Image Processing - 67829

Exercise 2: Fourier Transform & Convolution

Due date: 24.11.2022 at 23:59

Version 1.0 - Last update 10.11.2022

The purpose of this exercise is to help you understand the concept of the frequency domain by performing simple manipulations on sounds and images. This exercise covers:

- Implementing Discrete Fourier Transform (DFT) on 1D and 2D signals
- Performing sound fast forward
- Performing image derivative

1 Discrete Fourier Transform - DFT

1.1 1D DFT

Write two functions that transform a 1D discrete signal to its Fourier representation and vice versa. The two functions should have the following interfaces:

```
DFT(signal)             {Discrete Fourier Transform}
IDFT(fourier_signal)    {Inverse DFT}
```

where: `signal` is an array of dtype float64 with shape (N,) or (N,1) , and `fourier_signal` is an array of dtype complex128 with the same shape. The returned values of `DFT` and `IDFT` are the complex Fourier signal and **complex** signal, respectively, both of the same shape as the input. Note that when the `fourier_signal` is a transformed into a real signal you can expect `IDFT` to return real values as well, although it may return with a tiny imaginary part. You can ignore the imaginary part (see tips).

You should use the following formulas:

For DFT transform:

$$F(u) = \sum_{x=0}^{N-1} f(x) e^{-\frac{2\pi i u x}{N}} \tag{1}$$

And for IDFT transform:

$$f(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{\frac{2\pi i u x}{N}} \tag{2}$$

Both functions should be implemented without the use of loops.

1.2 2D DFT

Write two functions that convert a 2D discrete signal to its Fourier representation and vice versa. The two functions should have the following interfaces:

```
DFT2(image)
IDFT2(fourier_image)
```

where: `image` is a grayscale image of dtype float64, and `fourier_image` is a 2D array of dtype complex128, both of shape (M,N) or (M,N,1).

The return shape should be the same as the shape of the input.

Again, when the origin of `fourier_image` is a real image transformed with DFT2 you can expect the returned `image` to be real valued.

You should use the following formulas:

For DFT2:

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-2\pi i (\frac{ux}{N} + \frac{vy}{M})} \tag{3}$$

And for IDFT2:

$$f(x,y) = \frac{1}{NM} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{2\pi i (\frac{ux}{N} + \frac{vy}{M})} \tag{4}$$

Where M and N are the numbers of rows and columns, respectively.

Note1: You should implement these 2D transformation using the `DFT` in Eq. (1) and `IDFT` in Eq. (2). You can loop over one dimension and call 1D transformation on each of the rows/columns.

Note2: you should **not** implement the Fast Fourier Transform.

2 Speech Fast Forward – Simple vs. Spectrogram

2.1 Fast forward by rate change

Write a function that changes the duration of an audio file by keeping the same samples, but changing the sample rate written in the file header. When the audio player uses the same samples as if they were taken in a higher sample rate, a "fast forward" effect is created. Given a WAV file, this function saves the audio in a new file called `change_rate.wav`. You can use the functions read, write from `scipy.io.wavfile`. The function should have the following interface:

```
change_rate(filename, ratio)
```

where: `filename` is a string representing the path to a WAV file, and `ratio` is a positive float64 representing the duration change.
You may assume that $0.25 <$ `ratio` < 4 .
Example: if the original sample rate is 4,000Hz and ratio is 1.25, then the new sample rate will be 5,000Hz. The function should not return anything.

2.2 Fast forward using Fourier

Write a fast forward function that changes the duration of an audio file by reducing the number of samples using Fourier. This function does not change the sample rate of the given file.
Example: if the original file has 10,000 samples, and ratio is 1.25, then the new file will have 8,000 samples.
The result should be saved in a file called `change_samples.wav`.
The function should have the following interface:

```
change_samples(filename, ratio)
```

where: `filename` is a string representing the path to a WAV file, and `ratio` is a positive float64 representing the duration change.
The function should return a 1D ndarray of dtype float64 representing the new sample points. You may assume that $0.25 <$ `ratio` < 4.
This function will call the function `resize` to change the number of samples by the given ratio.

The interface is:

```
resize(data, ratio)
```

where: `data` is a 1D ndarray of dtype float64 or complex128(*) representing the original sample points, and the returned value of `resize` is a 1D ndarray of the dtype of `data` representing the new sample points. This function should call `DFT` (1) and `IDFT` (2). In Fourier representation, you can use `np.fft.fftshift` (and `np.fft.ifftshift` later) in order to shift the zero-frequency component to the center of the spectrum before clipping the high frequencies.

* You will see later why this is necessary.

Note 1: In case of slowing down, we add the needed amount of zeros at the high Fourier frequencies.

Note 2: In case you need to pad with zeros and you have 2 unequal sides, you may choose which side to pad with the one extra 0.

For example, if you have an array with size `25` and you are given a `ratio=0.5`, you will need to pad with `25` zeros, one side with `12` zeros and the other with `13` zeros. In such a case you may choose which side has `13` and which one has `12`.

Note 3: In case you end up with a non-integer, floor it.

Question 1: Run `change_rate` and `change_samples` with the given WAV file `aria_4kHz.wav` and a `ratio` of 2. Listen to both `change_rate.wav` and change_samples.wav. Can you explain the reason for the difference between the two?

2.3 Fast forward using Spectrogram

Write a function that speeds up a WAV file, without changing the pitch, using spectrogram scaling. This is done by computing the spectrogram, changing the number of spectrogram columns, and creating back the audio. The function should have the following interface:

```
resize_spectrogram(data, ratio)
```

where: `data` is a 1D ndarray of dtype float64 representing the original sample points, and `ratio` is a positive float64 representing the rate change of the WAV file. The function should return the new sample points according to `ratio` with the same datatype as `data`.

