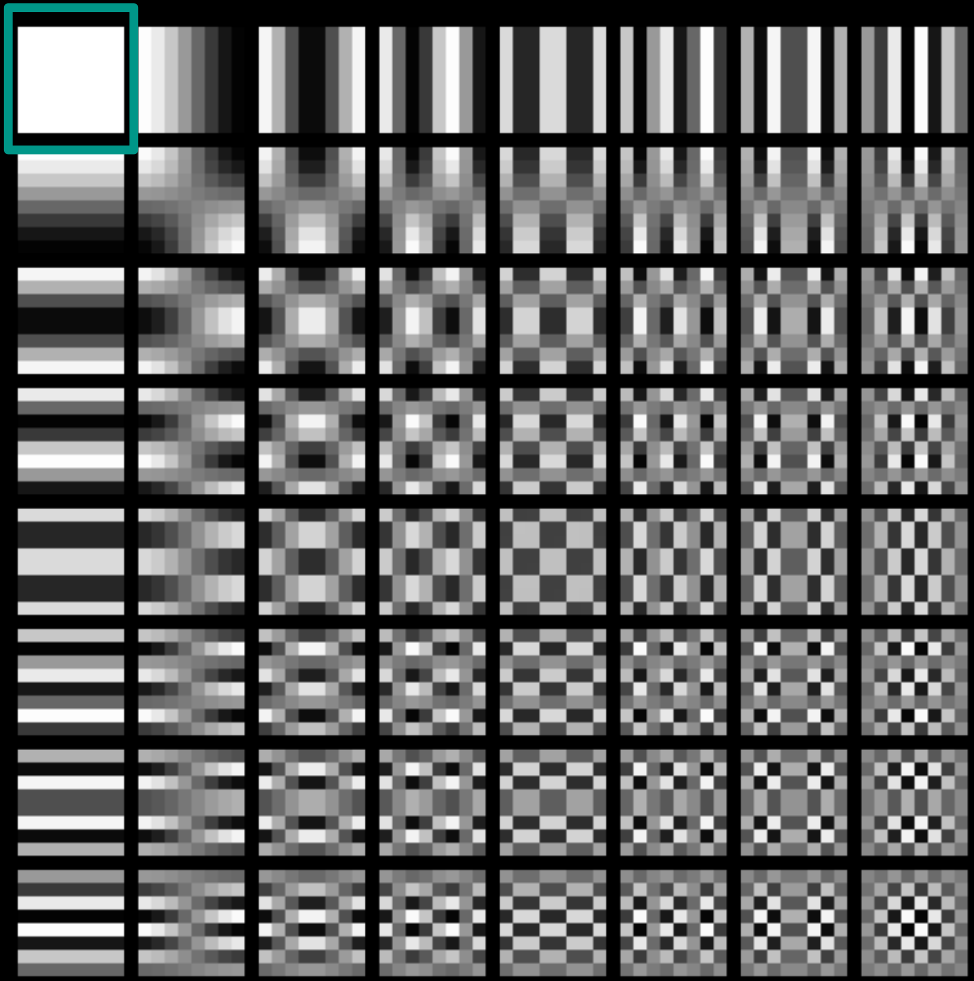
Block-Size (X) = NxN
DCT-Size = (NxN) x (NxN)

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

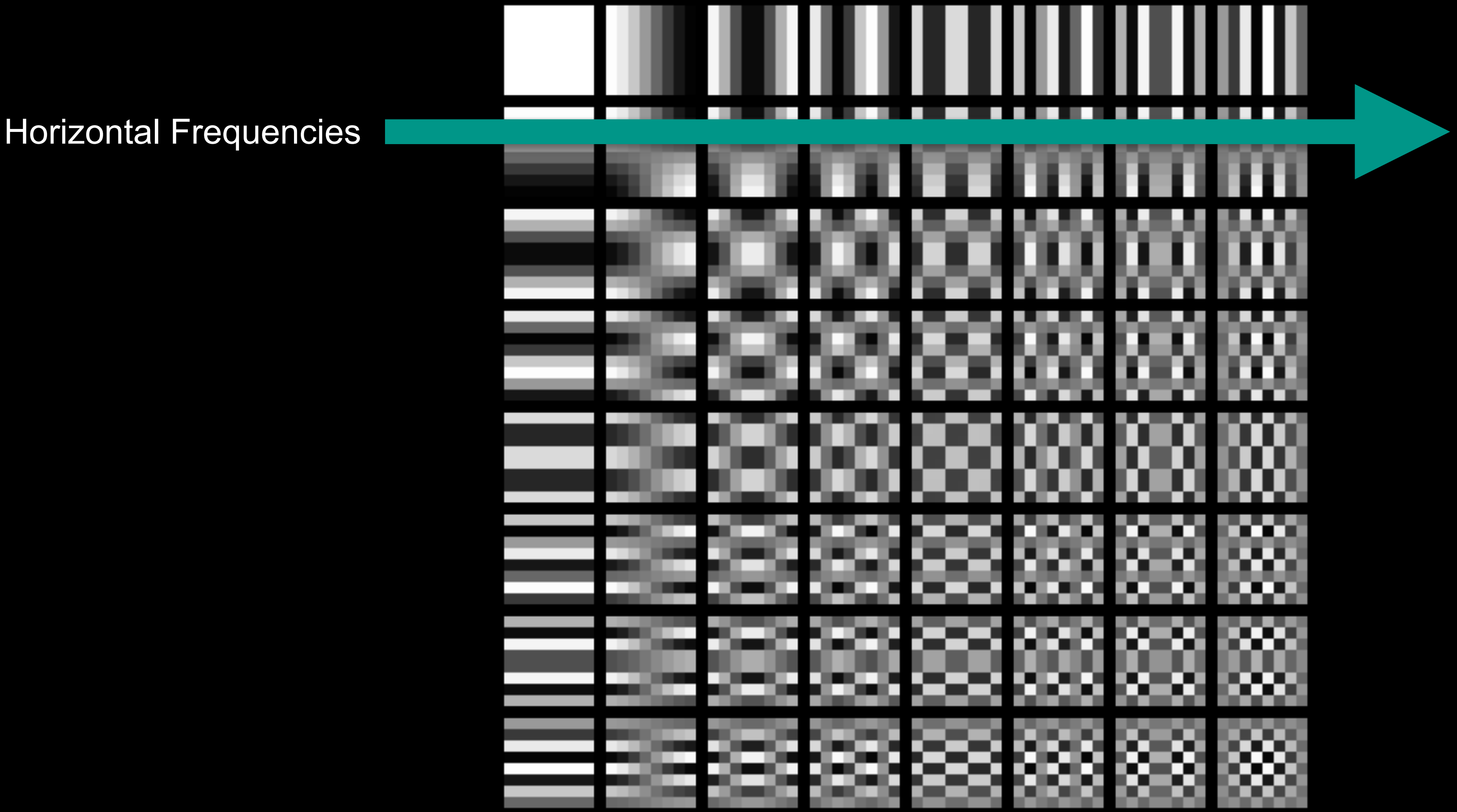


Transform Coding: 2D-DCT

Average value
(DC component)

