1 DCT Transform

1.1 Theory

You have been introduced Principle Component Analysis, which is a data-driven transform. However, it requires a high amount of training data, which is not always available. Therefore, a transform independent from data is a desire. Toward this end, Discrete Cosine Transform (DCT) allows decomposing data from time domain into DCT domain using fixed basic functions. The basis functions of one-dimensional DCT are as follows:

$$\begin{bmatrix} \boldsymbol{p}^{(0)} \\ \boldsymbol{p}^{(1)} \\ \vdots \\ \boldsymbol{p}^{(k-1)} \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{1}{m}}\cos\left(\frac{0}{2m}\right) & \sqrt{\frac{1}{m}}\cos\left(\frac{0}{2m}\right) & \dots & \sqrt{\frac{1}{m}}\cos\left(\frac{0}{2m}\right) \\ \sqrt{\frac{2}{m}}\cos\left(\frac{(2\cdot 0+1)\cdot 1\cdot \pi}{2m}\right) & \sqrt{\frac{2}{m}}\cos\left(\frac{(2\cdot 1+1)\cdot 1\cdot \pi}{2m}\right) & \dots & \sqrt{\frac{2}{m}}\cos\left(\frac{(2\cdot (m-1)+1)\cdot 1\cdot \pi}{2m}\right) \\ \vdots & & & \\ \sqrt{\frac{2}{m}}\cos\left(\frac{(2\cdot 0+1)\cdot (k-1)\cdot \pi}{2m}\right) & \sqrt{\frac{2}{m}}\cos\left(\frac{(2\cdot 1+1)\cdot (k-1)\cdot \pi}{2m}\right) & \dots & \sqrt{\frac{2}{m}}\cos\left(\frac{(2\cdot (m-1)+1)\cdot (k-1)\cdot \pi}{2m}\right) \end{bmatrix}$$

then the j-th transform coefficient is calculated by

$$y_j = \boldsymbol{p}^{(j)} \times [x_0 \; x_1 \; \dots \; x_{m-1}]^T$$

1.2 Python Exercise

The DCT transform has been used widely for image compression such as in JPEG format. In this exercise, you will use DCT transform to do image compression. Steps for the image compression are as follows.

Step 1: Load and display the given image. In the first step, you will load and display an existing image from `scipy` named `face`. The image is in grayscale; therefore, loading the image results in a two-dimensional array. The loading procedure has been done for you.

Step 2: Extract patches, compute DCT coefficients and reconstruct data. In this step, you are going to extract patches of size 4×4 from the loaded image given two functions for extracting patches and re-creating the image from the patches. With the patches, you will compute the DCT coefficients

using function `dct` from `scipy`. Then, you will remove high frequency coefficients and keep the most important coefficients. You will do the reconstruction using these coefficients. The reconstruction can be done with function `idct` (Inverse DCT) in `scipy`.

Step 3: Visualize the reconstructed image. In this step, you will visualize the reconstructed image using `matplotlib`. As the DCT transform is lossy, you will see that the reconstructed image has the lower quality than the original one. You can change the number of lower frequency components in the step 2 to see the effect.

2 Sparse Coding and Dictionary Learning

Sparse Coding and Dictionary Learning is a very important set of technique in machine learning and data representation recently. It has a well-developed mathematical foundation and has been applied to solve a wide range of problems, from signal processing and image processing to computer vision and language understanding.

In this exercise, you will get hand-on experience with both dictionary learning and sparse coding, with the help from built-in functions from `sklearn`.

2.1 Theory

Sparse coding is a class of unsupervised methods to represent data efficiently. Unlike the DCT, Sparse coding is data dependant. Unlike the PCA which represents the data using a complete set of basis, Sparse coding represents the data using an over-complete set of basis. This representation is often constrainted to be sparse, meaning that only a few number of features are non-zero.

Sparse coding often goes with dictionary learning stage. While the former finds the data representations, the latter learns the most suitable dictionary or set of basis for the representations.

2.2 Python Exercise

In this exercise, you will use Dictionary Learning and Sparse Coding to image compression and reconstruction. You will make full use of existing functions in `sklearn`.

Step 1: Load the image We will use the *face* image, available inside the `scipy` package for this exercise. For your convenience, the code for this step is provided. To reduce computational consumption, we will use first resize the image to half its dimensions.

Step 2: Sample image patches As the size of the whole image is 384×512, performing dictionary learning and sparse coding on the whole image is impractically expensive. As a result, we will work with small image patches.

In this step, you need to sample a lot of patches from the image to use in learning the dictionary. You can use the function `extract_patches_2d` from `feature_extraction.image` to do this sampling. We will sample 30K patches of size 4×4 in this step.

Step 3: Normalize the patches for learning dictionary In this step, you need to:

- First, flatten the patches, from 2D arrays to 1D vectors; save all of them as a Numpy array X
- Second, convert X to `float` type
- Third, normalize X by subtract mean and divide by its standard deviation

Step 4: Learn the dictionary We are now all set to learn the distionary or the set o basis for the patches. For efficiency reason, we will use the `MiniBatchDictionaryLearning` function from `sklearn`. As the number of features is $4 \times 4 = 16$, we will use 64 components for the dictionary (to be over-complete). Besides, let's set the number of iterations to 500, number of of non-zero components to 6 and use the *least angle regression method* as the fitting algorithm.

After initializing the learner, call the `fit` function to learn the dictionary.

Step 5: Prepare patches for Sparse coding experiment In this step, we use the provided utility function, `sample_patches` to sample non-overlapping patches from the image. We preprocess these patches by subtract the mean values.

Step 6: Run Sparse coding experiment In this step, you need to loop over different sparsity level, find the sparse representations of the patches using the learned dictionary, and reconstruct the original image. Follow the steps in the code. You should be able to observe the gradual quality improvement of the reconstructed image when the more entries are non-zero.