프로젝트 기말과제 최종 보고서
- Seam Carving with Preservation

수학과 2020540023 이명규

1. 초록

현재 다양한 미디어 매체, 예를 들어 휴대폰과 컴퓨터, TV 등 다양한 기기가 존재한다. 하지만 이러한 기기가 보여주어야 할 사진은 동일하지만, 화면의 크기와 해상도는 모두 다르기에 사진의 크기를 조절할 필요는 늘어가고 있다. 이러한 이미지 크기 조절을 위한 다양한 알고리즘이 있고, 그 중 seam carving의 한계와 개선 방향에 대해 소개한다.

2. 서론

seam carving 알고리즘의 목적은 이미지 리타겟팅(image retargeting)으로, HTML과 같이 문서 표준을 사용하여 다양한 크기의 매체(휴대전화, 프로젝션 스크린)에 이미지를 왜곡 없이 표시하는 문제가 있다.

하지만 이러한 알고리즘에도 다양한 문제와 한계가 있으며, 이러한 한게와 문제를 해결하는 방안들을 찾아보고, 특히 3.4절에서는 제한적인 상황에서 개선된 알고리즘을 제안하고 이를 실제로 적용하는 예시르 살펴본다.

3. 본론

3.1 동적 계획법 (Dynamic Programming)

동적 계획법(dynamic programming)은 수학적 최적화 방법이자 알고리즘 패러다임 중 하나이다. 다음과 같은 과정을 거치며 원래의 문제를 해결한다. 우선 주어진 문제를 풀기 위해서, 문제를 여러 개의 하위 문제(subproblem)로 나누어 하위 문제를 해결한다. 이후, 그것들을 결합하여 최종적인 목적에 도달한다. 특히, 각 하위 문제를 해결한 뒤, 그 해결책들을 저장하여 후에 같은 하위 문제가 나왔을 경우 이미 풀었던 문제이기 때문에 그것을 간단하게 해결할 수 있다. 이러한 방법으로 동적 계획법은 계산 횟수를 줄일 수 있다.

실제로 동적 계획법을 이용하여, 조합 최적화 문제 중 하나인 배낭 문제를 $O(NW)$의 사간 복잡도로 해결하는 풀이를 [부록 1]에서 확인할 수 있다. 참고론 배낭 문제는 배낭에 담는 모든 경우를 전부 살펴보는 완전 탐색을 적용하면 $O(2^N)$의 사간 복잡도로 풀이할 수 있다.

3.1.1 최적화 원리 (Principle of Optimality)

동적 계획법은 최적화 원리(principle of optimality)를 만족하는 문제에 대해 적용할 수 있다. “어떤 문제의 최적해의 부분해가 부분문제의 최적해를 만족한다”면 그 문제는 principle of optimality를 만족한다고 이야기한다. (A problem is said to satisfy the Principle of Optimality if the subsolutions of an optimal solution of the problem are themselves optimal solutions for their subproblems.)

그리고 이는 다음의 두 가지 조건으로 나누어 설명할 수 있다.

(1) 최적 부분 구조 (optimal substructure) : 최적해가 부분문제의 최적해로부터 구성될 수 있다.

(2) 중복되는 부분 문제 (overlapping subproblems) : 어떤 문제가 여러 번 재사용되는 하위 문제로 분해된다. 또는 문제에 대한 재귀 알고리즘이 항상 새로운 하위 문제를 생성하는 것이 아니라 반복적으로 동일한 하위 문제를 해결한다.

최적화 원리가 성립하는 예시와 성립하지 않는 예시는 아래와 같다.

그림 1 최단 경로

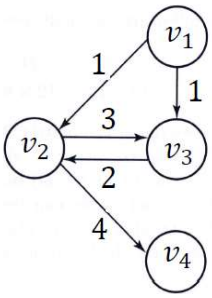

그림 2 최장 경로

먼저, 어느 두 지점 간의 최단 경로(shortest path)를 구하는 문제는 최적화 원리가 성립한다. 위의 [그림 1]에서 s-t 사이의 최단 경로 중 s-w 부분과 w-t 부분으로 나누어 생각한다면, 각각은 s-w와 w-t 사이의 최단 경로라는 것을 확인할 수 있다. 이에 대한 증명은 귀류법을 통해 가능할 것이다. 어떤 부분에 대해 더 짧은 경로가 존재한다면, 전체에서의 최단 경로 역시 해당 경로를 거치는 것이 더 짧으므로 해당 부분에서 더 짧은 경로가 존재한다는 것은 모순이다.

