Image Compression
A recent sunset in Half Moon Bay

Picture taken on my iPhone 7 (12 MPixel sensor)

$4032 \times 3024 \text{ pixels} \times (3 \text{ bytes/pixel}) = 34.9 \text{ MB uncompressed image}$

JPG compressed image = 2.9 MB
Idea 1:

Q. What is the most efficient way to encode intensity values as a byte?

A. Encode based on how the brain perceives brightness not, based on actual response of eye
Lightness (perceived brightness) aka luma

Lightness ($L^*$) \( \rightarrow \) Luminance ($Y$) = \( \int \lambda \) Spectral sensitivity of eye
(Perceived by brain) (Response of eye) (eye’s response curve)

Dark adapted eye: \( L^* \propto Y^{0.4} \)
Bright adapted eye: \( L^* \propto Y^{0.5} \)

In a dark room, you turn on a light with luminance: \( Y_1 \)
You turn on a second light that is identical to the first. Total output is now: \( Y_2 = 2Y_1 \)

Total output appears \( 2^{0.4} = 1.319 \) times brighter to dark-adapted human

Note: Lightness ($L^*$) is often referred to as luma ($Y'$)
Consider an image with pixel values encoding luminance (linear in energy hitting sensor)

Luminance \( (Y) \)

Perceived brightness: \( L^* \)

Consider 12-bit sensor pixel:
Can represent 4096 unique luminance values in output image

Values are \( \sim \) linear in luminance since they represent the sensor’s response

\[ L^* = Y^{0.45} \]
Problem: quantization error

Many common image formats store 8 bits per channel (256 unique values)
Insufficient precision to represent brightness in darker regions of image

Luminance ($Y$)

Perceived brightness: $L^*$

$L^* = Y^{0.45}$

Bright regions of image: perceived difference between pixels that differ by one step in luminance is small!
(human may not even be able to perceive difference between pixels that differ by one step in luminance!)

Dark regions of image: perceived difference between pixels that differ by one step in luminance is large!
(quantization error: gradients in luminance will not appear smooth.)

Rule of thumb: human eye cannot differentiate <1% differences in luminance

Stanford CS348K, Fall 2018
Store lightness, not luminance

Idea: distribute representable pixel values evenly with respect to perceived brightness, not evenly in luminance (make more efficient use of available bits)

Solution: pixel stores $Y^{0.45}$
Must compute $(\text{pixel\_value})^{2.2}$ prior to display on LCD

Warning: must take caution with subsequent pixel processing operations once pixels are encoded in a space that is not linear in luminance.

e.g., When adding images should you add pixel values that are encoded as lightness or as luminance?
Idea 2:

- Chrominance ("chroma") subsampling

- The human visual system is less sensitive to detail in chromaticity than in luminance
  - So it is sufficient to sample chroma at a lower rate
Y’CbCr color space

Y’ = luma: perceived luminance (non-linear)
Cb = blue-yellow deviation from gray
Cr = red-cyan deviation from gray

Conversion from R’G’B’ to Y’CbCr:

\[
Y’ = 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256}
\]

\[
C_B = 128 + \frac{-37.945 \cdot R'_D}{256} - \frac{74.194 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256}
\]

\[
C_R = 128 + \frac{112.439 \cdot R'_D}{256} - \frac{94.154 \cdot G'_D}{256} - \frac{18.285 \cdot B'_D}{256}
\]
Example: compression in Y’CbCr

Original picture of Kayvon
Example: compression in Y’CbCr

Contents of CbCr color channels downsampled by a factor of 20 in each dimension (400x reduction in number of samples)
Example: compression in Y’CbCr

Full resolution sampling of luma (Y’)

Example: compression in Y’CbCr

Reconstructed result
(looks pretty good)
# Chroma subsampling

Y′CbCr is an efficient representation for storage (and transmission) because Y′ can be stored at higher resolution than CbCr without significant loss in perceived visual quality.

