HOMEWORK 5

Due Friday March 15 by 4pm

Note: Problems 1-5 are pencil-and-paper problems. Only 6 and 7 require Matlab.

1. **Huffman coding 1:**

   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) Although both sets of codewords have the same average length, 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. **Huffman Coding 2:**

   For i.i.d. data, the smallest achievable expected length of a binary code is the first-order entropy, defined as

$$H = -\sum_{i=1}^{N} P(a_i) log_2 P(a_i)$$

   where $\{a_1, \ldots, a_N\}$ are the source symbols with associated probabilities $\{P(a_1), \ldots, P(a_N)\}$. Find a probability distribution on 6 source symbols such that the expected length of Huffman coding would achieve the entropy. List the probabilities and show the calculations of expected length and entropy.

3. **Huffman coding 3:**

   Which of these codes (there may be none, or one, or more than one) *cannot be Huffman codes* for any probability assignment?

   - {0, 10, 01, 11}
   - {00, 01, 10, 110}
   - {01, 10}
   - {0, 10, 11}

4. **Huffman coding 4:**

   Consider an alphabet with 7 symbols. We will design a full Huffman code for this alphabet. We will also design a truncated Huffman code for this alphabet, in which the 4 least probable symbols are grouped together and coded with a single base code for the group, followed by a 2-bit codeword which specifies the element of the group.

   (a) Choose a set of probabilities for the 7 symbols such that full Huffman code and the truncated Huffman code do not have the same codeword lengths. Show the codewords and their lengths. Find the expected lengths of both codes.

   (b) Choose a set of probabilities for the 7 symbols such that full Huffman code and the truncated Huffman code have the same codeword lengths. Show the codewords and their lengths. Find the expected lengths of both codes.

5. **JPEG:**

   An image such as the aerial image of the next problem is JPEG compressed, producing bit rate R_1 and MSE D_1. The point (R_1,D_1) can be plotted in a plot of distortion versus rate, such as shown below, where the point is labeled J.

   For each of the following, mark where you'd expect the (R,D) point to lie in the plane, relative to J. Label the points A through E. In some cases below, we pre-process

the image *before* compression, and in some cases we post-process the image *after* decompression (before display). In *all* cases, however, the MSE is computed between the final displayed image and the original aerial image.

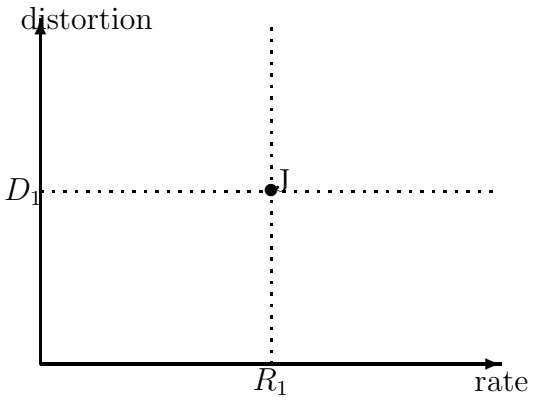
(A) The image first undergoes filtering with the 3x3 unweighted spatial averaging filter, and then it is JPEG compressed.

(B) The image is JPEG compressed. After decompression and prior to display, it undergoes filtering with the 3x3 unweighted spatial averaging filter.

(C) Rather than using JPEG's truncated Huffman coder, we substitute a full Huffman coder that does not do any grouping of symbols.

(D) The image is JPEG compressed using spectral selection progressive mode (progressing through all 4 scans) rather than baseline sequential JPEG.

(E) The encoder uses "Approximate Adaptive Quantization" as described in the JPEG handout on the web site, identifying unimportant image blocks & sending the End-of-Block symbol for them without sending AC coefs.



6. **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 anmed 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?

Depending in which version of Matlab you use, you may 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.

7. **Entropy before and after first-order prediction:**

(a) Below is a function which computes the first-order entropy of the individual pixel values in an image. In the first line after the function declaration, explain what probs and x are. In the next line, explain what we are trying to accomplish with the find function. In the next line, explain why we divide by sum(probs).

```
function entr = entropy(im)
[probs,x] = hist(im(:),(0:255));
probs = probs(find(probs));
probs = probs/sum(probs);
entr = -probs * log2(probs)';
```

Use this to compute the first-order entropies for the images baboon512.tif and peppers512.tif. Since the entropy is a measure of the minimum number of bits required, on the average, to losslessly encode the source, what does this result say about how compressible the two images are? Is this what you would have expected from looking at the two images?

(b) Suppose one wanted to compress these images using each pixel as a predictor for the pixel immediately to its right. Form the "residual" image in which each pixel is used as a prediction for the pixel to its right, and the prediction is subtracted from the actual value at that location. What is the entropy of this residual image? For the leftmost pixel in each row, you will need to find some alternative way to form a prediction for it, as there is no pixel to its left, or you can simply discard it and use the rest. Is this what you would have expected from looking at the two images?

For each of the images, would it make more sense to use prediction in the vertical direction or the horizontal direction? Comment on the result.