다음으로, 어느 두 지점 간의 최장 경로(longest path)를 구하는 문제는 최적화 원리가 성립하지 않는다. 이는 위의 [그림 2]에서 반례를 확인함으로써 알 수 있다. v1에서 v4까지의 최장 경로는 v1, v3, v2, v4를 차례대로 방문하는 경로이고, 이때의 길이는 1+2+4=7이다. 하지만 v1에서 v3까지의 최장 경로는 v1, v2, v3를 차례대로 방문하는 경로이고, 이때의 길이는 1+3=4이다. 즉, v1에서 v4까지의 최장 경로를 구하는 원래 문제의 부분 문제에 대해 부분해가 최적해를 만족하지 않는다.

3.2 Seam Carving

seam carving은 내용 인식 이미지 크기 조정(Content-Aware Image Resizing)을 위한 알고리즘 중 하나이다. 이미지의 크기를 조절하면서, 이미지의 내용은 유지할 수 있도록 고안되었다. 이 알고리즘은 이미지에 다수의 seam(중요도가 가장 낮은 경로)을 설정한 뒤, seam을 제거하거나 삽입함으로써 이미지의 크기를 줄이거나 확장한다. seam의 예시는 [부록 2]에서 확인할 수 있다.

실제로, 이미지의 크기를 조절하는 다양한 방법이 있다. 이미지의 전체적인 비율을 조절하는 scaling, 이미지의 특정 부분을 잘라내는 cropping, 그리고 앞으로 살펴보게 될 seam carving이 바로 그것이다. [부록 3]에서 실제 이미지에 적용한 예시와 각 방식에 대한 설명을 살펴볼 수 있다.

그림 3 원본 이미지와 Seam Carving을 적용한 이미지

3.2.1 Optimization Problem

seam carving의 문제를 optimization problem으로 모델링하자.

(1) 우선 seam의 정의는 다음과 같다.

- Vertical seam : $s^X = \{(x(i), i)\}_{i=1}^{H},\ s.t. \quad \forall i \geq 2, |x(i) - x(i-1)| \leq 1$

- Horizontal seam : $s^Y = \{(j, y(j))\}_{j=1}^{W},\ s.t. \quad \forall j \geq 2, |y(j) - y(j-1)| \leq 1$

이때 x와 y는 다음과 같은 mapping을 이야기한다. 즉, x는 i에 대하여 seam의 i번째 행의 픽셀이 위치하는 열의 번호를 대응시켜주며, y 역시 j에 대해 그러하다.

$$x : \{1, \dots, H\} \rightarrow \{1, \dots, W\},\ \ y : \{1, \dots, W\} \rightarrow \{1, \dots, H\}$$

또한, 차의 절댓값이 1이라는 조건은 seam의 각 픽셀이 인접한 행에 대해 기껏해야 한 칸 떨어져 있다는 의미이며, seam이 끊어지지 않고 이어짐을 보장한다.

(2) objective function은 다음과 같다.

$$f_0 = \sum_{i=1}^{H} E(x(i), i)$$

이때 E는 energy function을 의미하며, 주변 픽셀과의 색상 차이를 기반으로 중요도를 계산하는 함수이다. 실제 구현에서는 파이썬 cv2 라이브러리의 sobel 함수를 이용하여 중앙 차분을 계산하였다. [그림 3]의 원본에 대한 에너지 이미지를 [부록 4]에서 볼 수 있다.

주의해야 하는 부분은 이미지의 (x(i), i)에는 하나의 수가 아니라, RGB를 나타내는 3개의 정수로 이루어진 벡터가 존재하기 때문에, energy도 스칼라(정수)가 아닌 벡터라는 것이다. 따라서 적절한 대소 비교 방법으로 2-norm을 통한 대소 비교를 선택할 것이다.

(3) 최종적으로 optimization problem은 다음과 같다,

$$\min_{x} f_0 = \sum_{i=1}^{H} E(x(i), i)$$

$$subject\ to \quad |x(i) - x(i-1)| \leq 1, \qquad i = 2, \dots, H$$

특이한 점은 objective function을 최소화하는 mapping x를 찾는다는 것이다. 제약조건을 seam의 정의를 이용해서 없애고 표현하면, 아래 식과 같다.

$$\min_{S^X} f_0 = \sum_{x,y \in S^X} E(x, y)$$

또한, 문제 풀이 및 구현에서는 사용하지 않았지만, 동일한 문제를 다양한 형태로 표현한 것을 [부록 5]에서 확인할 수 있다.