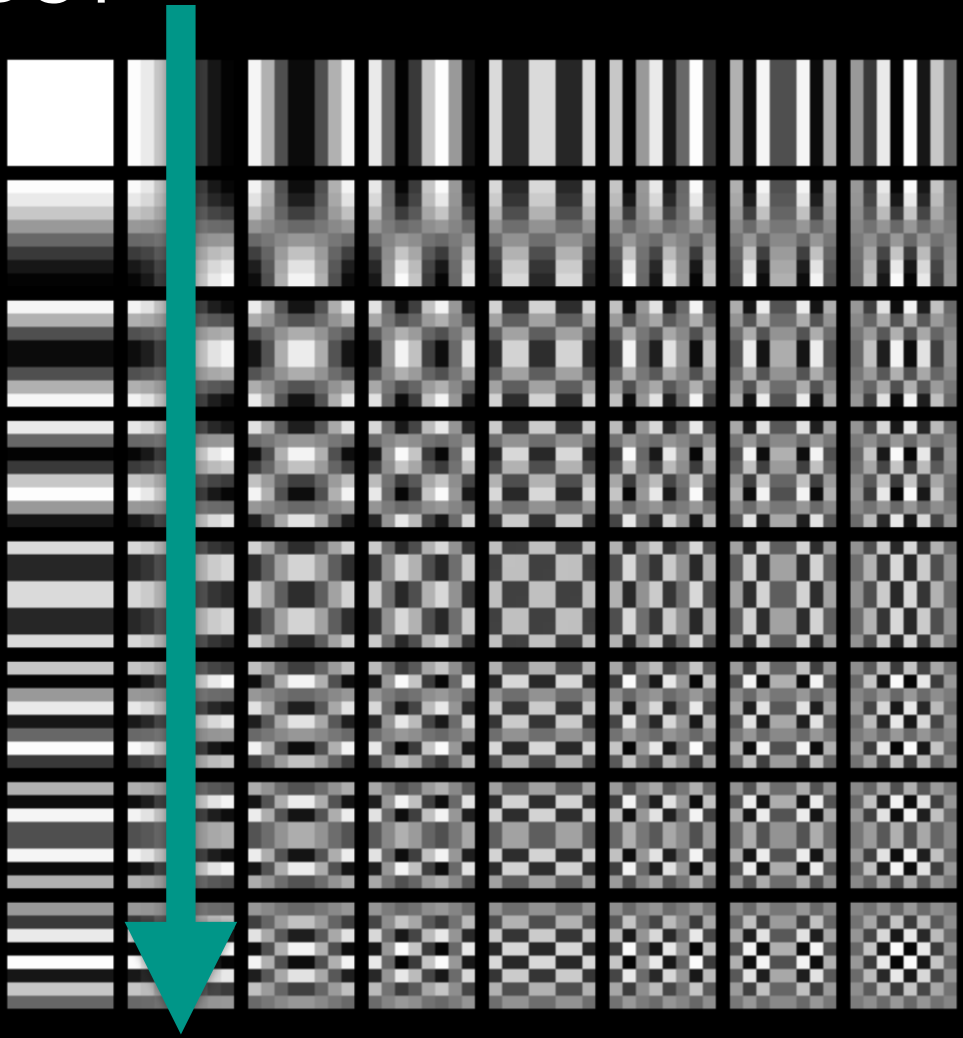


Transform Coding: 2D-DCT



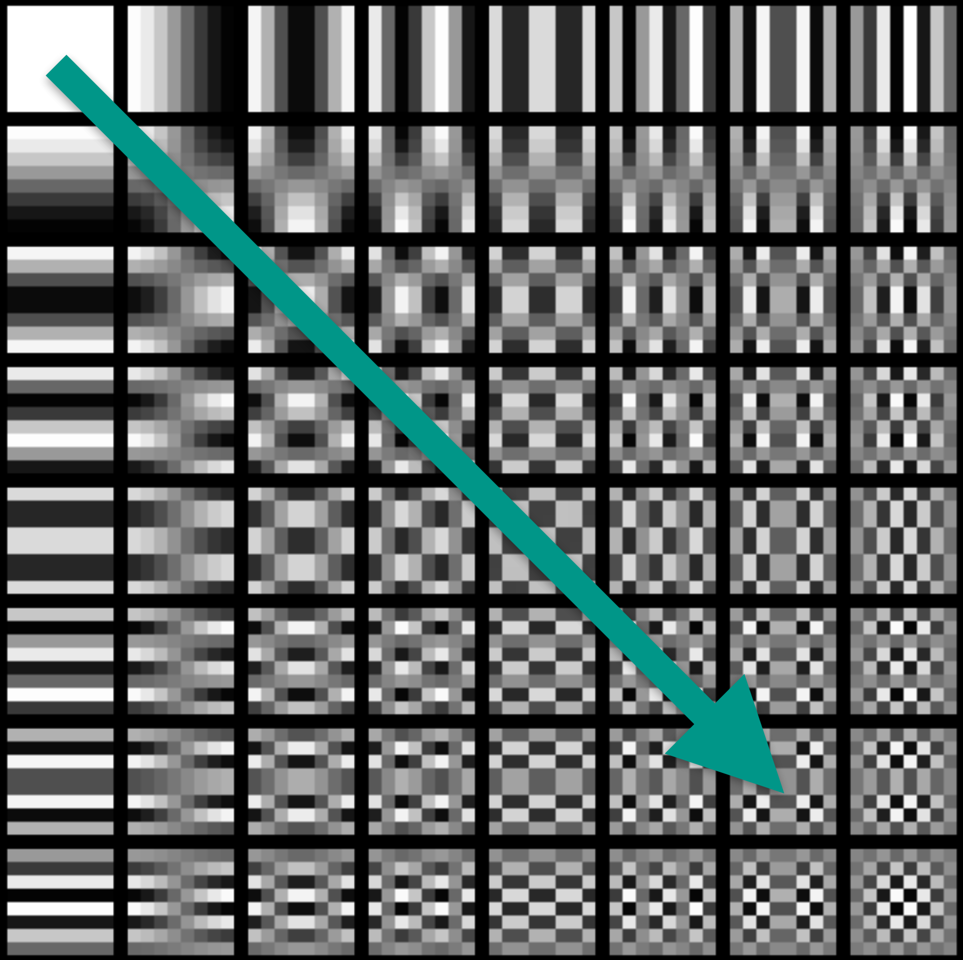
Transform Coding: 2D-DCT

Vertical Frequencies

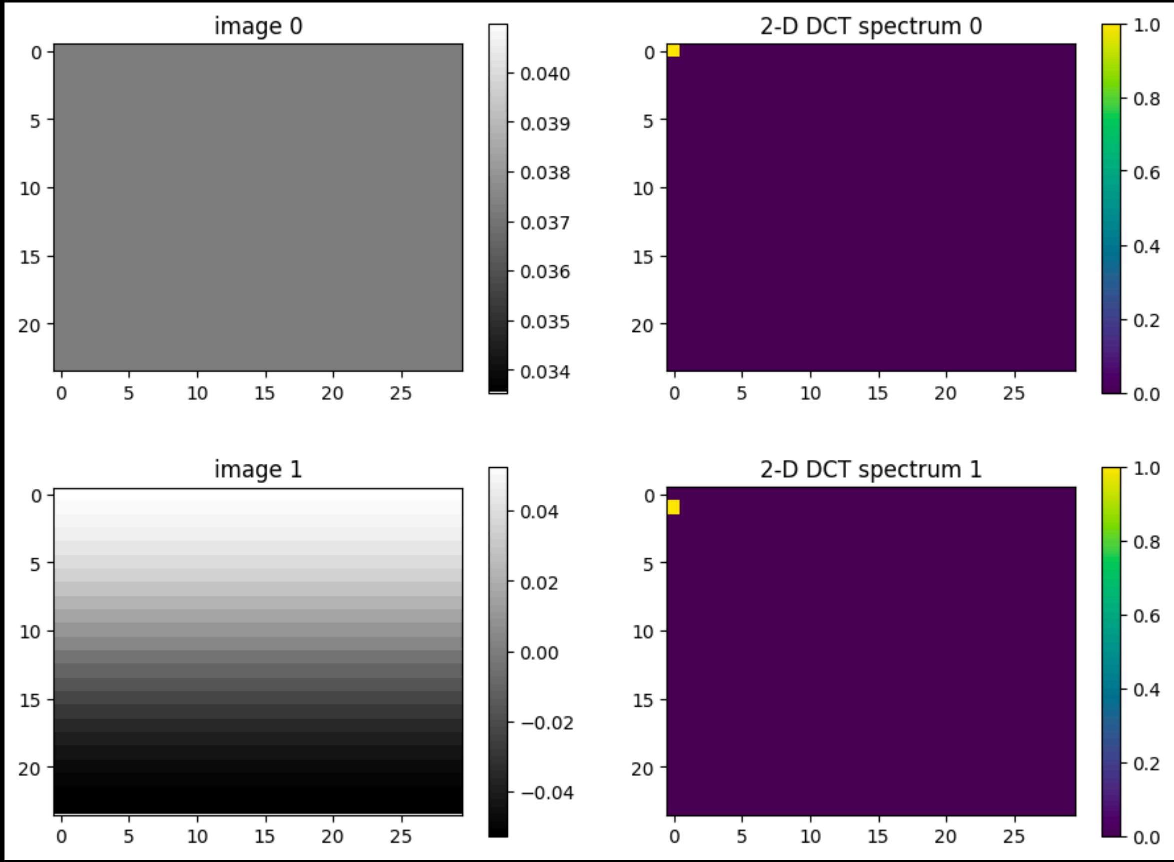


Transform Coding: 2D-DCT

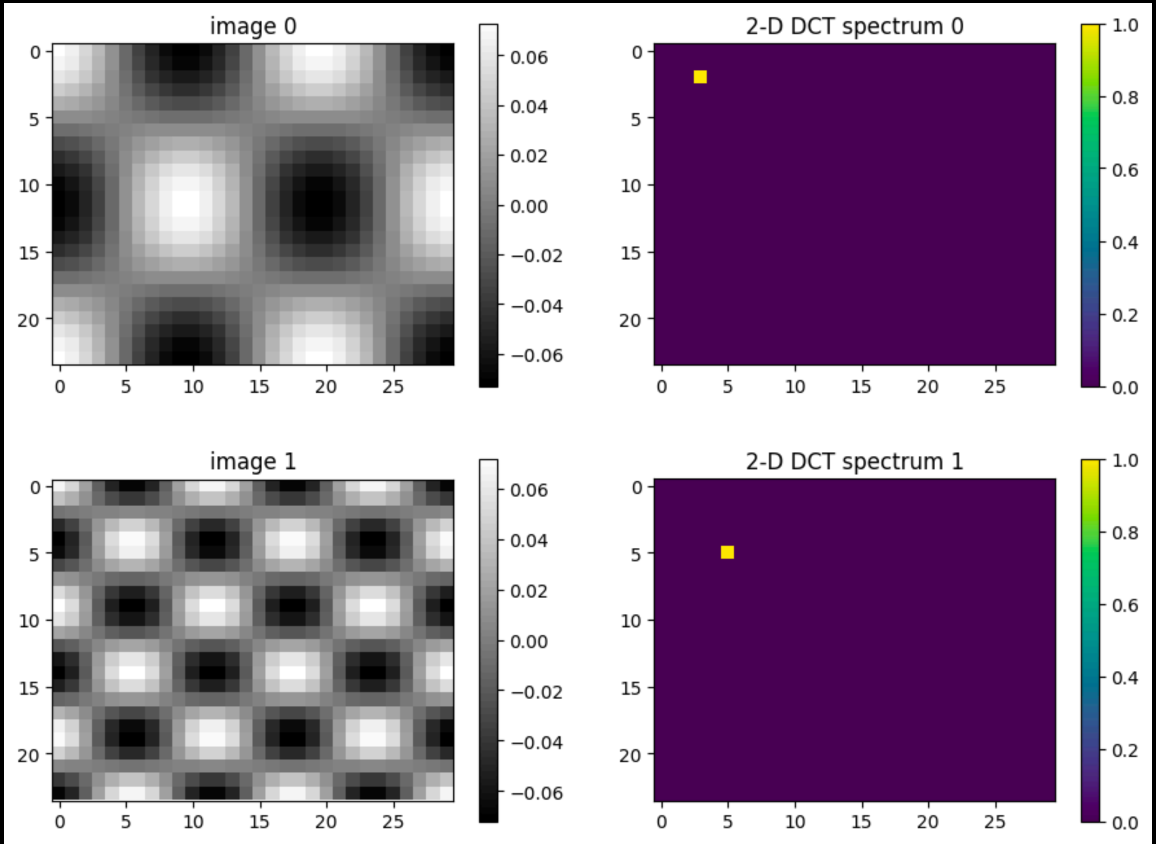
Horizontal + Vertical
Frequencies



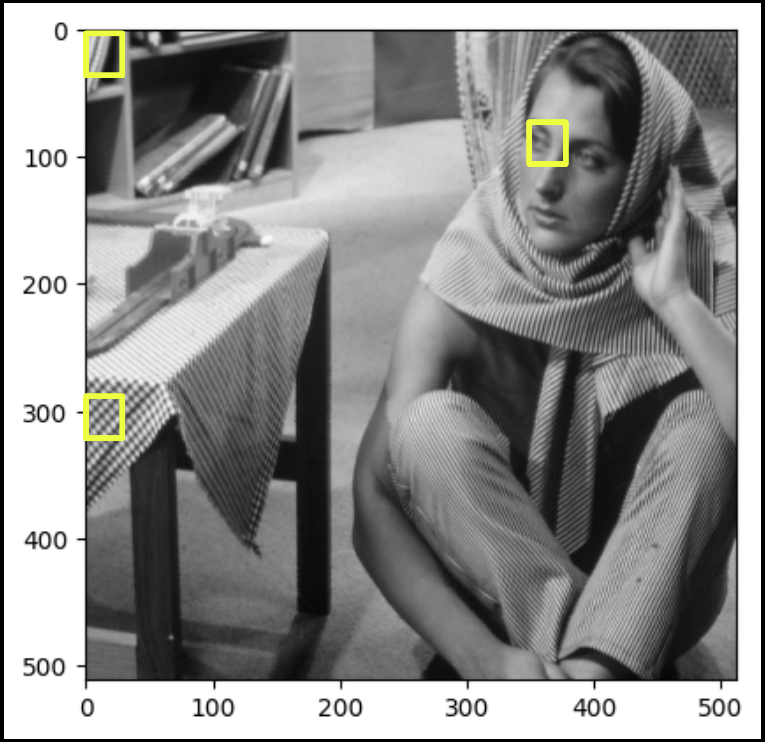
Transform Coding: DCT

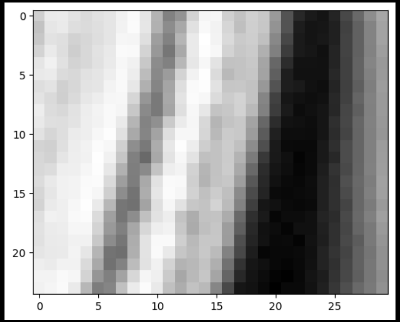


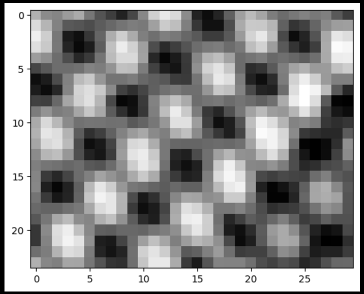
Transform Coding: DCT

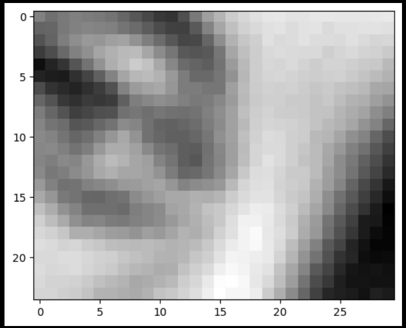


Transform Coding: DCT

