2018 ES 156 Project

Due date: May 10th, 11:59pm

Instruction

This project has 3 intertwined parts, and is worth up to 15% additional bonus points to your grade. The parts become broader/newer as you progress: Part I follows directly from class, Part II is new material but connected to class, Part III involves completely new material. You should turn in:

1. A well-written, typed write-up with the solution to each part, as well as any references used in your research. **Make sure you make use/refer to concepts seen in class**. The solutions should be written in IEEE format using the `IEEETran` conference (two-column) format. The latex template can be found here: https://ctan.org/pkg/ieeetran?lang=en, and equivalent Word templates can be found on Google. The write-up should include:
   - An abstract, concisely summarizing what you did and your contributions.
   - An introduction section composed of (i) an overview of the project, (ii) a related work section with the references you used, and an (iii) a notation section.
   - Each of the three parts of the project should be a section in your write-up, composing the main body of the document. Any graphs/equations/mathematical derivations that are relevant to your answers should also be included here (make sure you add captions to your figures).
   - A conclusion section.
   - Bibliographical references.
2. Working and well-documented code. You are expected to write your own code and, if possible, minimize the use of existing packages. You can use the programming language of your choice, but Matlab and/or Python are preferred.

This project is open-ended, and no formal solution will be posted online. There is no single solution to each problem, and different approaches may produce different outcomes. Grading will take into account not only your answers, but the overall quality of your write-up, code, insights, references, etc. The grading template that can serve as guidance for what is expected from you. Remember – these are free points, that are worth 50% of all the PSets you did, so the bar for grading will be set very high.

- **OK project (0-3 points):** You attempted 1, maybe 2 of the items below. Your answer is in IEEE format, but not particularly well written. You don't use many references, and some of the sections are missing. The time you put into the project is about the time you spent on 1 or 2 problem sets.
- **Good project (3-6 points):** You attempted all 3 of the components, making significant progress in at least one of them. Your write-up is insightful, with neat figures and significant references, and includes all the items mentioned above. Your code may be a bit sloppy, but can be used to reproduce your results. You spend about 2-3 problem sets worth of time.
- **Very good project (6-10 points):** You successfully solved all of the items below. In two of the components, your write-up includes resources and derivations from a few research papers/advanced text books. The introduction of the write-up can be used as a overview/study-guide for neural networks, DCT, or one of the other topics studied in the project. Your code is efficient and well-documented. You spent 3-4 problem sets worth of time.

- **Great project (10-13 points):** Same as the previous item, but for all 3 components. You spent about 4-5 problem sets worth of time.
- **Amazing project (13-15 points):** You produced near research-grade new results in your project, and your write-up is very close to a submission to an IEEE conference like ICASSP, ISIT, Globecom, ICC, etc, or to textbooks in the field. You not only solved the problems below, but extended them significantly by exploring new, state-of-the-art. For an amazing project, your write-up should be at the level of a textbook or conference submission in its precision, notation, conciseness, clarity, and literature overview.

Manage your time! The goal of the project is to allow you to explore more advanced topics, but that should not come at the cost of preparing for your finals, **so be reasonable and work wisely**. Remember: a good grade on the final is more beneficial than the project. As always, the teaching staff is here to help.

1 Part I – How good is 2-D Fourier for image processing?

1. **Image denoiser.** Produce an image processing/filtering routine to reconstruct an original black and white image from an observation that is noisy, down-sampled, and punctured. You can use all of the techniques you have learned in this course.

   Deliverable: You are required to submit a complete, well-documented function **together with test code** that (i) makes an image "noisy" by down-sampling, puncturing and adding noise and, (ii) attempts to reconstruct the original image. In particular, you will need to submit the following: (Note that an image refers to an array of unsigned integers (uint8))

   (a) A function called `noise_image` that downsamples a black and white image to 256×256 pixels, adds noise (of your choice) and punctures it. The input can be 512×512.

   (b) A function called `filter_image` that takes a 256×256 arbitrarily punctured image as input (output of `noise_image`) and produces the reconstructed full image of size 512×512 as output.

2. **Image compressor.** Write code that takes a 512×512 image as input and compresses it by computing the Fourier series and keeping the larger coefficients.

   Deliverable:

   (a) **Two-dimensional Fourier series.** You are required to submit a function that takes an image as input and computes the Fourier series of the periodically extended image (see below for details). In particular, two-dimensional Fourier series of a periodic function $x[m, n]$ with period M and N, respectively, in the first and the second dimension is given by:

   $$a_{k,l} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] e^{-j\left(\frac{2\pi}{M} km + \frac{2\pi}{N} ln\right)},$$

   for $k = 0, \ldots, M-1$ and $l = 0, \ldots, N-1$.

   (b) **Two-dimensional inverse Fourier series.** A second function will need to produce an image from a subset of the computed of Fourier coefficients above.

The inverse Fourier series is then given by

$$x[m,n] = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} a_{k,l} e^{j\left(\frac{2\pi}{M}km + \frac{2\pi}{N}ln\right)},$$

for $m = 0, \ldots, M-1$ and $n = 0, \ldots, N-1$.