3.2.2 Seam Carving with Dynamic Programming

$W \times H$ 크기의 이미지에 대해 vertical seam을 구하는 문제는 한 칸마다 다음 행에 대해 '왼쪽 아래, 바로 아래, 오른쪽 아래'의 3가지 선택지가 있으므로, 완전 탐색을 적용하면 대략 $O(3^{H-1})$의 시간복잡도를 가진다. 또한 해당 문제는 위에서 아래로 이동하는 최단 경로를 문제라고 생각할 수 있으므로 최적화 원리(Principal of optimality)가 성립한다. 따라서 동적 계획법을 적용하여 해당 문제를 풀 수 있고, 이때의 시간복잡도는 $O(WH)$가 된다.

[부록 1]에서 배낭 문제에 적용한 것처럼 DP 행렬을 다음과 같이 정의하자.

DP_{ij} := 맨 위부터 i번째 행까지의 이미지에 대해, (i, j)에서 끝나는 $seam$으로 가능한 최소의 에너지

이때 최적화 원리가 성립하므로, 아래와 같은 점화식을 찾을 수 있다. 아래 점화식에 의해 최소 에너지를 가지는 경로가 선택된다. 또한, [부록 6]에서 Seam Carving 문제를 동적 계획법을 적용하여 해결하는 python 스타일의 pseudo-code를 확인할 수 있다.

$$DP_{i,j} = E(i,j) + min\left(DP_{i-1,j-1}, DP_{i-1,j}, DP_{i-1,j+1}\right)$$

3.3 Seam Carving for Region Preservation

seam carving가 모든 이미지에 잘 적용된다면 좋겠지만, 항상 그러하지는 않다. 아래의 [그림 15]에서 기존 seam carving이 잘 적용되지 않은 원인은, 실제로는 중요하지 않은 영역(고양이의 수염)의 에너지가 크게 계산되었기 때문이라는 것을 알 수 있다. 이에 따라 실제로는 중요하지만, 에너지가 상대적으로 작게 계산된 영역(예시의 '얼굴')이 지워진 것이다.

그림 15 왼쪽부터 원본 이미지, 에너지, 기존 seam carving, 수정된 seam carving

이러한 문제를 방지하기 위해 특정 영역은 살리면서, 이외의 영역에서 seam을 찾는 방법을 생각하자. extended-value extension([부록 7] 참고)의 아이디어를 가져와서, 살리고 싶은 영역의 에너지를 매우 크게 주어서 해당 영역에서는 최적해(seam)를 갖지 못하도록 강제할 것이다. 이를 위해 기존의 optimization problem을 다음과 같이 수정해보자,

$$\min_{x} f_0 = \sum_{i=1}^{H} \tilde{E}(x(i), i)$$

$$subject\ to \quad |x(i) - x(i-1)| \le 1, \qquad i = 2, \dots, H$$

$$\tilde{E}(x,y) = \begin{cases} E(x,y) & \notin P \\ \infty & \in P \end{cases}$$

여기서 P는 보존하고자 하는 영역이다. $\tilde{E}$는 P에서 ∞ 값을 가지고, 이외의 영역에서는 원래의 에너지 값을 가지도록 확장한 함수이다. 동적 계획법을 적용할 때는 똑같은 DP 행렬을 적용하되, 점화식의 형태를 다음과 같이 수정하면 된다.

$$DP_{i,j} = \tilde{E}(i,j) + min\left(DP_{i-1,j-1}, DP_{i-1,j}, DP_{i-1,j+1}\right)$$

수정된 알고리즘의 결과로, 고양이의 얼굴이 잘 유지된 것을 볼 수 있다. ([부록 7] 참고)

3.4 Seam Carving with Dominant Color

3.3절의 수정된 알고리즘을 적용하기 위해서는 어느 영역을 살리고 싶은지 미리 지정해야 했다. 즉, 이미지에 대한 특징을 파악하고 있어야 한다는 것이다. 그렇다면 이미지에 대한 사전 정보 없이, 자동으로 보존을 할 수 있는 방법은 없을까?