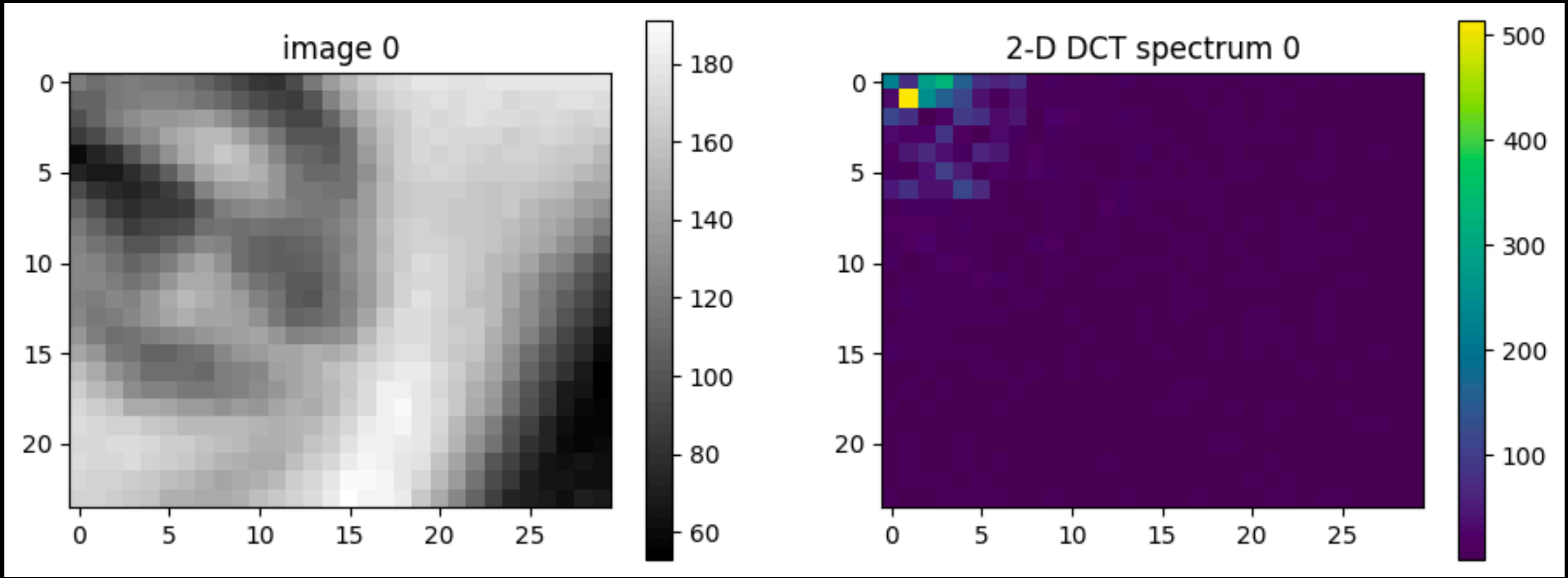






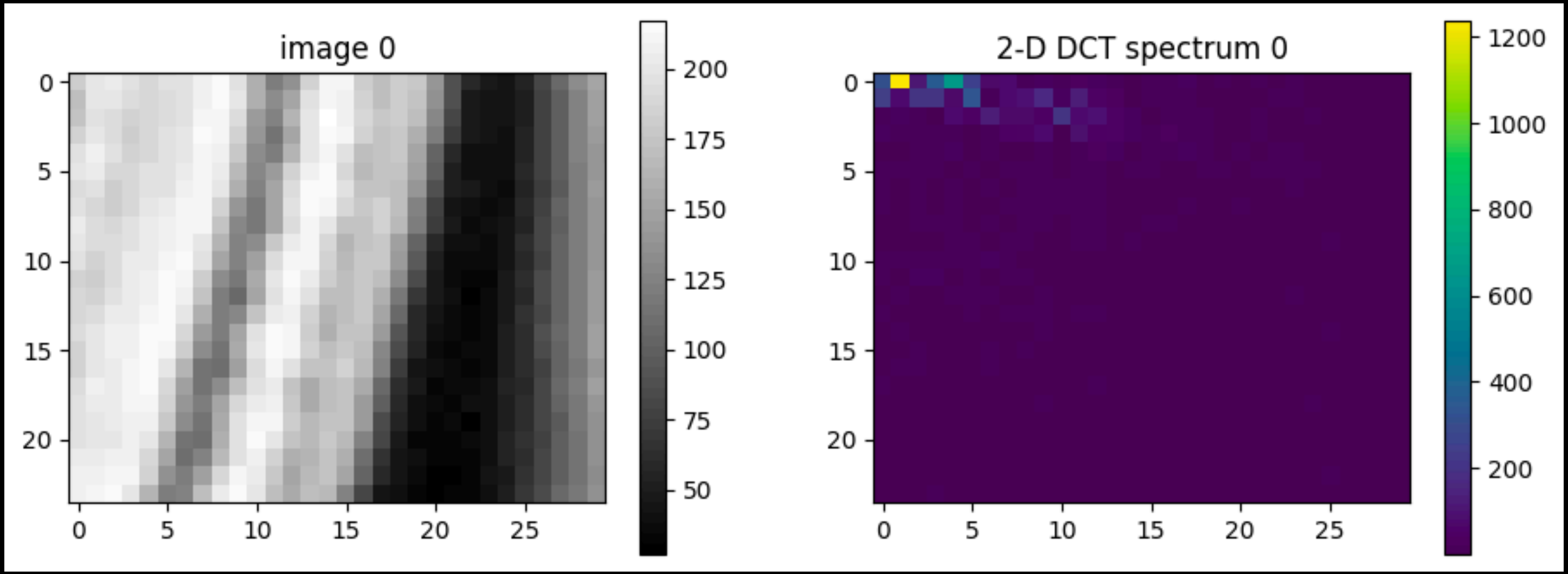


Transform Coding: DCT



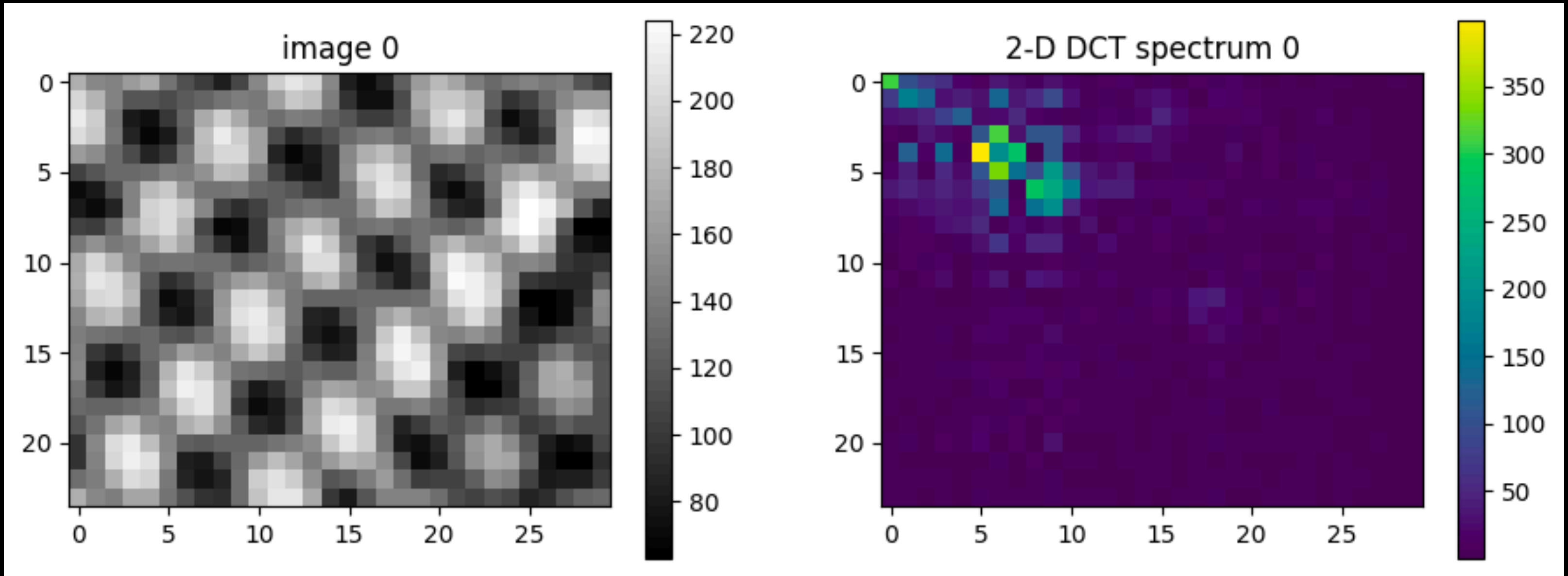
DCT is **Sparse!**

Transform Coding: DCT



DCT is **Sparse!**

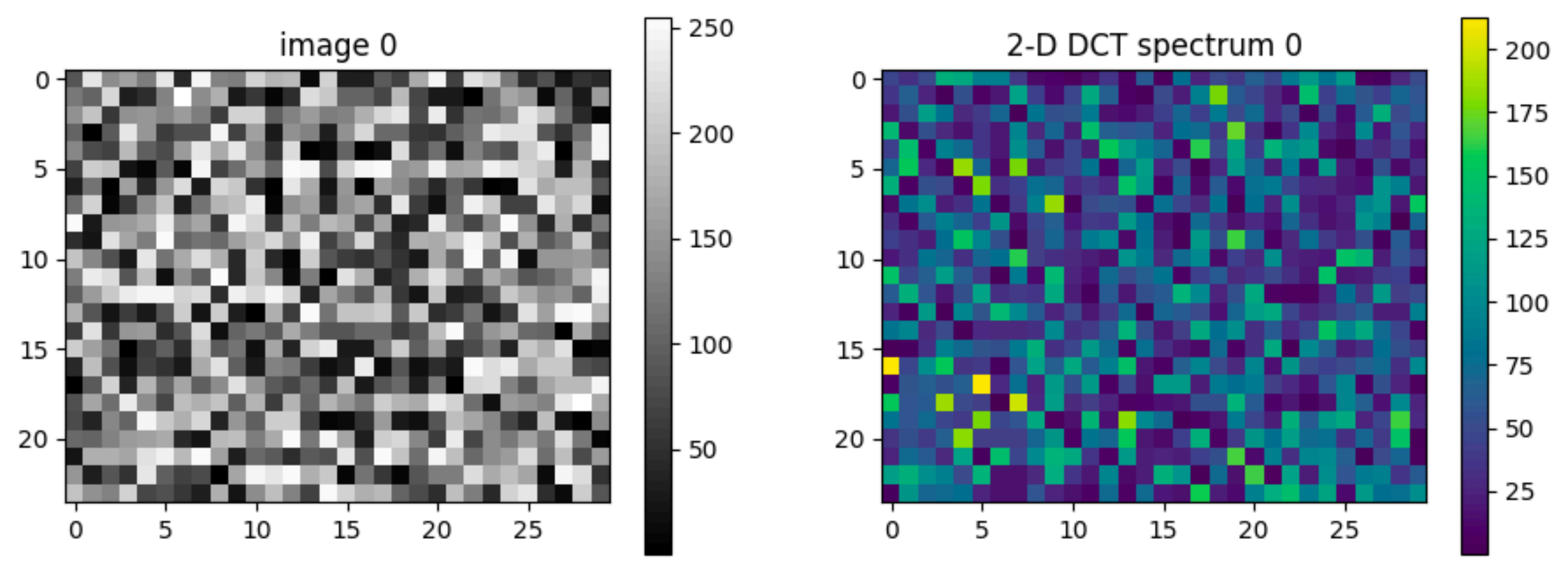
Transform Coding: DCT



DCT is **Sparse!**

but some higher frequency components

Transform Coding -> DCT of noise



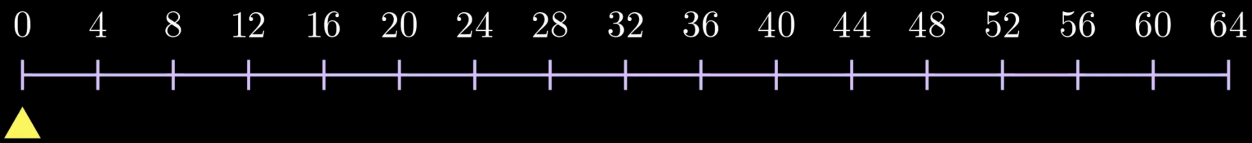
DCT is ~~**Sparse!**~~ not sparse!

~~but some higher frequency components~~

Transform Coding: DCT

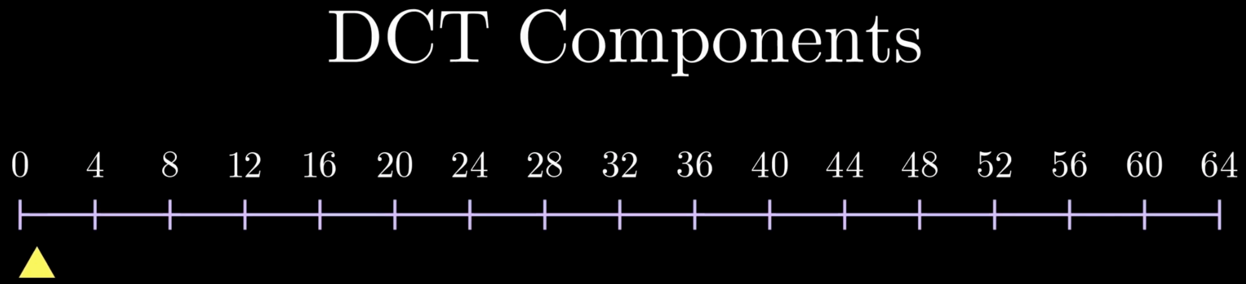
