

HOMEWORK 6

Due Friday December 5th, by start of class

1. Huffman coding:

If you are given a set of input symbols and their associated probabilities, and you apply the Huffman code design algorithm to them, it is possible to obtain more than one set of codewords for which (a) the expected length of the code is the same, (b) all codes are the result of applying the Huffman code design algorithm, and (c) the actual codeword lengths are not the same from one set to another.

Here is an example of symbols and probabilities, and two different Huffman codeword sets that come from them:

Symbol	Probability	Codeword Set 1	Codeword Set 2
a_1	0.2	01	10
a_2	0.4	1	00
a_3	0.2	000	11
a_4	0.1	0010	010
a_5	0.1	0011	011

- (a) For both codes, go through the Huffman design procedure and draw the tree to show that both codeword sets result from a correct application of the Huffman code design algorithm.
- (b) Show that both sets of codewords have the same average length.
- (c) The variance in the length of the codewords is an important consideration. In many applications, the available transmission bandwidth is fixed. Although over a long time, both codes would produce the same average rate, over some short period of time, the code with more variance might produce a rate that is substantially less than or more than the average.

Another consideration is the ability to recover from errors in the channel. Encode the sequence

$$a_2 a_1 a_3 a_2 a_1 a_2$$

using the first code. Suppose the first bit is received in error. Decode the received sequence of bits. How many characters are received in error before the decoder gets back on track and correctly decodes a character?

Now do the same thing for the 2nd code.

- (d) Repeat the previous part with the error now in the third bit, instead of the first bit. Is there one code that, at least for the cases tested, does worse on error propagation?

2. JPEG quality levels:

This is a simple exercise intended just to give you a feel for the quality versus bit rate performance of JPEG. Read the images `aerial4.tif` and `xray.tif` into Matlab using `imread`.

Matlab allows you to control the quality level (and therefore the bit rate) of the JPEG image that you are writing out. Try writing out the image with terrible quality (quality level = 1) and with excellent quality (quality = 100) and with several values in between. If your image is named `aerial`, the `imwrite` command to get quality level 1 would be:

```
imwrite(aerial, 'aer1.jpg', 'jpg', 'quality', 1);
```

For each image, make a plot of MSE versus bit rate, and a plot of PSNR versus bit rate. That is, we want a plot that shows the MSE between the JPEG compressed image and the original image, versus the bit rate of the JPEG compression. To get the MSE, read the JPEG image back into Matlab, and compute the MSE between it and the original image. To get the bit rate, look at the file size of the jpeg file that Matlab created.

You do not have to generate very many points for your plot. Five points would be enough. Look at the images with `imshow`. For the lower bit rate versions, what kinds of artifacts do you see? What can you say comparatively about the compressibility of the `xray` and `aerial` images?

Note that you need to convert from `uint8` format to `double` to compute things like MSE and PSNR and so forth, but you don't want to convert to `double` for `imwrite`.

3. JPEG: Successive Approximation Progressive Mode

- (a) The following 8x8 block of DCT step indices is to be encoded by successive approximation, where we divide initially by 16. Show the (runlength, value) symbols that the encoder will need to encode for the AC coefficients. Show the decoder reconstruction of the step indices at this point (where the stage corresponding to division by 16 has been transmitted, but the stage corresponding to division by 8 has not). (Note: Bottom half of the 8x8 block is all zeros).

29	21	-14	3	1	0	-1	0
38	12	-7	2	0	0	0	0
-4	5	-3	-2	0	0	0	0
11	-1	0	0	1	0	0	0

- (b) Those (runlength, value) symbols could be encoded with the usual Huffman tables that were optimized for baseline sequential JPEG usage at 1 bit per pixel. (You are not asked to encode them.) How could we modify the Huffman tables so that they would perform better for symbols of this type (that is symbols from successive approximation such as those in part (a))? Suggest 2 modifications.
- (c) In the next scan (corresponding to division by 8), is it possible that sending an additional refinement bit to provide more information about a coefficient with non-zero history could degrade the representation of that coefficient at the decoder (i.e., the decoder will have a higher error for that coefficient after decoding that bit than it did before)? Why or why not?

4. Lempel-Ziv coding

A sequence is encoded using the Lempel-Ziv algorithm (LZ78) and the initial dictionary shown here.