일반적인 사진에 대해서는 쉽지 않을 것이다. 대신, 조금 더 쉬운 문제를 생각하자. 해당 이미지의 중요한 정보 혹은 객체가 전체적으로 비슷한 색을 가진 상황을 생각하자. 예컨대, 위의 [그림 15]에서는 중요한 객체(고양이)가 공통적인 색으로 고양이의 털 색을 가진다. 이러한 공통된 색을 Dominant Color(대표색)라고 하자.

dominant color를 가지는 이미지에 대한 새로운 objective function과 optimization problem으로 아래의 문제를 제안한다. (식의 자세한 유도과정은 [부록 9] 참고)

$$\min_{S^X} f_0 = \beta \sum_{x,y \in S^X} \frac{E(x,y) - \mu_E}{\sigma_E} + (1-\beta) \sum_{x,y \in S^X} \frac{\|I(x,y) - D\|_2 - \mu_{I-D}}{\sigma_{I-D}}$$

여기서 $I(x,y)$는 (x, y)에 존재하는 픽셀의 색상(RGB 벡터)를 나타내며, D는 dominant color의 색상(RGB 벡터)을 나타낸다. 또한, β는 두 항의 비율을 조정하는 factor이다. 또한, 실제 구현에서는 colorthief라는 파이썬 라이브러리를 이용하여 dominant color를 구하였다. 개선된 알고리즘을 통한 결과는 아래 그림과 같다.

그림 25 3.3절(좌), 새로운 알고리즘(우)

상대적으로 3.3절의 알고리즘보다 눈, 코, 입의 중앙 몰림이 덜한 것을 확인할 수 있다.

4. 결론

본론의 3.1절, 3.2절에서는 동적 계획법과 seam carving의 이론적 개념을 살펴보았고, seam carving의 objective function과 optimization problem을 정의하였다. 3.3절에서는 기존의 seam carving 알고리즘이 가지는 한계를 살펴보았고, 주요 객체의 위치를 알 때 이를 해결하는 개선된 알고리즘을 보았다. 하지만 이것 역시 이미지에 대한 사전 정보를 알아야 한다는 한계를 가졌기에, 3.4절에서는 dominant color를 가지는 이미지에 대한 새로운 알고리즘을 제시하였다.

새롭게 제안한 알고리즘은 이미지가 dominant color를 가진다는 것을 안다면, 주요 객체의 위치를 모르더라도 적용할 수 있다. 하지만 기존 seam carving에 비해 이미지의 형태가 온전하지는 못하다는 단점과 β를 정하는 기준에 대해서는 제시하지 못했다는 한계가 있다.

5. 참고문헌

1. Wikipedia, Dynamic programming
https://en.wikipedia.org/wiki/Dynamic_programming
2. 위키백과, 동적 계획법
https://ko.wikipedia.org/wiki/%EB%8F%99%EC%A0%81_%EA%B3%84%ED%9A%8D%EB%B2%95
3. Wikipedia, Seam Carving
https://en.wikipedia.org/wiki/Seam_carving
4. Wikipedia, Overlapping subproblems
https://en.wikipedia.org/wiki/Overlapping_subproblems
5. tistory, 6-3강 - Dynamic Programming 3 (the principal of optimality, 행렬 곱셈)
https://intelligentcm.tistory.com/268
6. Dynamic Programming
https://www2.seas.gwu.edu/~ayoussef/cs6212/dynamicprog.html
7. Knapsack Problem
https://en.wikipedia.org/wiki/Knapsack_problem
8. Seam carving for content-aware image resizing
https://dl.acm.org/doi/10.1145/1276377.1276390
9. Supervised Deep Learning for Content-Aware Image Retargeting with Fourier Convolutions, https://arxiv.org/abs/2306.07383
10. Algorithms for Optimization, Mykel J. Kochenderfer , Tim A. Wheeler, The MIT Press · 2019.03.01, p.351
11. dynamic programming and optimal control 3rd edition, dimitri p. bertsekas , p.18

- 이미지 출처
https://intelligentcm.tistory.com/268
https://namu.wiki/w/%EB%B0%B0%EB%82%AD%20%EB%AC%B8%EC%A0%9C
https://convex-optimization-for-all.github.io/contents/chapter24/2021/03/28/24_01_Definition/
https://pixabay.com/photos/blue-beach-surf-travel-surfer-4145659/

부록

1. Knapsack problem과 Dynamic Programming을 적용한 풀이

동적 계획법이 실제 문제에서 어떻게 적용되는지, 조합 최적화의 유명한 문제 중 하나인 '배낭 문제(knapsack problem)'를 풀어보며 살펴보자. 우선, 배낭 문제는 아래와 같은 문제를 일컫는다. "각각의 무게와 가치가 주어진 물건들의 집합에 대해, 무게의 총합은 제한보다 작으면서 가치의 총합을 최대화하는 부분집합을 찾는 문제". 따라서 아래의 [그림 3]과 같은 최적화 문제로 나타낼 수 있다.