<table>
<thead>
<tr>
<th>Y′_{00}</th>
<th>Y′_{10}</th>
<th>Y′_{20}</th>
<th>Y′_{30}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cb_{00}</td>
<td></td>
<td>Cb_{20}</td>
<td></td>
</tr>
<tr>
<td>Cr_{00}</td>
<td></td>
<td>Cr_{20}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y′_{01}</th>
<th>Y′_{11}</th>
<th>Y′_{21}</th>
<th>Y′_{31}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cb_{01}</td>
<td></td>
<td>Cb_{21}</td>
<td></td>
</tr>
<tr>
<td>Cr_{01}</td>
<td></td>
<td>Cr_{21}</td>
<td></td>
</tr>
</tbody>
</table>

### 4:2:2 representation:

Store Y′ at full resolution
Store Cb, Cr at full vertical resolution, but only half horizontal resolution.

### 4:2:0 representation:

Store Y′ at full resolution
Store Cb, Cr at half resolution in both dimensions

### X:Y:Z notation:

- X = width of block
- Y = number of chroma samples in first row
- Z = number of chroma samples in second row

### Real-world 4:2:0 examples:

- most JPG images and H.264 video
- Blue-Ray
Idea 3:

- Low frequency content is predominant in the real world

- The human visual system is less sensitive to high frequency sources of error in images

- So a good compression scheme needs to accurately represent lower frequencies, but it can be acceptable to sacrifice accuracy in representing higher frequencies
Recall: frequency content of images
Recall: frequency content of images

Spatial domain result

Spectrum (after low-pass filter)
All frequencies above cutoff have 0 magnitude
Recall: frequency content of images

Spatial domain result (strongest edges)

Spectrum (after high-pass filter)
All frequencies below threshold have 0 magnitude
A recent sunset in Half Moon Bay
A recent sunset in Half Moon Bay (with noise added)
A recent sunset in Half Moon Bay (with more noise added)
A recent sunset in Half Moon Bay

Original image

Noise added
(increases high frequency content)

More noise added
What is a good representation for manipulating frequency content of images?

Hint:
Image transform coding via discrete cosine transform (DCT)

8x8 pixel block (64 coefficients of signal in "pixel basis")

64 basis coefficients

64 cosine basis vectors (each vector is 8x8 image)

\[
basis[i,j] = \cos \left( \frac{i \pi}{\frac{1}{2}} \right) \times \cos \left( \frac{j \pi}{\frac{1}{2}} \right)
\]

In practice: DCT applied to 8x8 pixel blocks of Y’ channel, 16x16 pixel blocks of Cb, Cr (assuming 4:2:0)
Examples of other bases
This slide illustrates basis images for 4x4 image block

Pixel Basis
(Compact: each coefficient in representation only effects a single pixel of output)

DCT

Walsh-Hadamard

Haar Wavelet

[Image credit: https://people.xiph.org/~xiphmont/demo/daala/demo3.shtml]
Quantization

Quantization produces small values for coefficients (only few bits needed per coefficient)
Quantization zeros out many coefficients

Result of DCT
(representation of image in cosine basis)

\[
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -8 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2
\end{bmatrix}
\]

Quantization Matrix

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

Changing JPEG quality setting in your favorite photo app modifies this matrix (“lower quality” = higher values for elements in quantization matrix)

Result of DCT

Quantization Matrix

Slide credit: Wikipedia, Pat Hanrahan
Stanford CS348K, Fall 2018
JPEG compression artifacts

Noticeable 8x8 pixel block boundaries

Noticeable error near high gradients

Low Quality

Medium Quality

Low-frequency regions of image represented accurately even under high compression
Why might JPEG compression not be a good compression scheme for illustrations and rasterized text?
Images with high frequency content do not exhibit as high compression ratios. Why?

Original image: 2.9MB JPG

Medium noise: 22.6 MB JPG

High noise: 28.9 MB JPG

Photoshop JPG compression level = 10 used for all compressed images

Uncompressed image: 4032 x 3024 x 24 bytes/pixel = 36.6 MB
Lossless compression of quantized DCT values

Quantized DCT Values

Entropy encoding: (lossless)

Reorder values

Run-length encode (RLE) 0's

Huffman encode non-zero values

JPEG compression summary

Coefficient reordering → Quantized DCT

quantization matrix

= RLE compression of zeros

Entropy compression of non-zeros

Compressed bits

Quantization loses information (lossy compression!)

