Compressive sensing on images

Physics 305 Activity 4
2nd Sem AY 2022-2023

Mark Jeremy G. Narag
2014-64423
PhD Physics

All the codes and files in this activity are available on my Github:

https://github.com/mgnarag/physics305_computational_imaging

Compressive sensing

Signal to audio, we can also do compressive sensing on images! Remember that audio is 1D signal and image is 2D signal! So just a recall:

Say for example that we let y be the random measurement

$$\boldsymbol{y = Cx}$$

where **C** is the measurement matrix. We can express it as

$$\boldsymbol{y = C\Psi s}$$

where $\boldsymbol{\Psi}$ is our basis function (usually discrete cosine transform) and **s** is what we want to solve

$$\hat{\boldsymbol{s}} = argmin\|\boldsymbol{s}\|_1 \text{ subject to } \boldsymbol{y = C\Psi s}$$

where $\|\cdot\|_1$ is the l_1 norm (Manhattan distance) given by

$$\|\boldsymbol{s}\|_1 = \sum_{k=1}^{n} |s_k|$$

Equation from Ma'am Jing's note on Compressive sensing

1. Divide images into non-overlapping patches

JPEG compression does not actually compress the whole image as one. It actually subdivides the image first into small non-overlapping blocks (typically of size 8 by 8) then perform the compression for each block before stitching it all back.

This is what we will do in this activity. But we don't literally need to divide the images, in MATLAB, we can actually utilize blockproc function.

```
B = blockproc(A,[M N],FUN)
```

It is a distinct block processing where it applies a function FUN on all the subset of the image of size [M N].

Here is an example from my code:

```
out = blockproc(A,[N N],@(block_struct)
comp_sen(block_struct.data,sampling_rate));
```

My input image is `A`. FUN here is `comp_sen()` which is a function I wrote to compressive sense an input matrix `block_struct.data`.

The `block_struct.data` is basically the patches of the images of size N by N where N=8 here.

The `@(block_struct)` ensures that the function will be applied to the patches

`out` will now consists of the compressed image.

Using `blockproc`, we need not to divide the images and stitch it back!

2. Perform compressive sensing on each block using DCT as basis

Similar to audio, we perform compressive sensing on each block. Since this is an image, a 2D signal, we will flatten it so it will now resemble a 1D signal. Then we just do what we do on audio signal. So if we have 8 by 8 block, then signal *f* has 64 data points.

We pick random samples from that 64 data points then create the linear system ***Ax=B***

Just a recall from audio signal compressive sensing:

The condensed signal is a vector *b* of *m* random samples of the original signal *f*. We construct a matrix *A* by extracting *m* rows from the *n*-by-*n* DCT matrix

$$A = D(k,:)$$

where k is the vector of indices used for the sample b. The resulting linear system is in the form ***Ax = b***

3. Find solution to Ax = b

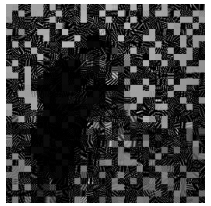
To reconstruct the signal, we need to find the solution to *Ax = b* that minimizes the *l1* norm of *x*.

In Matlab, we will use the *l1*-magic package written by Justin Romberg and Emmanuel Candès. Specifically, we will use the function l1eq_pd

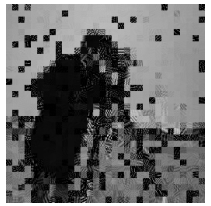
3. Reconstruct

Finally, after getting x, we can reconstruct the signal by getting the DCT of x. But don't forget to resize it since this is 1D. We want 2D ofcourse!

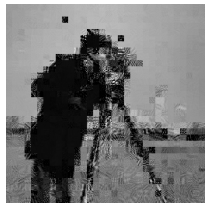
6.25%
psnr = 6.1663
ssim = 0.054939



12.5%
psnr = 12.2213
ssim = 0.38152



18.75%
psnr = 17.1574
ssim = 0.5707



25%
psnr = 19.414
ssim = 0.64675

31.25%
psnr = 20.3378
ssim = 0.68828

37.5%
psnr = 21.6017
ssim = 0.73218

43.75%
psnr = 22.7303
ssim = 0.77008

50%
psnr = 23.8645
ssim = 0.8005

56.25%
psnr = 24.7831
ssim = 0.83188

62.5%
psnr = 26.0305
ssim = 0.85604

68.75%
psnr = 27.3027
ssim = 0.88398

75%
psnr = 28.3867
ssim = 0.90679

81.25%
psnr = 30.2355
ssim = 0.92968

87.5%
psnr = 32.4433
ssim = 0.95247

93.75%
psnr = 36.0324
ssim = 0.9764

100%
psnr = 164.6221
ssim = 1

I tried it first on cameraman built-in image of Matlab. It's a grayscale image.