- Observation: For most of the “natural” image blocks, the DCT is sparse, and concentrated in the lower frequencies

DCT Components



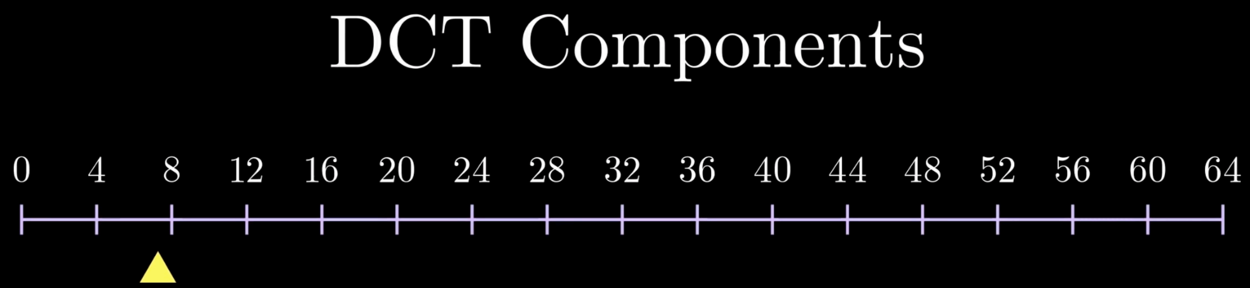
Transform Coding: DCT

- 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



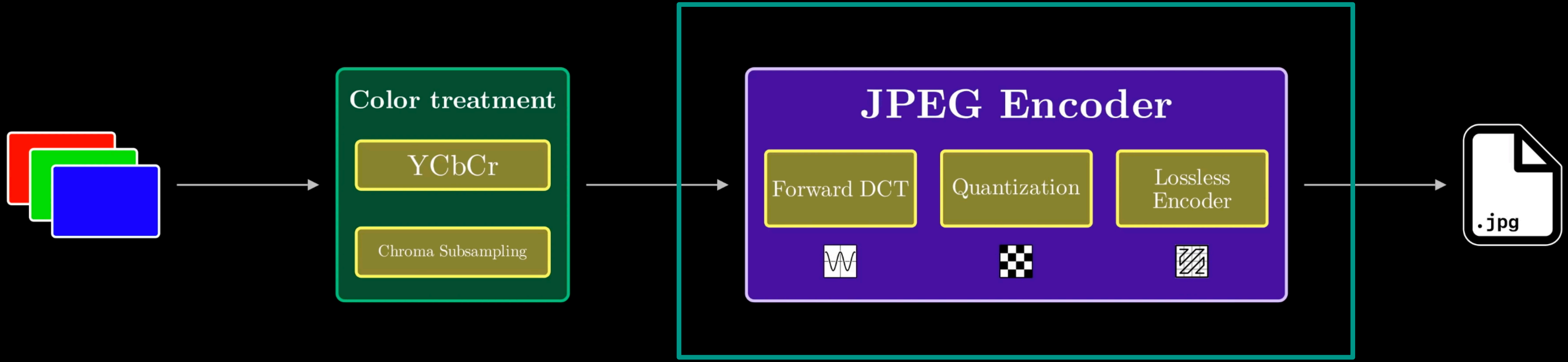
Transform Coding: DCT

- 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)

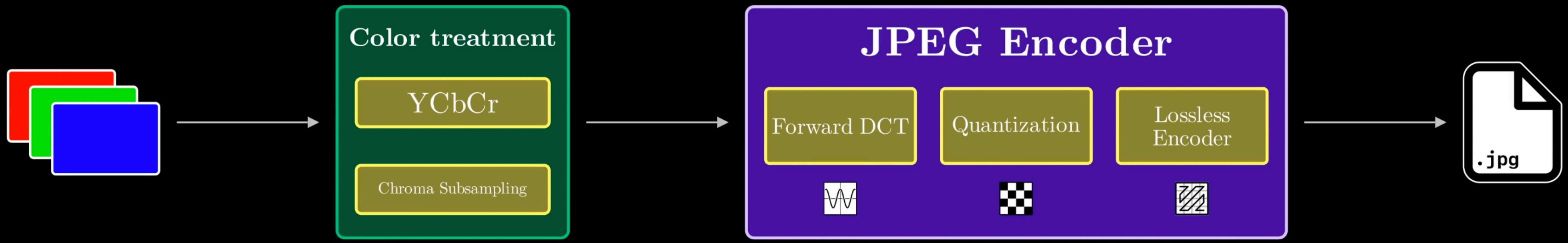
This key observation forms the basis of JPEG image compression

JPEG Image Compression



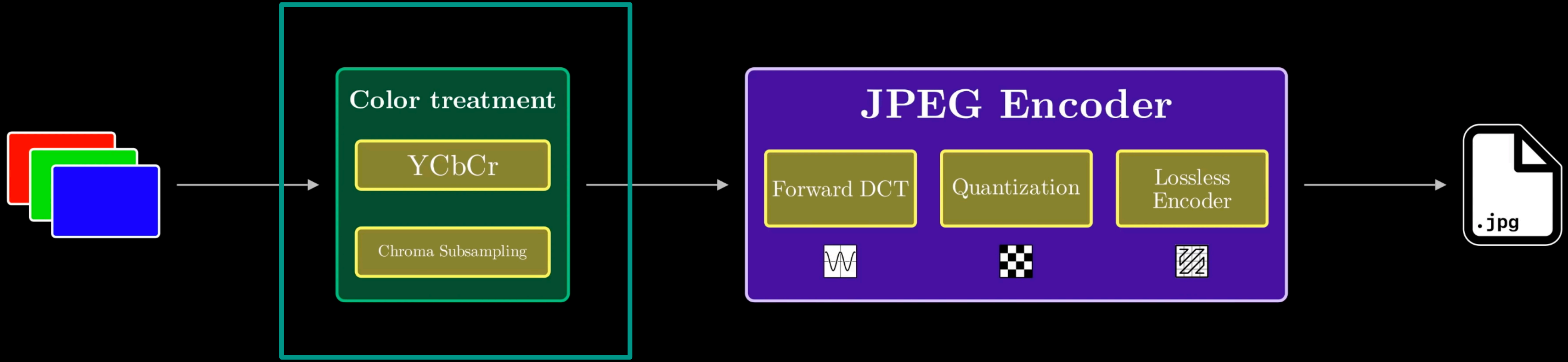
Familiar Block!