Lossless compression!
JPEG compression summary

Convert image to Y’CbCr
Downsample CbCr (to 4:2:2 or 4:2:0)  (information loss occurs here)
For each color channel (Y’, Cb, Cr):
   For each 8x8 block of values
      Compute DCT
      Quantize results  (information loss occurs here)
      Reorder values
      Run-length encode 0-spans
      Huffman encode non-zero values
Key idea: exploit characteristics of human perception to build efficient image storage and image processing systems

- Separation of luminance from chrominance in color representation (Y’CrCb) allows reduced resolution in chrominance channels (4:2:0)

- Encode pixel values linearly in lightness (perceived brightness), not in luminance (distribute representable values uniformly in perceptual space)

- JPEG compression significantly reduces file size at cost of quantization error in high spatial frequencies
  - Human brain is more tolerant of errors in high frequency image components than in low frequency ones
  - Images of the real world are dominated by low-frequency components
H.264 Video Compression
30 second video: 1920 x 1080, @ 30fps

After decode: 8-bits per channel RGB $\rightarrow$ 24 bits/pixel $\rightarrow$ 6.2MB/frame

$(6.2 \text{ MB} \times 30 \text{ sec} \times 30 \text{ fps}) = 5.2 \text{ GB}$

Size of data when each frames stored as JPG: 531MB

Actual H.264 video file size: 65.4 MB (80-to-1 compression ratio, 8-to-1 compared to JPG)

Compression/encoding performed in real time on my iPhone

Go Swallows!
H.264/AVC video compression

- AVC = advanced video coding
- Also called MPEG4 Part 10
- Common format in many modern HD video applications:
  - Blue Ray
  - HD streaming video on internet (YouTube, Vimeo, iTunes store, etc.)
  - HD video recorded by your smart phone
  - European broadcast HDTV (U.S. broadcast HDTV uses MPEG 2)
  - Some satellite TV broadcasts (e.g., DirecTV)
- Benefit: much higher compression ratios than MPEG2 or MPEG4
  - Alternatively, higher quality video for fixed bit rate
- Costs: higher decoding complexity, substantially higher encoding cost
  - Idea: trades off more compute for requiring less bandwidth/storage
Hardware implementations

- Support for H.264 video encode/decode is provided by fixed-function hardware on most modern processors (not just mobile devices)

- Hardware encoding/decoding support existed in modern Intel CPUs since Sandy Bridge (Intel “Quick Sync”)

- Modern operating systems expose hardware encode decode support through hardware-accelerated APIs
  - e.g., DirectShow/DirectX (Windows), AVFoundation (iOS)
Video container format versus video codec

- Video container (MOV, AVI) bundles media assets

- Video codec: H.264/AVC (MPEG 4 Part 10)
  - H.264 standard defines how to represent and decode video
  - H.264 does not define how to encode video (this is left up to implementations)
  - H.264 has many profiles
    - High Profile (HiP): supported by HDV and Blue Ray
Video compression: main ideas

- Compression is about exploiting redundancy in a signal
  - Intra-frame redundancy: value of pixels in neighboring regions of a frame are good predictor of values for other pixels in the frame (spatial redundancy)
  - Inter-frame redundancy: pixels from nearby frames in time are a good predictor for the current frame’s pixels (temporal redundancy)
Residual: difference between compressed image and original image

Original pixels

Compressed pixels (JPEG quality level 2)

Compressed pixels (JPEG quality level 6)

Residual (amplified for visualization)
H.264/AVC video compression overview

Residual: difference between predicted pixel values and input video pixel values

Credit: Figure derived from H.264 Advanced Video Compression Standard, I. Richardson, 2010
16 x 16 macroblocks

Video frame is partitioned into 16 x 16 pixel macroblocks

Due to 4:2:0 chroma subsampling, macroblocks correspond to 16 x 16 luma samples and 8 x 8 chroma samples
Macroblocks in an image are organized into slices

Figure to left shows the macro blocks in a frame

Macroblocks are grouped into slices

Can think of a slice as a sequence of macroblocks in raster scan order *

Slices can be decoded independently **

One 16x16 macroblock

* H.264 also has non-raster-scan order modes (FMO), will not discuss today.