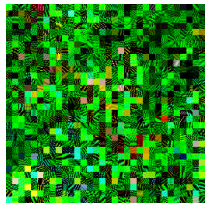
I tried it for different sampling rate from 6.25% to 100%.
6.25% translates to 4 random points from 64 data points
12.5% translates to 8 random points from 64 data points
18.75% translates to 12 random points from 64 data points

Basically, in increments of 4 until I reached 100% (64 out of 64).

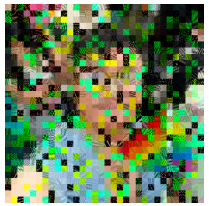
I also computed the PSNR and SSIM for each reconstruction. As expected, PSNR and SSIM increases as we increase the sampling rate.

In contrast to audio where we can reconstruct it only using 5%, on images it seems that 5% is not enough. From my results, we can see that it is more or less acceptable at more than 50%!! *Do Shannon-Nyquist theorem holds here? hmmm*

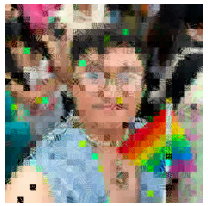
6.25%
psnr = 5.9751
ssim = 0.029798



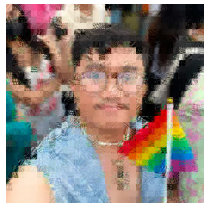
12.5%
psnr = 10.7142
ssim = 0.33766



18.75%
psnr = 16.5327
ssim = 0.64463



25%
psnr = 19.1781
ssim = 0.73958



31.25%
psnr = 21.1434
ssim = 0.79516

37.5%
psnr = 22.3393
ssim = 0.82832

43.75%
psnr = 23.4066
ssim = 0.85936

50%
psnr = 24.7501
ssim = 0.88644

56.25%
psnr = 25.8922
ssim = 0.90706

62.5%
psnr = 26.9926
ssim = 0.92378

68.75%
psnr = 28.4654
ssim = 0.94139

75%
psnr = 29.9652
ssim = 0.95621

81.25%
psnr = 31.885
ssim = 0.97082

87.5%
psnr = 34.1839
ssim = 0.98153

93.75%
psnr = 38.1066
ssim = 0.99244

100%
psnr = 152.0999
ssim = 1

Can we do it tho on colored images? YES!

Actually, in JPEG compression for RGB images, it does not compress on RGB but instead on YCbCr color space!

Here, I performed compression on each of three channels of YCbCr then reconstructed it back to its RGB color space

Since basically it's three channels, the running time is actually three times more than the grayscale image.

From ours results, we see similar trend with the grayscale where PSNR and SSIM increase as we increase the sampling rate. Moreover, if fails on lower sampling rate as we can see on 6.35% to 18.75% with green artifacts. Artifacts are gone starting 25%.

Conclusion:

- We were able to reconstruct the image using compressive sensing both in grayscale and RGB!
- 5% sampling does not work as compared to audio signal. We need more for images. In DCT JPEG compression, it actually works with 10 coefficients only out of 64, that is around 15% sampling rate.

Reflection

Love love love this activity. I do image compression most of the time so this is fun!!! It's actually hard to do it in Matlab since I can't see built-in packages. But I realize that I can convert my image to 1D since that's the code that I have last activity, then just transform it back to 2D. I was very it actually worked!!

CRITERIA	QUALIFICATIONS	SCORE
Technical correctness	• Met all objectives. • Theory is discussed sufficiently. • Procedures and Results are complete. • Procedures and Results are verifiably correct. • Understood the lesson.	40
Quality of presentation	• All text and images are of good quality. • Code has sufficient comments and guides. • All plots are properly labeled and are visually understandable. • The report is clear.	30
Self-Reflection	• Explained the validity of results. • Discussed what went right or wrong in the activity. • Justified the self-score. • Acknowledged sources (e.g. persons consulted, references, etc.)	30
Initiative	• Experimented beyond what was required. **Yes – I did in RGB** • Made significant improvements to existing code. **Yes – automated a code that asks for sampling rate. Also, I transformed my 1D into 2D** • Analyzed limitations or potential of technique, etc. **Yes**	10

100