JPEG Image Compression



A **lot** of design decision for JPEG compressor are based on human vision properties!
(we will have a dedicated lecture on this)

JPEG Image Compression

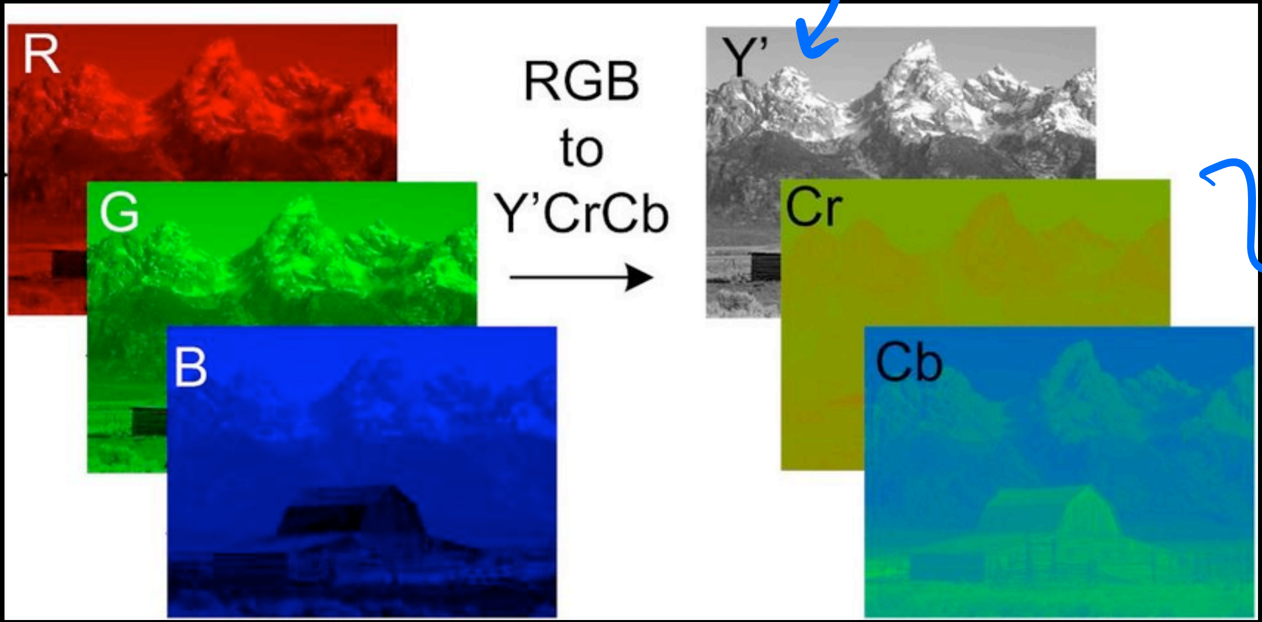


Color transform + Subsampling

JPEG Image Compression

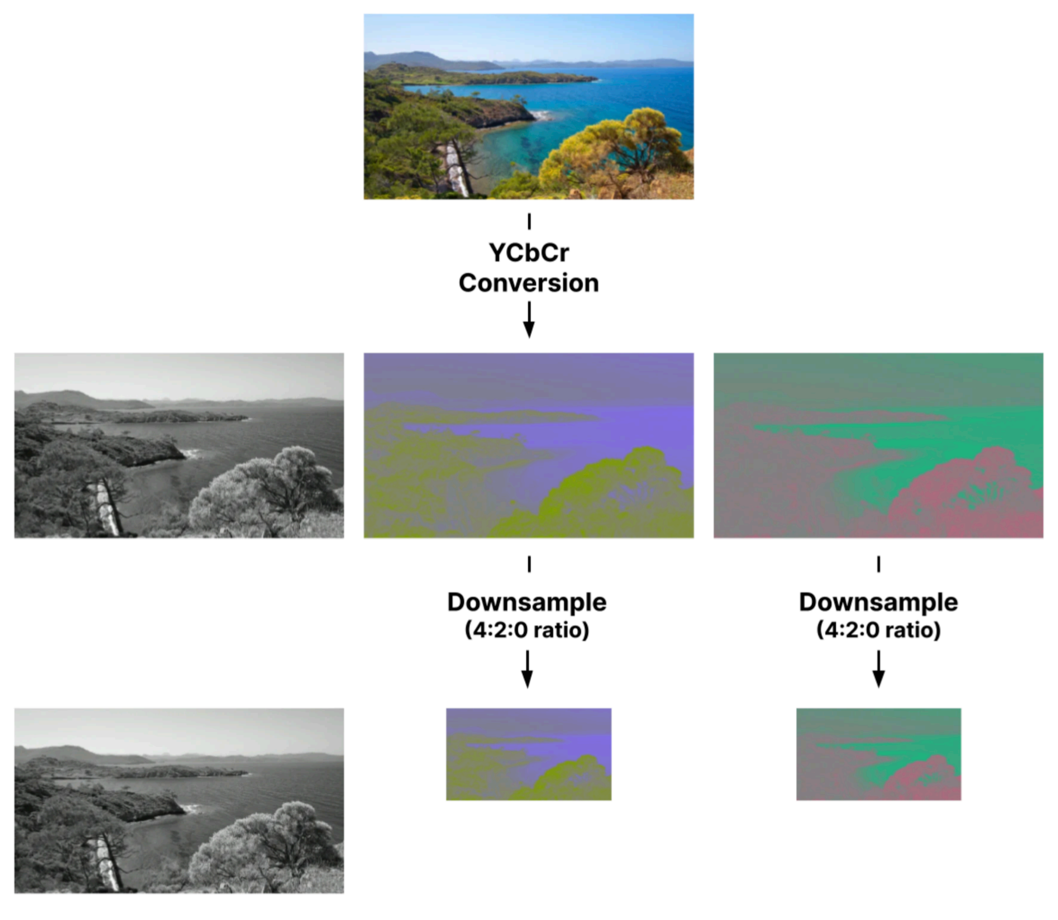
Luma

Color transform

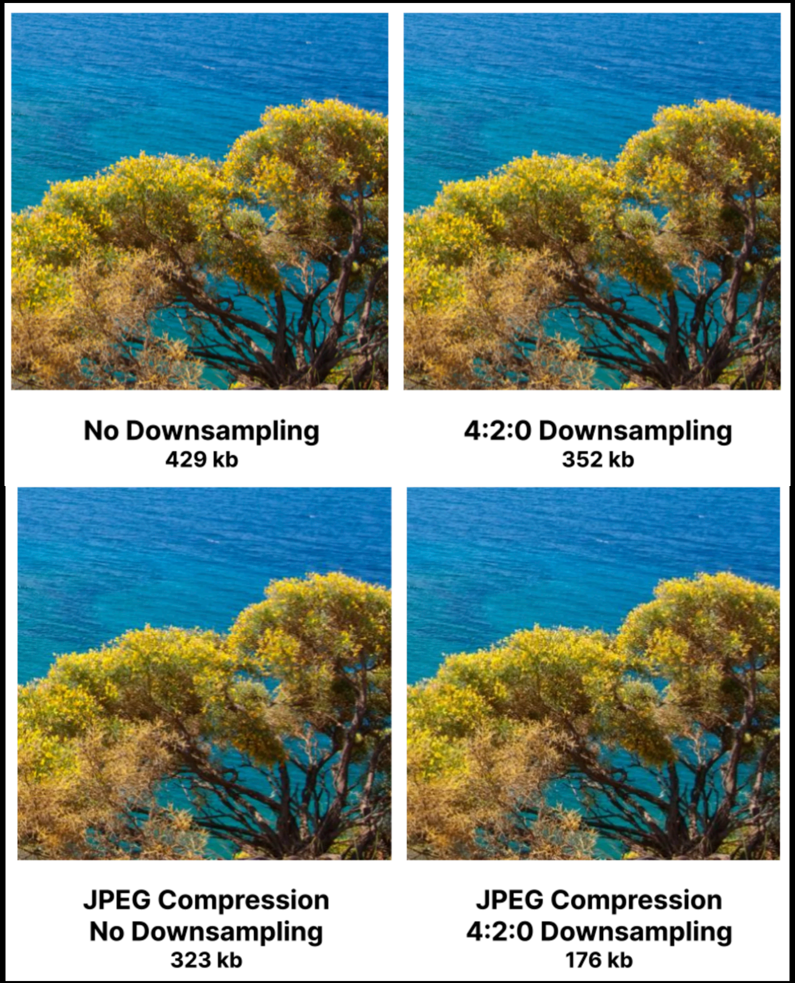


Chroma

JPEG Image Compression



Chroma Subsampling



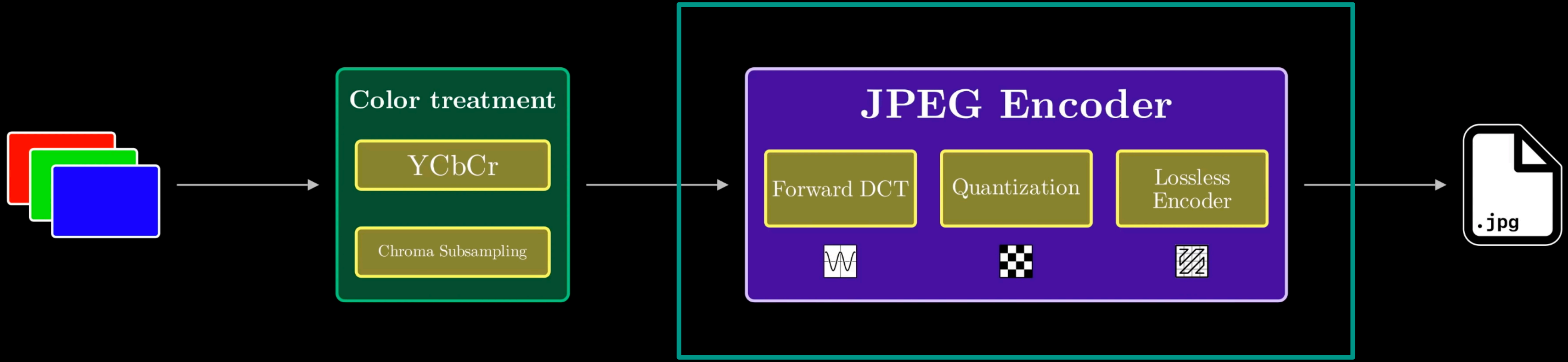
JPEG Image Compression

First step is Color Transformation from ***RGB*** to ***YCbCr (or Y'CbCr, YUV, Y'UV)***