** Final “deblocking” pass is often applied to post-decode pixel data, so technically slices are not fully independent.
Decoding via prediction + correction

During decode, samples in a macroblock are generated by:

1. Making a prediction based on already decoded samples in macroblocks from the same frame (intra-frame prediction) or from other frames (inter-frame prediction)
2. Correcting the prediction with a residual stored in the video stream

Three forms of prediction:

- **I-macroblock**: macroblock samples predicted from samples in previous macroblocks in the same slice of the current frame
- **P-macroblock**: macroblock samples can be predicted from samples from one other frame (one prediction per macroblock)
- **B-macroblock**: macroblock samples can be predicted by a weighted combination of multiple predictions from samples from other frames
Intra-frame prediction (I-macroblock)

- Prediction of sample values is performed in spatial domain, not transform domain
  - Predicting pixel values, not basis coefficients

- Modes for predicting the 16x16 luma (Y) values: *
  - Intra_4x4 mode: predict 4x4 block of samples from adjacent row/col of pixels
  - Intra_16x16 mode: predict entire 16x16 block of pixels from adjacent row/col
  - I_PCM: actual sample values provided

* An additional 8x8 mode exists in the H.264 High Profile

Yellow pixels: already reconstructed (values known)
White pixels: 4x4 block to be reconstructed

Intra_4X4

Intra_16x16
Intra_4x4 prediction modes

- Nine prediction modes (6 shown below)
  - Other modes: horiz-down, vertical-left, horiz-up

Mode 0: vertical
(4x4 block is copy of above row of pixels)

Mode 1: horizontal
(4x4 block is copy of left col of pixels)

Mode 2: DC
(4x4 block is average of above row and left col of pixels)

Mode 3: diagonal down-left (45°)

Mode 4: diagonal down-right (45°)

Mode 5: vertical-right (26.6°)
Intra_16x16 prediction modes

- 4 prediction modes: vertical, horizontal, DC, plane

Mode 0: vertical

Mode 1: horizontal

Mode 2: DC

Mode 4: plane

\[ P[i,j] = A_i \times B_j + C \]

A derived from top row, B derived from left col, C from both
Further details

- Intra-prediction of chroma (8x8 block) is performed using four modes similar to those of intra_16x16 (except reordered as: DC, vertical, horizontal, plane)

- Intra-prediction scheme for each 4x4 block within macroblock encoded as follows:
  - One bit per 4x4 block:
    - if 1, use most probable mode
      - Most probable = lower of modes used for 4x4 block to left or above current block
    - if 0, use additional 3-bit value rem_intra4x4_pred_mode to encode one of nine modes
      - if rem_intra4x4_pred_mode is smaller than most probable mode, use mode given by rem_intra4x4_pred_mode
      - else, mode is rem_intra4x4_pred_mode+1

<table>
<thead>
<tr>
<th>rem_intra4x4_pred_mode</th>
<th>mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>??</td>
<td>??</td>
</tr>
</tbody>
</table>
Inter-frame prediction (P-macroblock)

- Predict sample values using values from a block of a previously decoded frame *
- Basic idea: current frame formed by translation of pixels from temporally nearby frames (e.g., object moved slightly on screen between frames)
  - “Motion compensation”: use of spatial displacement to make prediction about pixel values

* Note: “previously decoded” does not imply source frame must come before current frame in the video sequence. (H.264 supports decoding out of order.)
P-macroblock prediction