$$\begin{aligned} \max_{x} \quad & c^{\mathsf{T}}x \\ \text{subject to} \quad & a^{\mathsf{T}}x \leq b \\ & x_j \in 0,1, j = 1,\ldots,n \end{aligned}$$

그림 27 배낭 문제

$$\begin{aligned} \max_{x} \quad & 4x_1 + 2x_2 + 10x_3 + x_4 + 2x_5 \\ subject\ to \quad & 12x_1 + x_2 + 4x_3 + x_4 + 2x_5 \leq 15 \\ & x_i \in 0,1, \qquad j = 1,2,3,4,5 \end{aligned}$$

그림 28 실제 예시

구체적으로, 5개의 물건이 있고, 각각의 무게는 12, 1, 4, 1, 2kg이며, 가치는 4, 2, 10, 1, 2$인 문제를 생각하자. 마찬가지로 [그림 4]와 같은 최적화 문제로 나타낼 수 있다. 동적 계획법을 적용하기 위해 DP라는 행렬을 다음과 같이 정의한다.

DP_{ij} := i번째 물건까지 고려했을 때, 무게가 j 이하인 조합 중 가치가 최대인 경우

또한 $DP_{i,j} = \max(DP_{i-1,j}, DP_{i-1,j-w_i} + V_i)$ 와 같은 점화식을 만족한다.

위의 정의와 점화식에 따라 DP 행렬의 각 원소를 계산하면 아래 표와 같고, 구하고자 하는 문제의 답은 15라는 것을 알 수 있다.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	0	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

그림 29 그림 4에서의 문제에 대한 DP 행렬

2. Seam의 예시

그림 30 seam의 예시

3. 이미지 조절 예시

그림 31 이미지의 크기를 조절하는 예시 (왼쪽부터 Scaling, Cropping, Seam Carving)

scaling은 이미지의 비율을 조절하면서 형태가 왜곡되었다. cropping의 경우에는 이미지가 잘린 절단선 부분이 부자연스럽게 이어진다. 반면, seam carving은 상대적으로 이미지의 형태나 내용은 유지하면서 크기를 조절하고 있다.

4. Energy의 예시 : cv2.Sobel 적용

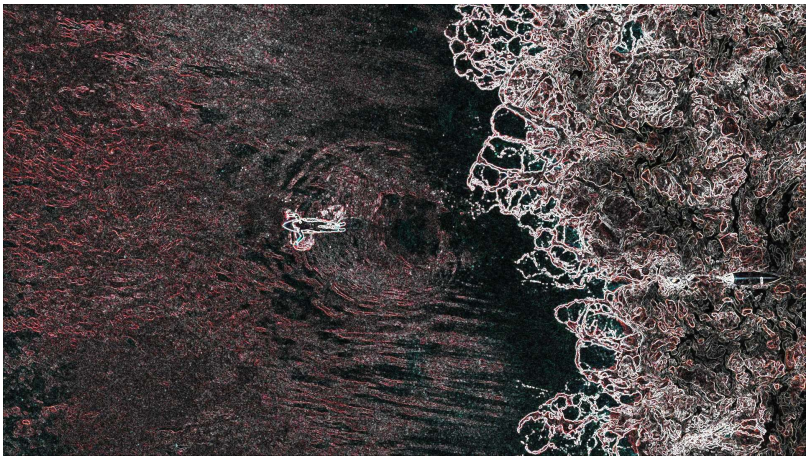

그림 32 소벨 필터를 적용하여 계산된 에너지의 이미지

5. 다양한 형태로 나타낸 Seam Carving의 Optimization Problem

$$\min f_0 = \sum_{j=1}^{W}\sum_{i=1}^{H} E(j,i) \times x_{j,i}$$

$$subject\ to \qquad x_{j,i}, \in \{0,1\}$$

$$\sum_{j=1}^{W} x_{j,i} = 1, \qquad i = 1, \dots, H$$

$$x_{j_i,i} = x_{j_{i-1},i-1} = 1 \Rightarrow \left|x_{j_i,i} - x_{j_{i-1},i-1}\right| \le 1, \qquad i = 2, \dots, H, \qquad j_i = x(i)$$