But why? Human-aspect (more on this later)

- Reason 1:
  perceptual color space based on opponent process theory of color vision
- Reason 2:
  different ***contrast sensitivity*** of Y, Cb, Cr channels

JPEG Image Compression

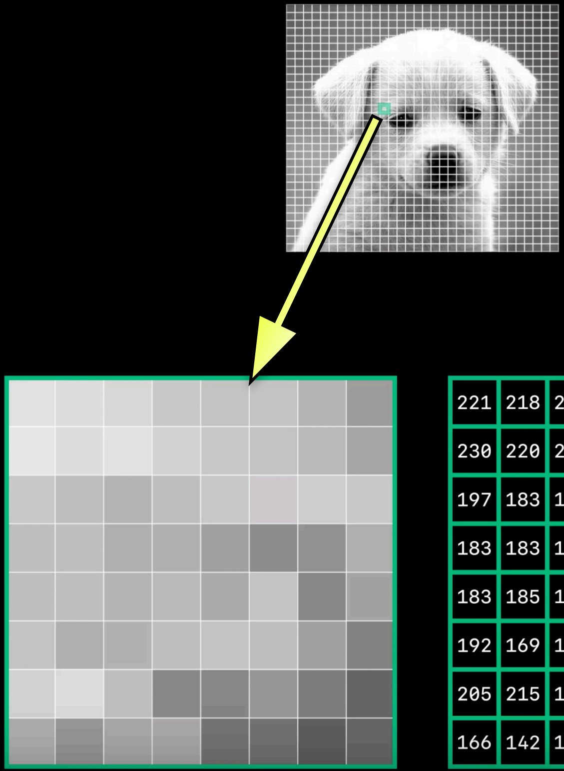


Familiar Block!

JPEG Image Compression: 2D-Block DCT

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

STEP-1.5:
subtract 128, to center the pixels



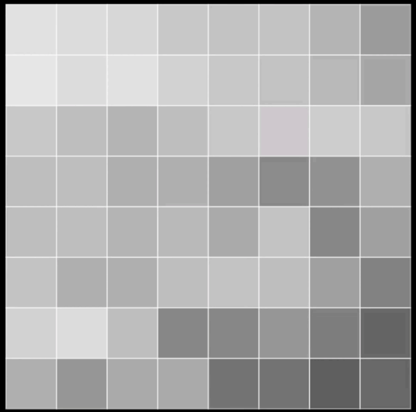
221	218	211	196	189	189	174	149
230	220	223	205	193	188	177	160
197	183	175	185	193	198	200	193
183	183	168	170	151	129	139	166
183	185	175	181	163	187	126	154
192	169	170	183	188	185	153	120
205	215	186	126	123	142	118	93
166	142	161	161	107	105	85	94

Input 8x8 block

JPEG Image Compression: 2D-Block DCT

STEP-2:

2D DCT of each 8x8 block



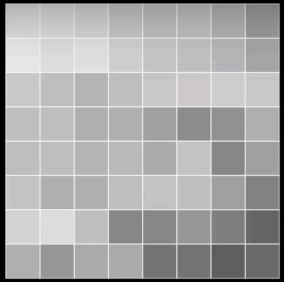
93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	-2	26
64	41	42	55	60	57	25	-8
77	87	58	-2	-5	14	-10	-35
38	14	33	33	-21	-23	-43	-34

Input 8x8 block
(zero centered)

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

2D DCT

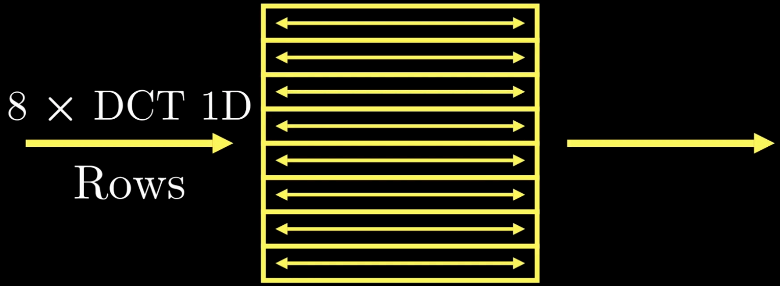
JPEG Image Compression: 2D-Block DCT



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

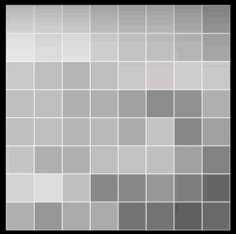
Input 8x8 block
(zero centered)



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)

JPEG Image Compression: 2D-Block DCT

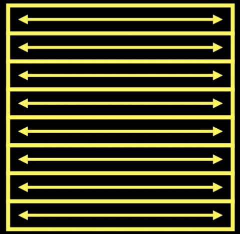


93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	-2	26
64	41	42	55	60	57	25	-8
77	87	58	-2	-5	14	-10	-35
38	14	33	33	-21	-23	-43	-34

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

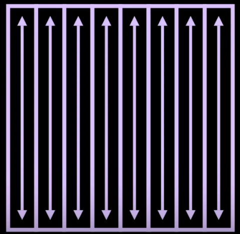
$8 \times$ DCT 1D
Rows



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
Columns



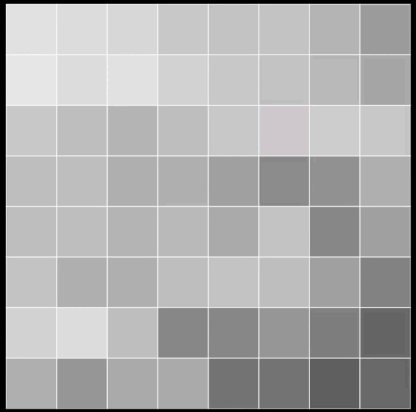
338	145	-8	18	-7	4	16	-14
162	-41	3	2	3	-1	-13	9
-17	57	-2	-2	-20	16	2	10
41	19	-24	31	-19	-8	4	-1
-59	7	-2	-32	21	-1	6	-15
-19	12	32	0	-16	-9	-15	12
7	-55	-24	17	20	15	-4	0
15	-11	10	11	-18	-13	10	-10

2D DCT

JPEG Image Compression: 2D-Block DCT

STEP-2:

2D DCT of each 8x8 block



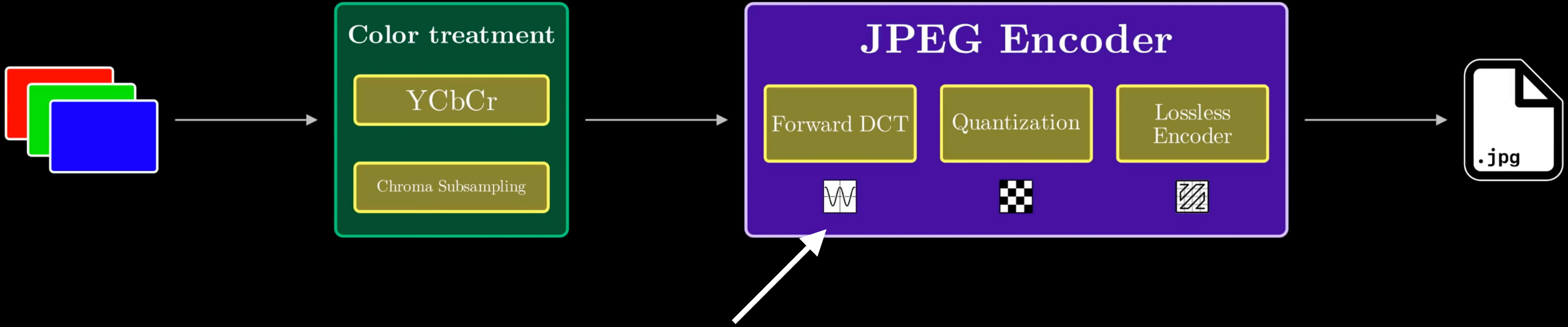
93	90	83	68	61	61	46	21
102	92	95	77	65	60	49	32
69	55	47	57	65	70	72	65
55	55	40	42	23	1	11	38
55	57	47	53	35	59	-2	26
64	41	42	55	60	57	25	-8
77	87	58	-2	-5	14	-10	-35
38	14	33	33	-21	-23	-43	-34

Input 8x8 block
(zero centered)

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

2D DCT

JPEG Image Compression



Efficient separable
DCT implementation

Complexity:
$\underbrace{2*8*8}_{2 \times N^2}$, instead of $\underbrace{64}_{}*64$

$N^2 \times N^2$

$N=8$

1D-DCT : N^2

2D-DCT : $N^2 \times N^2$

JPEG Image Compression -> Quantization

Rate Controller

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

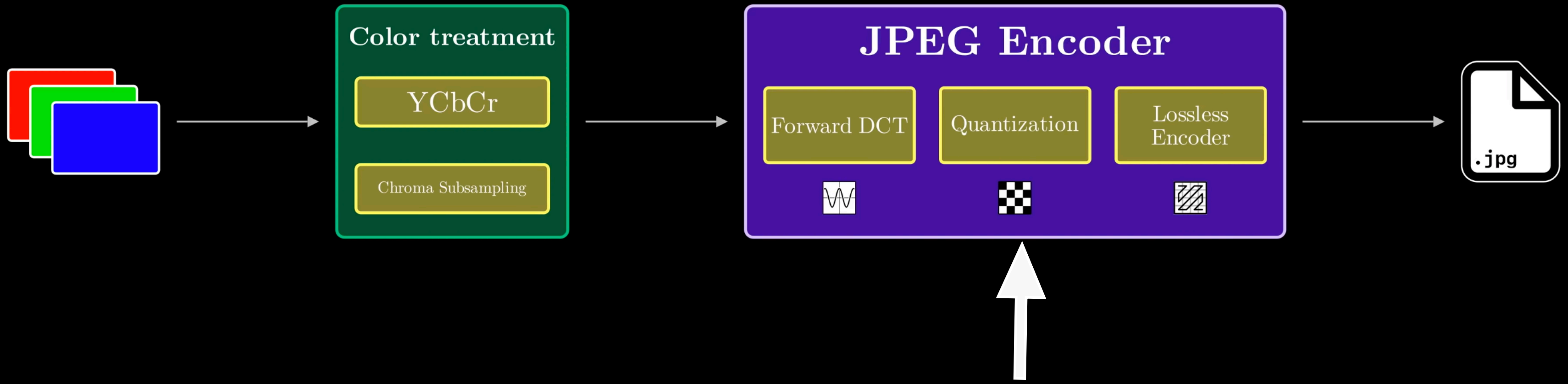
Quantization Table

$\longrightarrow$

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



Different quantization tables
For different compression rate

Designed based on water-filling + human vision properties

JPEG Image Compression: Entropy coding

90, -40, 0, 0, 0, 5 ...

=

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

RLE [90, -40, (3, 0), 5, ...]