- Prediction can be performed at macroblock or sub-macroblock granularity
  - Macroblock can be divided into 16x16, 8x16, 16x8, 8x8 “partitions”
  - 8x8 partitions can be further subdivided into 4x8, 8x4, 4x4 sub-macroblock partitions

- Each partition predicted by sample values defined by: 
  (reference frame id, motion vector)

- Block A: predicted from (frame 0, motion-vector = [-3, -1])
- Block B: predicted from (frame 1, motion-vector = [-2.5, -0.5])

Note: non-integer motion vector
Motion vector visualization

Image credit: Keyi Zhang
Non-integer motion vectors require resampling

Interpolation to 1/2 pixel sample points via 6-tap filter:
\[
\text{half} \_\text{integer} \_\text{value} = \text{clamp}((A - 5B + 20C + 20D - 5E + F) / 32)
\]

H.264 supports both 1/2 pixel and 1/4 pixel resolution motion vectors
- 1/4 resolution resampling performed by bilinear interpolation of 1/2 pixel samples
- 1/8 resolution (chroma only) by bilinear interpolation of 1/4 pixel samples

Example: motion vector with 1/2 pixel values.
Must resample reference block at positions given by red dots.
Motion vector prediction

- Problem: per-partition motion vectors require significant amount of storage
- Solution: predict motion vectors from neighboring partitions and encode residual in compressed video stream
  - Example below: predict D’s motion vector as average of motion vectors of A, B, C
  - Prediction logic becomes more complex when partitions of neighboring blocks are of different size
Question: what partition size is best?

- Smaller partitions likely yield more accurate prediction
  - Fewer bits needed for residuals

- Smaller partitions require more bits to store partition information (diminish benefits of prediction)
  - Reference picture id
  - Motion vectors (note: motion vectors are more coherent with finer sampling, so they likely compress well)
Inter-frame prediction (B-macroblock)

- Each partition predicted by up to two source blocks
  - Prediction is the average of the two reference blocks
  - Each B-macroblock partition stores two frame references and two motion vectors (recall P-macroblock partitions only stored one)
Additional prediction details

- Optional weighting to prediction:
  - Per-slice explicit weighting (reference samples multiplied by weight)
  - Per-B-slice implicit weights (reference samples weights by temporal distance of reference frame from current frame in video)
    - Idea: weight samples from reference frames nearby in time more

- Deblocking
  - Blocking artifacts may result as a result of macroblock granularity encoding
  - After macroblock decoding is complete, optionally perform smoothing filter across block edges.
Putting it all together: encoding an inter-predicted macroblock

- Inputs:
  - Current state of decoded picture buffer (state of the decoder)
  - 16x16 block of input video to encode

- General steps: (need not be performed in this order)
  - Resample images in decoded picture buffer to obtain 1/2, and 1/4, 1/8 pixel resampling
  - Choose prediction type (P-type or B-type)
  - Choose reference pictures for prediction
  - Choose motion vectors for each partition (or sub-partition) of macroblock
  - Predict motion vectors and compute motion vector difference
  - Encode choice of prediction type, reference pictures, and motion vector differences
  - Encode residual for macroblock prediction
  - Store reconstructed macroblock (post deblocking) in decoded picture buffer to use as reference picture for future macroblocks
H.264/AVC video encoding

MB = macroblock
MV = motion vector

Source Video Frame → Intra-frame Prediction → Actual MB pixels → Predicted MB → Inter-frame Prediction → Motion vectors → Motion Vector Pred. → Compute MV Diffs → Compute Residual → Transform/Quantize Residual → Entropy Encoder → Compressed Video Stream

Decoded picture buffer → Deblock → Inverse transform/quantize → Basis coefficients

Credit: Figure derived from H.264 Advanced Video Compression Standard, I. Richardson, 2010
Motion estimation

- Encoder must **find** reference block that predicts current frame’s pixels well.
  - Can search over multiple pictures in decoded picture buffer + motion vectors can be non-integer (huge search space)
  - Must also choose block size (macroblock partition size)
  - And whether to predict using combination of two blocks
  - Literature is full of heuristics to accelerate this process
    - Remember, must execute motion estimation in real-time for HD video (1920x1080), on a low-power smartphone

