Nén ảnh fractal

I. Lý thuyết

1. Các định lý

Không gian metric đầy đủ: Giả sử (E, d) là một không gian metric đầy đủ và hàm ánh xạ $f: E \to E$

Ta nói f là một hàm có nếu tồn tại hệ số bất động s sao cho $0 < s < 1$

$$\forall x, y \in E, d\big(f(x), f(y)\big) \leq sd(x, y)$$

Định lý ánh xạ co: Hàm f có duy nhất điểm bất động x_0

⇨ x_0 là điểm bất động của f

Định lý xấp xỉ gốc bởi tập điểm bất động (Collage Theorem):

Nếu tồn tại: $d\big(x, f(x)\big) < \epsilon$ thì $d(x, x_0) < \frac{\epsilon}{1-s}$

⇨ Vậy nếu ta tìm được ánh xạ co $f(x)$ gần bằng với x thì điểm bất động của $f(x)$ cũng gần bằng với x.

⇨ Vậy khi nén ảnh, chúng ta chỉ cần tìm một hàm co sao cho hàm đó chứa điểm bất động gần bằng với ảnh ban đầu.

2. Ứng dụng vào nén ảnh

Mục đích: cần tìm ánh xạ co chứa các điểm bất động gần bằng với điểm ảnh của ảnh gốc.

Giả sử ta ảnh có matrix $E = [0,1]^{h*w}$ với h dòng, w cột và hệ số co s nằm trong khoảng [0, 1], ta có:

$$d(x, y) = \left(\sum_{i}^{h} \sum_{j}^{w} \big(x_{ij} - y_{ij}\big)^2 \right)^{0.5}$$

Với d là khoảng cách tính theo norm = 2

Giả sử điểm $x \in E$ với E là ảnh cần nén, chúng ta chia E thành 2 kiểu như sau:

Domain blocks

D_1	D_2	D_3	D_4
D_5	D_6	D_7	D_8
D_9	D_{10}	D_{11}	D_{12}
D_{13}	D_{14}	D_{15}	D_{16}

Range blocks

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}
R_{17}	R_{18}	R_{19}	R_{20}	R_{21}	R_{22}	R_{23}	R_{24}
R_{25}	R_{26}	R_{27}	R_{28}	R_{29}	R_{30}	R_{31}	R_{32}
R_{33}	R_{34}	R_{35}	R_{36}	R_{37}	R_{38}	R_{39}	R_{40}
R_{41}	R_{42}	R_{43}	R_{44}	R_{45}	R_{46}	R_{47}	R_{48}
R_{49}	R_{50}	R_{51}	R_{52}	R_{53}	R_{54}	R_{55}	R_{56}
R_{57}	R_{58}	R_{59}	R_{60}	R_{61}	R_{62}	R_{63}	R_{64}

Với ảnh gốc(source – Domain blocks): chia ảnh thành các blocks $D_1, \dots, D_k$

Với ảnh đã được nén(destination - Range blocks): chia ảnh thành các blocks $R_1, \dots R_L$

Chúng ta cần xác định hàm f:

$$f(x)_{ij} = f_l\left(x_{D_{kl}}\right)_{ij} \text{ nếu } (i,j) \in R_l$$

Nếu tất cả f_l là các hàm ánh xạ co => f là một ánh xạ co

Vậy làm thế nào để chọn được D_{kl} và f_l để f là ánh xạ co

Giải pháp: Nếu $x_{Rl} \sim f(x_{D_{kl}}) \; \forall \; l \; th$ì $x \sim x_0$

⇨ Vậy hàm ánh xạ co từ mỗi block $D_k \to R_l$ sẽ được chọn là hàm xấp xỉ tốt nhất.

II. Thuật toán nén ảnh fractal

1. Chọn hàm ánh xạ

Source block: 8 x 8

Destination block: 4 x 4

Chọn hàm ánh xạ để biến đổi ảnh:

$$f_l(x_{Dk}) = s * rotate_\theta\left(flip_d\left(reduce(x_{Dk})\right)\right) + b$$