(run length encoding)

#zero preceding

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

Huffman Encoder

$\log_2 \lceil \# \rceil$

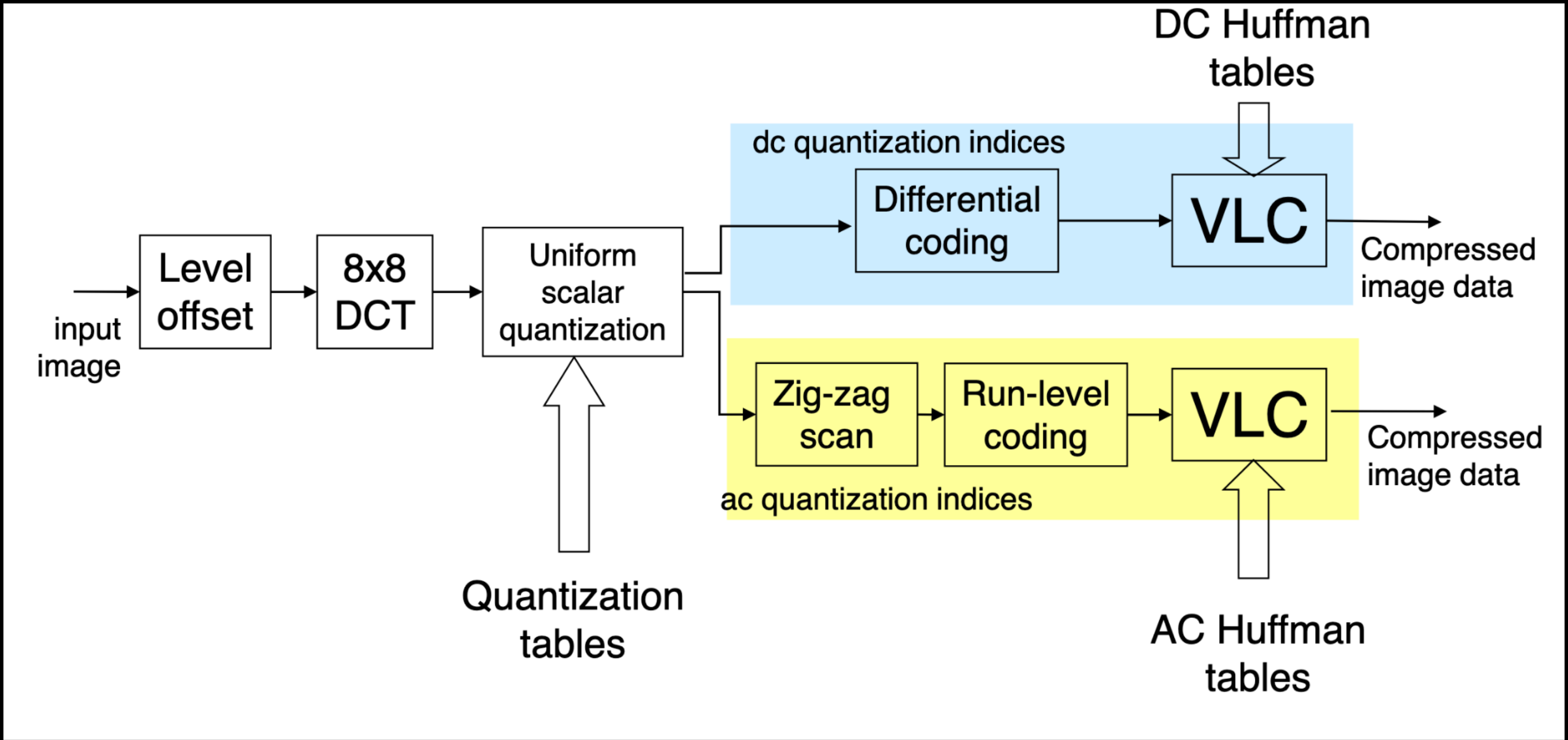
Variable-length code:
Huffman Encode the length;
store the value as-is in bits
Q. Why?

#bits in unary encoding of 90

JPEG Image Compression: Entropy coding

Quite complicated:

- Signs of coefficients
- DC vs AC coefficients
  - smoother variation in DC vs AC
  - DC more important to preserve
- Combining across blocks
- Luma vs Chroma channels
- Chroma subsampling


DC Huffman tables
dc quantization indices
Differential coding
VLC
Level offset
8x8 DCT
Uniform scalar quantization
input image
Compressed image data
Zig-zag scan
Run-level coding
VLC
Compressed image data
ac quantization indices
Quantization tables
AC Huffman tables

JPEG Compression Summary

- Color Conversion: RGB is converted to YUV color space
- Color Channels: 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. Lots of caveats on how Huffman is used!

Image Compression: Analysis

Image from Kodak dataset

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

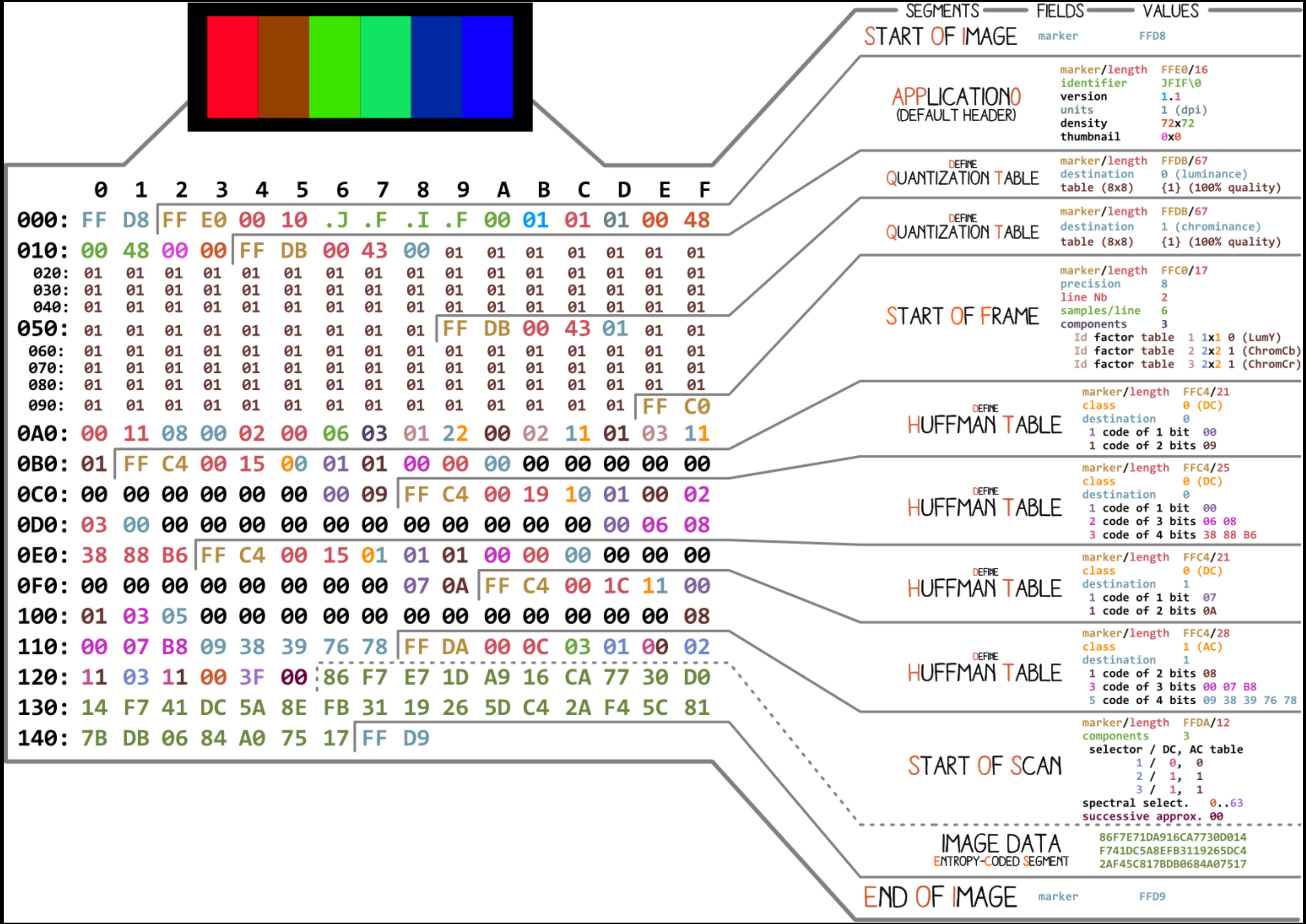
Image Compression: Analysis

Image from Kodak dataset

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

Notice ***BLOCKY*** artifacts!

JPEG Decoder specification



How would you improve upon JPEG?

BPG:

- Larger and variable-sized blocks are allowed

- Blocks are not independent
  Predict the next block based on the previous ones

Image Compression: JPEG 137X

Image from Kodak dataset

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

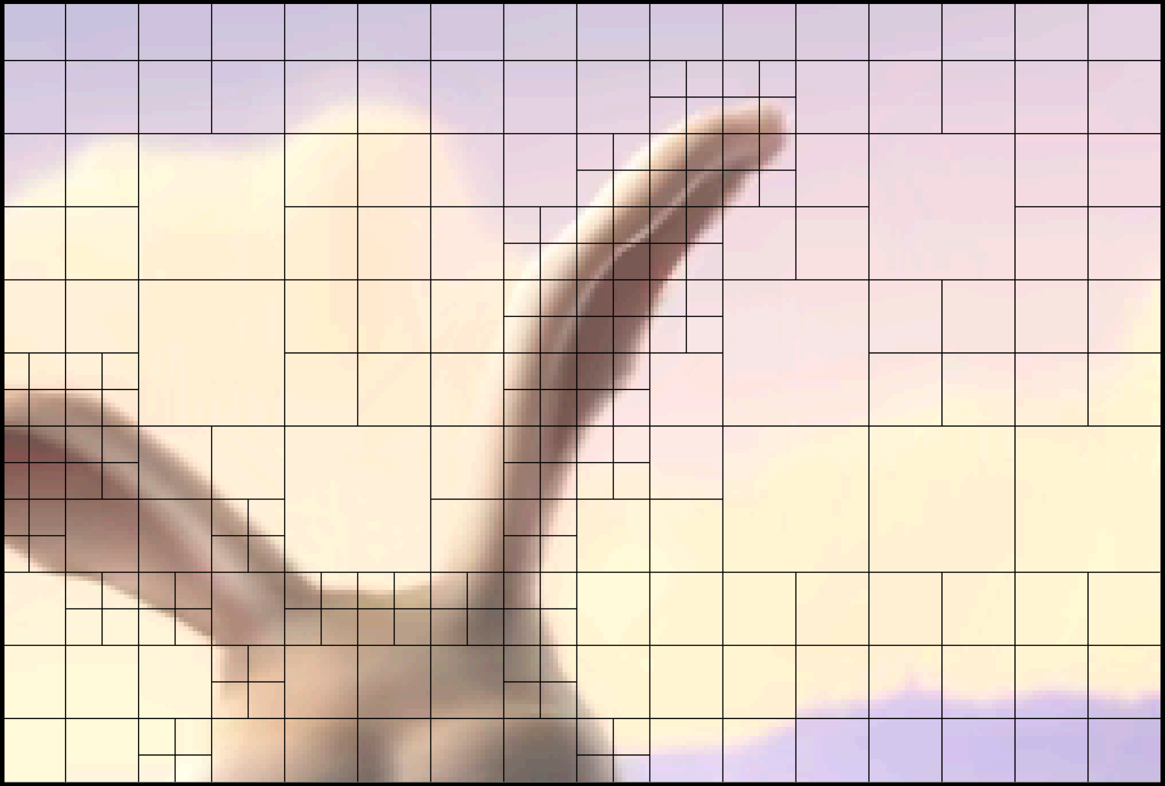
Image Compression: BPG 137X

Image from Kodak dataset

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

BPG/H.265-Iframe

Larger and variable-sized blocks are allowed (up to 32x32)