Limit search window:

- gray area: search region
  - Decoded picture buffer: frame 0
  - Current frame
Motion estimation optimizations

- **Coarser search:**
  - Limit search window to small region
  - First compute block differences at coarse scale (save partial sums from previous searches)

- **Smarter search:**
  - Guess motion vectors similar to motion vectors used for neighboring blocks
  - Diamond search: start by test large diamond pattern centered around block
    - If best match is interior, refine to finer scale
    - Else, recenter around best match

- **Early termination:** don’t find optimal reference patch, just find one that’s “good enough”: e.g., compressed representation is lower than threshold
  - Test zero-motion vector first (optimize for non-moving background)

- **Optimizations for subpixel motion vectors:**
  - Refinement: find best reference block given only pixel offsets, then try 1/2, 1/4-subpixel offsets around this match
Fraction of energy consumed by different parts of instruction pipeline (H.264 video encoding)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>IME</th>
<th>FME</th>
<th>IP</th>
<th>CABAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>90%</td>
<td>85%</td>
<td>75%</td>
<td>50%</td>
</tr>
<tr>
<td>SIMD+VLIW</td>
<td>95%</td>
<td>90%</td>
<td>80%</td>
<td>60%</td>
</tr>
<tr>
<td>CP Fus</td>
<td>92%</td>
<td>87%</td>
<td>77%</td>
<td>57%</td>
</tr>
<tr>
<td>Magic</td>
<td>93%</td>
<td>90%</td>
<td>85%</td>
<td>65%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>FU (%)</th>
<th>RF (%)</th>
<th>Ctrl (%)</th>
<th>Pip (%)</th>
<th>D-$ (%)</th>
<th>D-$ (%)</th>
<th>Pip (%)</th>
<th>IF (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IME</td>
<td>56%</td>
<td>44%</td>
<td>11%</td>
<td>9%</td>
<td>8%</td>
<td>6%</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>FME</td>
<td>36%</td>
<td>23%</td>
<td>12%</td>
<td>11%</td>
<td>10%</td>
<td>7%</td>
<td>3%</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>7%</td>
<td>6%</td>
<td>5%</td>
<td>4%</td>
<td>3%</td>
<td>2%</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>CABAC</td>
<td>1%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- FU = functional units
- RF = register fetch
- Ctrl = misc pipeline control
- Pip = pipeline registers (interstage)
- D-$ = data cache
- IF = instruction fetch + instruction cache

[Image of bar chart with labels for IME, FME, IP, and CABAC]

[Stanford CS348K, Fall 2018]
H.265 (HVEC)

- Standard ratified in 2013
- Goal: ~2X better compression than H.264
- Main ideas:
  - Macroblock sizes up to 64x64
  - Prediction block size and residual block sizes can be different
  - 35 intra-frame prediction modes (recall H.264 had 9)
  - ...
Learned compression schemes

- JPG image compression and H.264 video compression are “lossy” compression techniques that discard information that is present in the visual signal, but less likely to be noticed by the human eye.
  - Key principle: “Lossy, but still looks good enough to humans!”

- Compression schemes described in this lecture involved manual choice / engineering of good representations (features).
  - Frequency domain representation, YUV representation, disregarding color information, flow vectors, etc.

- Increasing interest in learning good representations for a specific class of images/videos, or for a specific task to perform on images/videos.
Learned compression schemes

- Many recent DNN-based approaches to compressing video learn to compress the residual

[Tsai et al. 2018]
Use standard video compression at low quality, then use an auto encoder to compress the residual.
Summary

- JPG image compression and H.264 video compression are “lossy” compression techniques that discard information is that less likely to be noticed by the human eye
  - Key principle: “Lossy, but still looks good enough to humans!”

- But most videos in the world will soon be analyzed by computers, not viewed by humans
  - New principle: “Lossy, but image analysis tasks still work!”
  - Can we “learn” domain-specific compressors for particular scenes, types of tasks, etc?