You may assume that $0.25 <$ `ratio` < 4.

This function should use the provided functions `stft` and `istft` in order to transfer the data to the spectrogram and back. Except for testing, you should use the default parameters for `win_length` and `hop_length`. Alternatively, you could use the functions `stft` and `istft` from `scipy.signal`. Note that the Scipy functions may have a different interface.

Each row in the spectrogram can be resized using `resize` according to `ratio`. Notice that while each row in the spectrogram should be resized correctly according to the ratio, the size of the rescaled 1D array will not be precisely accurate due to the window size.

Consider the resizing of the rows, why is this unintuitive? What could go wrong?

2.4 Fast forward using Spectrogram and phase vocoder

Phase vocoding is the process of scaling the spectrogram as done before, but includes the correction of the phases of each frequency according to the shift of each window. You are provided with the phase vocoding.

Write a function that speedups a WAV file by phase vocoding its spectrogram. The function should have the following interface:

```
resize_vocoder(data, ratio)
```

where: `data` is a 1D ndarray of dtype float64 representing the original sample points, and `ratio` is a positive float64 representing the rate change of the WAV file. The function should return the given data rescaled according to `ratio` with the same datatype as `data`.
You may assume that $0.25 <$ `ratio` < 4.

You can use the supplied function `phase_vocoder(spec, ratio)`, which scales the spectrogram `spec` by `ratio` and corrects the phases. You may also use the function `phase_vocoder` from librosa, which has a different interface.

Question 2: Record yourself using your smartphone (or any other recording device) and fast forward it using both `resize_spectrogram` and `resize_vocoder`. Generate two audio files from the new samples, and explain the differences.

3 Image derivatives

3.1 Image derivatives in image space

Write a function that computes the magnitude of image derivatives. You should derive the image in each direction separately (vertical and horizontal) using simple convolution with $[0.5, 0, -0.5]$ as a row and column vectors. Next, use these derivative images to compute the magnitude image.

The function should have the following interface:

```
conv_der(im)
```

Where the input and the output are grayscale images of type float64, and the output is the magnitude of the derivative, with the same dtype and shape. The output should be calculated in the following way:

```
magnitude = np.sqrt (np.abs(dx)**2 + np.abs(dy)**2)
```

3.2 Image derivatives in Fourier space

Write a function that computes the **magnitude** of the image derivatives using Fourier transform. Use `DFT`, `IDFT`, and the equations from class, to compute derivatives in the x and y directions. Use `np.fft.fftshift` in the frequency domain so that the (U,V)=(0,0) frequency will be at the center of the image, and multiply the frequencies in the range $[-N/2, ..., N/2]$ before shifting back. The function should have the following interface:

```
fourier_der(im)
```

Where the input and the output are float64 grayscale images.
In both sections (3.1 and 3.2) you should **not** normalize the magnitude values to be in the range of [0,1], just return the values you get.
Note: You may not assume the image is square.
Question3: Why did you get two different magnitude images?

4 Some tips

- **Fourier centering:** The output of your `DFT2` implementation is a matrix which contains the Fourier coefficients. This matrix is organized s.t. $F(0,0)$ is located at the origin (0,0) (top left corner). However, visualizing the Fourier coefficients may be easier to do with $F(0,0)$ shifted to the center of the matrix. For this purpose use `np.fft.fftshift` which performs a *cyclic translation* of a matrix in both axes (try to shift a grayscale image to understand the behavior of this function).
  Remember to shift back using `np.fft.ifftshift` before transforming back to the image domain.

- **Fourier display:** To best visualize a Fourier coefficient image `im_f` you should first apply the intensity transformation `np.log(1+np.abs(im_f))` discussed in class. This operation reduces the dynamic range of the coefficient magnitude image and will therefore cause more values to become visible (try to display `im_f` with and without this transformation and compare what you see).

- **Testing:** For the purpose of **your own testing** you may find normalizing the audio output or converting the output to np.int16 will make the results sound better. You should **not** do this for your actual submission. Additionally, you do not need your functions to match the official fft results exactly, using np.isclose() is good enough.
- Calls to resize can take some time even if implemented correctly, up to 10 minutes is within the range of normal (how much can you speed this up?).
- **Useful functions:**
  - `scipy.signal.convolve2d` 2D convolution – use the `'same'` option when you want the output to have the same size as the input
  - `np.meshgrid` used to create index maps, you can use `np.arange` instead, and perform the same operations via broadcasing
  - `np.complex128` dtype of array with complex numbers.
  - `np.fft.fft2, np.fft.ifft2` 2D discrete Fast Fourier Transform (and inverse). You can use these functions to check your results from section 1.1 and 1.2
  - `np.real` (or `np.real_if_close`) When you return from the frequency domain to the image domain, there might be some very small imaginary part in the matrix elements due to numerical errors. You can ignore them and take only the real part of the matrix.

5 Submission

The assignment should be submitted through git. Submission instructions may be found in the "Exercise Guidelines" document published on the course web page.Please read and follow them carefully.

Note 1: Please also include in your solution the function `read_image` from ex 1 (Just copy and paste it into the bottom of your solution).

Note 2: Please also include in your solution the contents of `ex2_helper.py` (Just copy and paste it into the bottom of your solution).

Note 3: You don't need to submit the answers to questions 1-3 but being capable answering those is a good indicator for your understanding the material.

Good luck and enjoy!