$$\min_X f_0 = tr(EX)\ (E \in M_{H\times W}, X \in M_{W\times H})$$

$$subject\ to \qquad X_{j,i} \in \{0,1\}$$

$$1^T v_i = 1, \qquad i = 1, \dots, H, \qquad v_i\ is\ column\ of\ X$$

$$|X_{ij} - X_{i-1,j}| \le 1, \qquad i = 2, \dots, H$$

$$X_{j_i,i} = X_{j_{i-1},i-1} = 1 \Rightarrow |X_{j_i,i} - X_{j_{i-1},i-1}| \le 1, \qquad i = 2, \dots, H$$

$$E = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}, \qquad X = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad EX = \begin{pmatrix} 3 & 3 & 2 \\ 7 & 7 & 6 \\ 11 & 11 & 10 \end{pmatrix}, \qquad tr(EX) = 20$$

6. seam carving을 수행하는 python 스타일의 pseudo-code

```
def seam_carving(image, num_iterations):
    for iter in range(num_iterations):
        dp = calculate_energy(image)
        h, w = image.shape
        reverse = zeros((h, w), dtype=int)

        for i in range(1, h):
            for j in range(w):
                temp = min(dp[i - 1, j - 1], dp[i - 1, j], dp[i - 1, j + 1])
                for jj in [j - 1, j, j + 1]:
                    if temp == dp[i - 1, jj]:
                        reverse[i, j] = jj
                dp[i, j] += temp

        start = np.argmin(dp[h - 1])
        delete_idx = [(h - 1) * w + start]
        for row in range(h - 2, -1, -1):
            start = reverse[row + 1, start]
            delete_idx.append(row * w + start)
        image = delete(image, delete_idx)
    return image
```

$$reverse[i,j] = \begin{cases} j-1 & if\ DP_{i,j} = E(i,j) + DP_{i-1,j-1} \\ j & if DP_{i,j} = E(i,j) + DP_{i-1,j} \\ j+1 & if DP_{i,j} = E(i,j) + DP_{i-1,j+1} \end{cases}$$

이때 reverse는 위와 같이, DP가 최소 에너지 값을 갖게 하는 sub-seam을 저장하는 역할을 한다.

실제 구현에서 사용한 calculate_energy 함수는 아래 이미지와 같다.

```
def calculate_energy(image):
    dx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    dy = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    energy = np.abs(dx) + np.abs(dy)
    return energy
```

7. Extended-value extension

Extended-value extension

extended-value extension $\tilde{f}$ of f is

$$\tilde{f}(x) = f(x), \quad x \in \mathbf{dom}\, f, \qquad \tilde{f}(x) = \infty, \quad x \notin \mathbf{dom}\, f$$

often simplifies notation; for example, the condition

$$0 \le \theta \le 1 \implies \tilde{f}(\theta x + (1-\theta)y) \le \theta \tilde{f}(x) + (1-\theta)\tilde{f}(y)$$

(as an inequality in $\mathbf{R} \cup \{\infty\}$), means the same as the two conditions

- $\mathbf{dom}\, f$ is convex
- for $x, y \in \mathbf{dom}\, f$,

$$0 \le \theta \le 1 \implies f(\theta x + (1-\theta)y) \le \theta f(x) + (1-\theta) f(y)$$

8. Dominant Color를 이용한 Seam Carving 식 유도

기존의 seam carving 식은 아래와 같다.

$$\min_{S^X} f_0 = \sum_{x,y \in S^X} E(x,y)$$

이 식에 대해 각 픽셀의 색상(RGB 벡터)과 dominant color과의 거리에 대한 항을 추가하고, 기존의 항과의 비율을 조절하는 factor β를 도입하자. 이때 I(x, y)는 (x, y) 픽셀에서의 RGB 벡터를 나타내며, D는 dominant color의 RGB 벡터를 나타낸다.

$$\min_{S^X} f_0 = \beta \sum_{x,y \in S^X} E(x,y) + (1-\beta) \sum_{x,y \in S^X} \|I(x,y) - D\|_2$$

하지만 $E(x,y)$와 $\|I(x,y) - D\|_2$의 범위와 분포가 다르므로, 아래와 같이 각각의 평균과 표준편차를 이용하여 Standardization을 진행한다.

$$\min_{S^X} f_0 = \beta \sum_{x,y \in S^X} \frac{E(x,y) - \mu_E}{\sigma_E} + (1-\beta) \sum_{x,y \in S^X} \frac{\|I(x,y) - D\|_2 - \mu_{I-D}}{\sigma_{I-D}}$$

9. Dominant Color를 계산하는 colorthief 라이브러리의 예시