Index	Entry
1	a
2	(space)
3	h
4	i
5	s
6	t

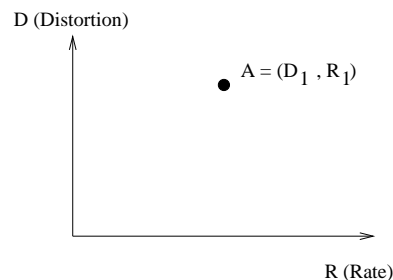
The output of the encoder is the following sequence

6	3	4	5	2	3	1	6	2	9	11	16	12	14	4	20	10	8	23	13
---	---	---	---	---	---	---	---	---	---	----	----	----	----	---	----	----	---	----	----

Pretend you are the decoder and you receive this sequence of outputs from the encoder. Decode this sequence. Show your dictionary.

5. Scalar and Vector Quantization

A random variable X is distributed in some unknown (perhaps bizarre) fashion over the interval $[0,1]$. A *uniform* 2-bit scalar quantizer is used to quantize X . This achieves a distortion D_1 and rate $R_1 = 2$ bits, and we can plot this point in the R,D plane.

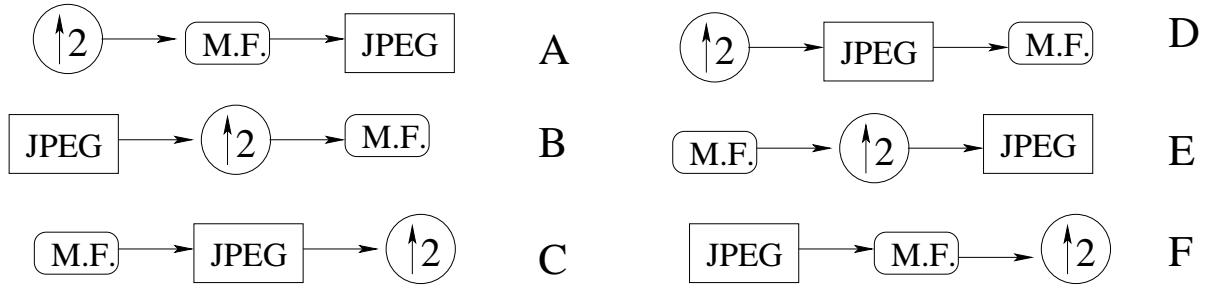


Consider the following 5 alternative ways of quantizing and representing X . For each, describe the *entire set of possible* (Rate,Distortion) points that could result. You can describe it in words or by drawing in the R,D plane.

- A globally optimal 2-bit Lloyd-Max scalar quantizer
- A Huffman coded uniform 2-bit scalar quantizer
- A globally optimal 1-bit Lloyd-Max scalar quantizer
- A 4-bit vector quantizer, used for two independent realizations (components) drawn from this same distribution. Consider the (Rate, Distortion) possible points *per component*. The VQ is designed by running the Generalized Lloyd Algorithm initialized by random selection from the distribution.
- Same as part (d), where the initialization is done with a product code that is a product of the SQs from part (a).

6. Order of Operations:

You have a set of images which have a low level (say, 1% or less) of a spiky (salt and pepper) noise. These images need to be upsampled by a factor of 2 using bilinear interpolation; they need to be compressed by JPEG (using strictly the default Huffman and Quantization tables), and they need to be median filtered to remove the noise. There are 6 possible orders in which the operations can be done. Note that the JPEG box corresponds to both an encoder and a decoder, so that the output of the box can be taken to be an image, not a compressed bit stream.



- Rank the 6 system from least to most in terms of the total number of bits in the compressed JPEG representation
- Which systems do you think would give the best and the worst image quality (where high quality would mean both maximal noise removal and minimal compression artifacts)?

7. JPEG on different images:

Baseline sequential JPEG is to be used to compress some images. The default Huffman and quantization tables will be used in all cases. The images are all grayscale, and all the same size. Rank the following images according to how large the JPEG compressed file size will be. If some of the images compress to exactly the same size as others, indicate that.

- A natural image such as the “airplane” image used in class.
- A natural image such as the “airplane” image, where all the pixel intensities have been divided by 2.
- An image where each *horizontal* strip of width 8 has a constant value, but adjacent strips have different and unrelated values.
- A constant valued image
- An image where each *horizontal* strip of width 1 has a constant value, but adjacent strips have different and unrelated values.
- An image where each *vertical* strip of width 8 has a constant value, but adjacent strips have different and unrelated values.