Với $reduce$: hàm giảm size block: 8 x 8 thành block size: 4 x 4

$flip\ v$à $rotate$: là các phép đổi affine

s: thay đổi độ tương phản của ảnh

b: thay đổi độ sáng của ảnh

reduce function: tính trung bình bằng các điểm ảnh lân cận

```
def reduce(img, factor):
    result = np.zeros((img.shape[0] // factor, img.shape[1] // factor))
    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            result[i,j] = np.mean(img[i*factor:(i+1)*factor,j*factor:(j+1)*factor])
    return result
```

flip function: lật lại ảnh nếu ảnh không quay về trước: trả về -1, nếu ảnh đã đúng hướng, trả về 1

```
def flip(img, direction):
    return img[::direction,:]
```

rotate function: Quay ảnh một góc cho trước (angle ∈ {$\mathbf{0}, \mathbf{90}, \mathbf{180}, \mathbf{270}$})

```
def rotate(img, angle):
    return ndimage.rotate(img, angle, reshape=False)
```

apply_transformation: Hàm tính $\boldsymbol{f_l}$

```
def apply_transformation(img, direction, angle, contrast=1.0, brightness=0.0):
    return contrast*rotate(flip(img, direction), angle) + brightness
```

2. Nén ảnh

B1: Tìm tất cả các phép biến đổi affine cho tất cả các block

```
def generate_all_transformed_blocks(img, source_size, destination_size, step):
    factor = source_size // destination_size
    transformed_blocks = []
    for k in range((img.shape[0] - source_size) // step + 1):
```

```
        for l in range((img.shape[1] - source_size) // step + 1):
            # Extract the source block and reduce it to the shape of a destination block
            S = reduce(img[k*step:k*step+source_size,l*step:l*step+source_size], factor)
            # Generate all possible transformed blocks
            for direction, angle in candidates:
                transformed_blocks.append((k, l, direction, angle, apply_transform(S, direction, angle)))
    return transformed_blocks
```

B2: Với mỗi block trong ảnh destination, bằng cách thử ngược lại tất cả phép biến đổi đã lấy được ở trên hàm **generate_all_transformed_blocks,** chọn phép biến đổi có kết quả tốt nhất.

```
def compress(img, source_size, destination_size, step):
    transformations = []
    transformed_blocks = generate_all_transformed_blocks(img, source_size, destination_size, step)
    for i in range(img.shape[0] // destination_size):
        transformations.append([])
        for j in range(img.shape[1] // destination_size):
            print(i, j)
            transformations[i].append(None)
            min_d = float('inf')
            # Extract the destination block
            D = img[i*destination_size:(i+1)*destination_size,j*destination_size:(j+1)*destination_size]
            # Test all possible transformations and take the best one
            for k, l, direction, angle, S in transformed_blocks:
                contrast, brightness = find_contrast_and_brightness2(D, S)
                S = contrast*S + brightness
                d = np.sum(np.square(D - S))
                if d < min_d:
```

```
                    min_d = d
                    transformations[i][j] = (k, l, direction, angle, contrast, brig
htness)
    return transformations
```

Trong mỗi block trên, ta cần tối ưu hàm biến đổi độ sáng và độ tương phản của block ảnh

```
def find_contrast_and_brightness2(D, S):
    # Fit the contrast and the brightness
    A = np.concatenate((np.ones((S.size, 1)), np.reshape(S, (S.size, 1))), axis=1)
    b = np.reshape(D, (D.size,))
    x, _, _, _ = np.linalg.lstsq(A, b)
    return x[1], x[0]
```

3. Giải nén ảnh

Sau khi có ảnh nén, chúng ta chỉ cần sử dụng hàm ánh xạ co cho 1 số lần để giải nén

```
def decompress(transformations, source_size, destination_size, step, nb_iter=8):
    factor = source_size // destination_size
    height = len(transformations) * destination_size
    width = len(transformations[0]) * destination_size
    iterations = [np.random.randint(0, 256, (height, width))]
    cur_img = np.zeros((height, width))
    for i_iter in range(nb_iter):
        print(i_iter)
        for i in range(len(transformations)):
            for j in range(len(transformations[i])):
                # Apply transform
                k, l, flip, angle, contrast, brightness = transformations[i][j]
                S = reduce(iterations[-1][k*step:k*step+source_size,l*step:l*step+s
ource_size], factor)
                D = apply_transformation(S, flip, angle, contrast, brightness)
```

```
                cur_img[i*destination_size:(i+1)*destination_size,j*destination_siz
e:(j+1)*destination_size] = D
        iterations.append(cur_img)
        cur_img = np.zeros((height, width))
    return iterations
```

III. Kết quả

Kết quả của ảnh gốc sau khi nén và giải nén với các độ lỗi tương ứng.

Ảnh 0: Ảnh sau khi nén

Ảnh 1,2,..8: Lần lượt giải nén ảnh tương ứng với 8 vòng lặp.