(c) In this problem, we explore the trade-off between reconstruction quality and the compression rate. You should pick a few black and white images as reference. For $0 < \alpha < 100$ and each reference image, keep only the largest $\alpha\%$ of the coefficients with largest absolute values and replace the rest by zeros. Reconstruct the image $\widetilde{x}_\alpha[m,n]$ in each case and compute the reconstruction error

$$E_\alpha = \frac{1}{M}\frac{1}{N}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(\widetilde{x}_\alpha[m,n] - x[m,n])^2.$$

Plot E_α as a function of α. What do you observe? How does the compression/distortion trade-off look like? What would be the best trade-off between compression and distortion? Is the Fourier transform well suited for this? What other approaches would you use?

Note that you must use the following steps to avoid dealing with complex numbers.
Step 1. Make a real and even periodic signal from the provided image of size $M \times N$ whose period in the first and second dimension is $2M$ and $2N$, respectively.

Step 2. Apply the two-dimensional Fourier series to compute the Fourier coefficients for the resulting signal in Step 1. It follows that the Fourier series is real and even in both directions. Use this fact to only retain one quarter of the Fourier coefficients for perfect reconstruction.

2 Part II – Image processing, DCT and JPEG

For image processing applications[1], it is useful to consider the Discrete Cosine Transform (2D DCT) instead of the 2D DFT due to its superior empirical performance for signal compression and reconstruction tasks. We first introduce the two-dimensional discrete cosine $C_{kl}(m,n)$ of frequencies k, l defined as

$$C_{kl,MN}(m,n) \;=\; \cos\left[\frac{k\pi}{2M}(2m+1)\right]\cos\left[\frac{l\pi}{2N}(2n+1)\right]. \tag{1}$$

Then the two-dimensional DCT of a signal x is given by substituting $\mathbf{C}_{kl,MN}$ into an expression similar to the two-dimensional DFT, which, after introducing normalization constants, yields

$$\begin{aligned} x_C(k,l) &:= \frac{2}{\sqrt{MN}}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x(m,n)c_1c_2\cos\left[\frac{k\pi}{2M}(2m+1)\right]\cos\left[\frac{l\pi}{2N}(2n+1)\right] \\ &= \langle x, C_{kl,MN}\rangle. \end{aligned} \tag{2}$$

[1] Problem courtesy of Prof. Alejandro Ribeiro, UPenn.

where $c_1 = \frac{1}{\sqrt{2}}$ for $k = 0$ and $c_1 = 1$ for $k = 1, \ldots, M-1$ and $c_2 = \frac{1}{\sqrt{2}}$ for $l = 0$ and $c_2 = 1$ for $l = 1, \ldots, N-1$. **Argue that this may be computed as an "inner product in two dimensions"**, just like the 2D DFT.

Crucial to the theory of image reconstruction and compression is the 2D inverse Discrete Cosine Transform (2D iDCT), which is the signal $\tilde{x}_C$ defined as

$$\tilde{x}_C(m,n) := \frac{2}{\sqrt{MN}} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X_C(k,l) c_1 c_2 \cos\left[\frac{m\pi(2k+1)}{2M}\right] \cos\left[\frac{n\pi(2l+1)}{2N}\right] \tag{3}$$

where $c_1 = \frac{1}{\sqrt{2}}$ for $m = 0$ and $c_1 = 1$ for $m = 1, \ldots, M-1$ and $c_2 = \frac{1}{\sqrt{2}}$ for $n = 0$ and $c_2 = 1$ for $n = 1, \ldots, N-1$. Analogous to the 2D DFT, we note that the sum in (3) allows us to represent an arbitrary two-dimensional signal as a sum of cosines, and hence we may ask how many cosines are necessary to represent the signal well in terms of reconstruction error. We explore this question in the first part of this problem. Henceforth you may assume that $M = N$, so that signals are of dimension N^2.

Consider adding to your write-up a brief explanation about the DCT, and why it is an orthogonal decompostion of a two-dimensional signal (like others seen in class).

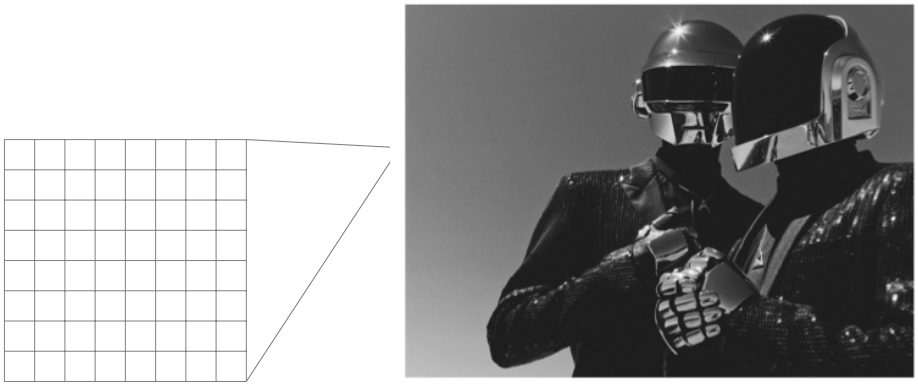
2.1 DCT in Two Dimensions

Write down a function which takes as input a two-dimensional signal of duration N^2 and computes its two-dimensional DCT defined in (2).