BPG/H.265-Iframe

Larger and variable-sized blocks are allowed (up to 32x32)

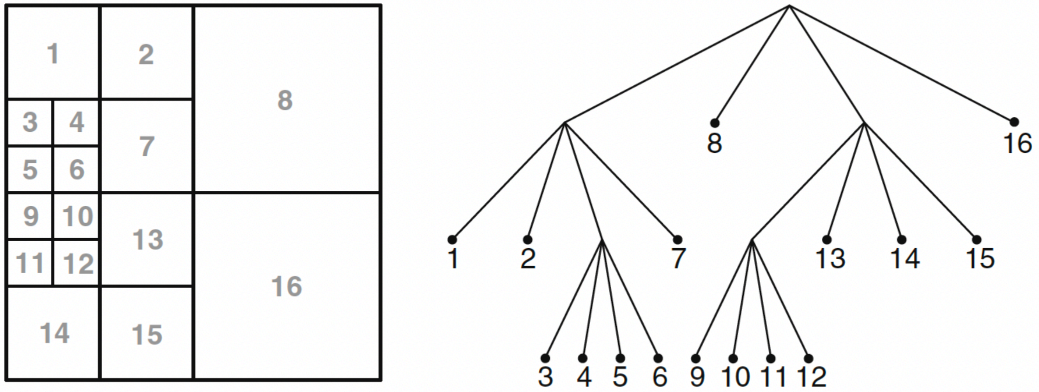
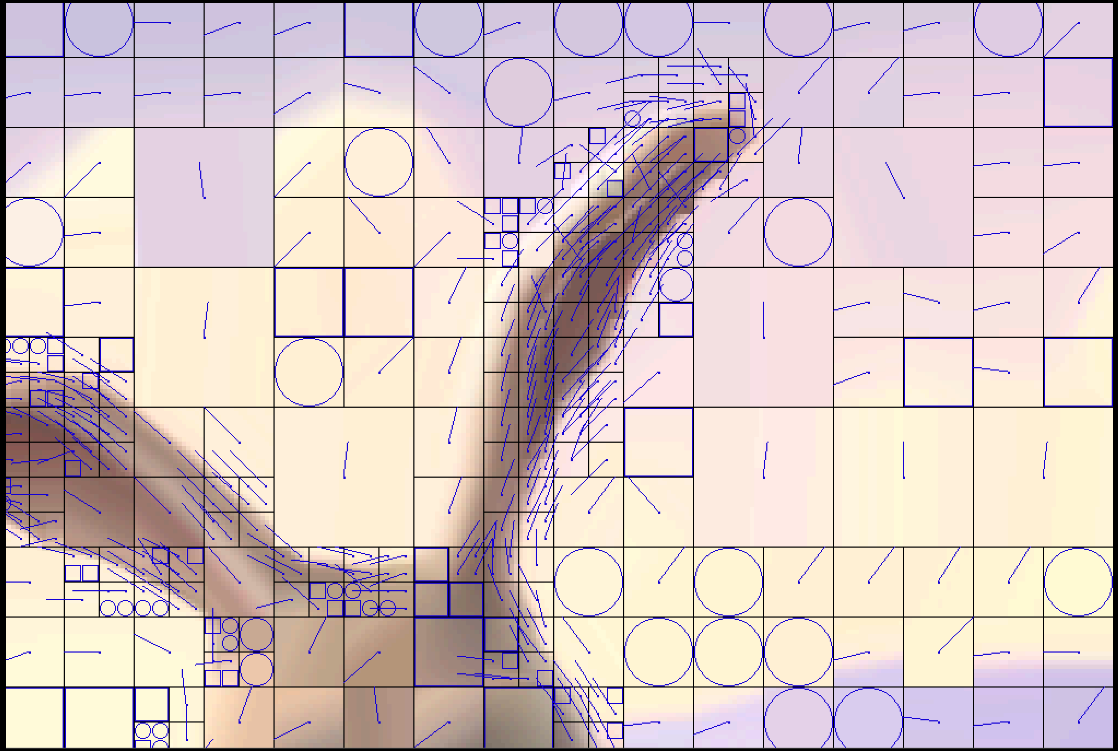


Fig. 3.4 Example for the partitioning of a 64×64 coding tree unit (CTU) into coding units (CUs) of 8×8 to 32×32 luma samples. The partitioning can be described by a quadtree, also referred to as coding tree, which is shown on the *right*. The numbers indicate the coding order of the CUs

BPG/H.265-Iframe

Blocks are not independent anymore!
Predictive coding



BPG Prediction modes

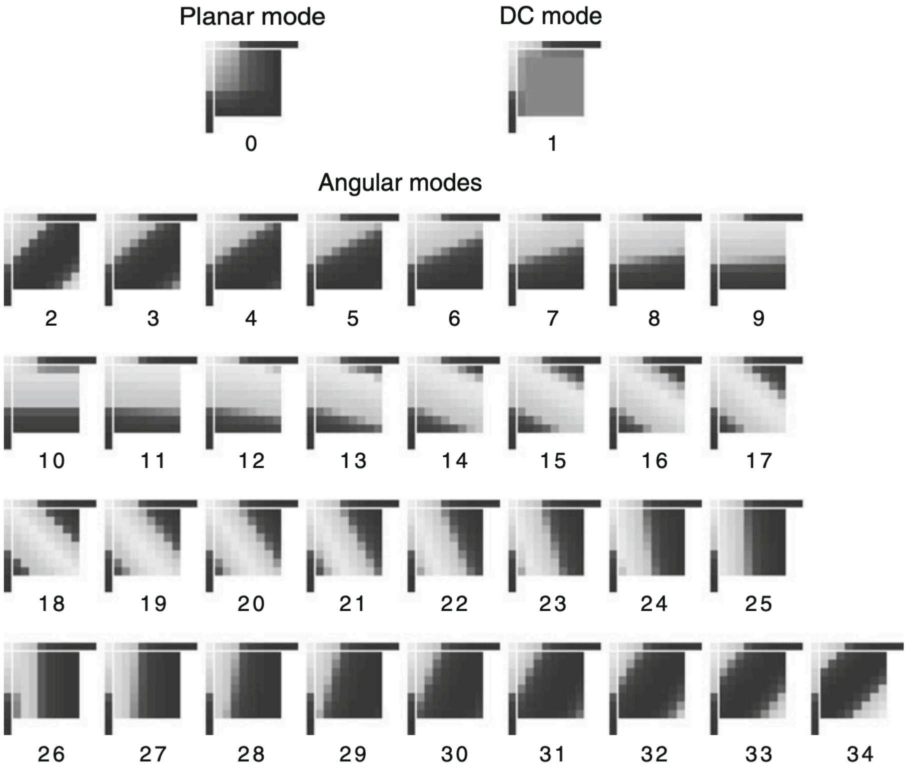
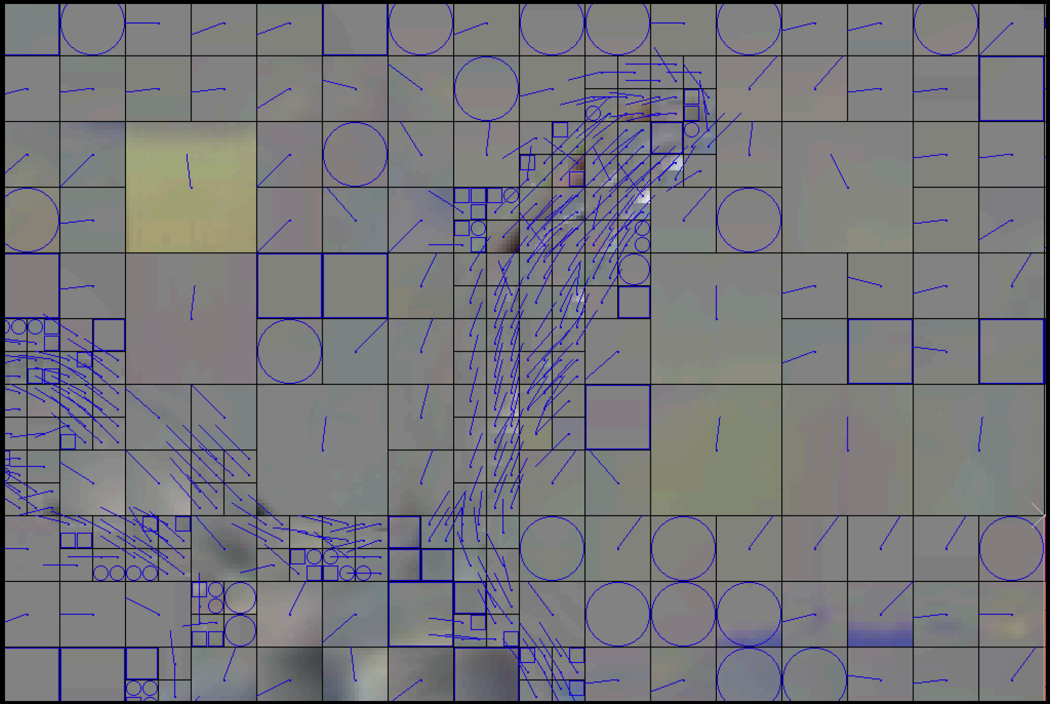


Fig. 4.2 Examples of 8×8 luma prediction blocks generated with all the HEVC intra prediction modes. Effects of the prediction post-processing can be seen on the top and left borders of the DC prediction (mode 1), top border of horizontal mode 10 and left border of vertical mode 26

BPG/H.265-Iframe

Larger and variable-sized blocks are allowed (up to 32x32)



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?