2.2 Image Compression

When the signal dimension N^2 is very large, it is difficult to represent it well across its entire domain using the same DFT or DCT coefficients. This is because the computation of the inverse in (3) uses the same DFT or DCT coefficients across the entire domain. We address this by partitioning the signal and computing the DFT and the DCT of each piece so that our DCT coefficients only needed to locally represent the signal over a small domain.

Hence, write a function that takes in a signal (image) of size N^2 and partitions it into patches of size 8×8, and for each patch stores the K^2 largest DFT coefficients and their associated frequencies. Your partitioning scheme should resemble the depiction below.



Write another function that executes this procedure for the two dimensional DCT. Try both of these functions out on sample image A for $K^2 = 4$, 16, 32. Make sure to keep track of each patch's frequencies associated with the dominant DCT coefficients.

2.3 Quantization

A rudimentary version of the JPEG compression scheme for images includes partitioning the image into patches, performing the two-dimensional DCT on each patch, and then rounding (or quantizing) the associated DCT coefficients. We describe this procedure below:

1. Perform the DCT on each 8x8 block.
2. $X_{ij}(k,l) = X_C(x_{ij})$, where x_{ij} is the (i,j)th block of the image.
3. We then quantize the DCT coefficients:

$$\hat{X}_{ij}(k,l) = \text{round}\Big[\frac{X_{ij}(k,l)}{Q(k,l)}\Big] \tag{4}$$

$Q(k,l)$ is a quantization coefficient used to control how much the (k,l)th frequency is quantized. Since human vision is not sensitive to these "rounding" errors, this is where the *compression* takes place. That is, a smaller set of pixel values requires less bits to represent in a computer.

The standard JPEG quantization matrix that you should apply to each patch is based upon the way your eye observes luminance, and is given as

$$Q_L = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 36 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{5}$$

Write a function that executes the above procedure. If your code is running too slowly, try using MATLAB or Python's built-in functions.

2.4 Image Reconstruction

Write down a function that takes in the compression scheme in question 2.2, computes the iDCT of each patch, and then stitches these reconstructed patches together to form the global reconstructed signal. Write another function that executes this procedure for the the quantized DCT coefficients from question 2.3.

Run these functions with the results of questions 2.2 and 2.3 as your inputs. That is, you should have one function that takes the patch-wise iDCT of your compressed image, and one function that "un-quantizes" your quantized DCT coefficients, then takes the patch-wise iDCT of the result. Plot the reconstruction error ρ_K versus K for your code from questions 2.2 and 2.3. What do you observe? Are you able to discern what is in the original image?

Play around with the quantization matrix. Do higher or lower values of $Q(k,l)$ yield better reconstruction performance? How much can you alter the entries $Q(k,l)$ and still obtain a compression for which the original image is discernible to your eye?

Try out all of these compression schemes on with different images. How does the compression/reconstruction achieved by the DCT compare if you used the DFT instead (as seen in part I)? How does this compare to the JPEG standard?

2.5 Image denoising using DCT

Repeat Part I-1 (image denoiser) using the DCT. How does the DCT fair against the DFT in imaging? Is it better/worse in general, or is it an artifact of your implementations?

3 Part III – Giving neural networks a shot

Attention: this part is very open-ended. Reach out to the teaching staff for additional pointers if you want to undertake this part in earnest.

Neural networks or increasingly being used in signal processing/imaging applications. In images, usually *convolutional* neural networks are used. For an overview, see[2]

- http://cs231n.github.io/convolutional-networks/
- http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/
- https://en.wikipedia.org/wiki/Convolutional_neural_network

3.1 Can Neural Networks learn the DFT and the DCT?

The DFT is a linear transformation and, as we saw in class, it can be implemented as a matrix multiplication. Thus, it is reasonable to expect that a neural network (which computes non-linear transformations and matrix multiplications) should be able to learn the DFT given enough training data. Can you write a Neural Network that receives an N-dimension complex vector, and outputs the (N-dimension) DFT? You will have to implement a NN that receives a $2N$-dimension real input, and has a $2N$-dimension real output. Repeat this with the DCT. How many layers does the NN need to have?

3.2 Image compression using neural networks

Create a neural network that compresses an image. Clearly describe what criteria you used to measure compression and distortion. How does your approach compare to DCT and DFT?

3.3 Image denoising neural networks

Repeat the denoising tests from Parts I and II using a neural network. You may want to explore "deconvolution" neural networks. You can find an example at

- https://papers.nips.cc/paper/5485-deep-convolutional-neural-network-for-image-deconvoluti
  pdf

and the references therein. How does your solution depend on the noise distribution? How does it compare to the DCT and DFT approach?

3.4 Voice recognition using neural networks.

In problem set 6 we used the DFT for voice recognition. Probably you noticed that the classification performance went down as the method tried to distinguish between more digits. What is the best performance you can achieve? You can go beyond NN if you want (consider, for example, wavelet transforms).

[2] There are many other resources on NN online.