Computational Photography Final

20170243 심재윤

#1.

Image의 color를 설명하기 위해서는 color model과 같은 수단이 필요하며, color model은 숫자들의 tuple로서 color를 설명하려는 추상적이고 수학적인 model이다. Color model은 model의 각 color가 해당 subspace에 포함된 하나의 point로 표현되도록 해당 시스템 내의 좌표계 및 subspace로 정의가 된다. 대표적으로 가장 많이 사용하는 color model로는 3차원에서의 cube 형태의 subspace로 정의가 되는 RGB color model이 있다.

Color space는 color model과 비슷하지만 조금은 다른 컨셉을 가지고 있어서, color와 특정 tuple이 color로 mapping이 되는 구체적인 function의 조합이다. 만약, mapping function이 없다면 이 tuple은 그저 임의의 숫자일 뿐이고, 예를 들어 RGB color model을 기반으로 서로 다른 mapping function을 가지는 color space가 여러 개 존재할 수 있다. 같은 (255, 0, 0) tuple이라도 red를 나타낼 수도 있고 orange를 나타낼 수도 있는 것이다. 여기서 mapping function의 경우는 해당 color가 가지고 있는 전자기파의 wave length 등에 의해서 결정하곤 한다.

#2.

CIE RGB color space는 3개의 primary color를 어떻게 섞어서 특정 color를 만들어내는지 설명할 수 있다. 그러나 CIE RGB color space는 3개의 primary color가 인식되는 color의 전체 범위를 나타내지 못하는 문제가 있다. 그래서 CIE XYZ color space가 새롭게 등장했으며, 여기서는 linear transformation을 사용해서 CIE RGB color space로부터 만들어질 수 있다. 이렇게 CIE XYZ color space를 만들게 되면 CIE RGB color space에 비해서 더 나은 특징을 가지게 된다. 가령 white point를 정의하고 싶을 때, X, Y, Z 모두 1/3에서 정의할 수 있다. 그리고 Y는 luminosity function이 되어 perceived brightness 혹은 photometry에 대응되게 된다. CIE XYZ color space를 사용하면 모든 color와 파장을 줄이기 위해서 더 이상 negative value를 가지지 않아도 된다.

#3.

Alpha blending은 기존의 cut-and-paste의 한계점을 보완하는 image blending 기법으로, alpha blending을 하기 위해서는 foreground image와 background image가 필요하게 된다. 그리고 어떻게 foreground image를 background image에 blending 해야하는지 설명해주는 일종의 도구로서

mask가 필요하다. Mask는 1과 0으로 구성되어 있으며 white가 1을 가리키게 되고 이를 foreground로부터 pixel 값을 이용해서 계산하게 된다. Black은 0을 가리키고 이를 이용하면 background image로부터 pixel값을 이용해서 계산이 가능하다. 다만 binary alpha mask를 사용하게 되면 경계선이 너무 뚜렷하는 문제가 생겨 결과를 보면 부자연스러워 보이게 된다. 그래서 feathering이라는 개념으로부터 non-binary alpha mask를 만들어 결과적으로 smooth하게 만들 수 있다. Feathering은 lowpass filter를 binary mask에 사용해서 만들 수 있으며, 이제 0과 1이 아닌 0.5와 같이 0에서 1사이의 실수 범위로 바뀌게 된다. 그래서 결과적으로 foreground와 background 사이의 경계선이 smooth해지게 되어 기존 binary mask를 쓴 결과보다 자연스러워진다. 이러한 아이디어가 alpha blending에 사용되었으며, mask를 만들 때 실제로는 겹치는 부분의 경우 0과 1 사이의 어떠한 값으로 설정해야 한다. 이 중간 경계 부분을 transition area라고 하는데 핵심은 이 크기를 어떻게 조절하는지가 관건이다. 만약 너무 좁으면 부자연스럽게 이어질 것이고, 너무 넓으면 투명하게 겹쳐 보이는 ghosting과 같은 현상이 발생한다. 그래서 이 범위를 잘 설정하고자 window를 사용하게 되며, window의 크기는 feature의 크기에 보통 맞추게 되는데, 만약 크기가 다른 feature들이 여러군데에 나타나게 된다면 적절한 크기를 설정하는데 어려움이 발생하게 된다. 그래서 이러한 alpha blending에서의 문제점을 보완하고자 multi-band blending을 사용할 수 있다.

Multi-band blending 2개의 image에 대해서 나타나는 구조들의 scale에 따라서 사용하고자 하는 window의 크기를 다르게 하고자 한다. 만약에 특정 구조가 크게 나타나면 low frequency component가 있다는 것인데, 이러한 경우에 경계선을 피하고 싶다면 천천히 blending을 하거나 크기가 큰 window를 사용하면 된다. 반대로 scale이 작은 구조에 해당하는 high frequency component가 있을 때는 ghosting을 피하고자 빠르게 blending을 하거나 크기가 작은 window를 사용해야 한다. 그래서 frequency component가 다르게 사용되는 구조들에 대해서 transition window의 크기를 다르게 해서 ghosting과 같은 문제를 피하고자 하는 것이다. 그래서 multi-band blending에서는 laplacian pyramid를 사용해서 각 level마다 alpha blending을 이용하고 이로부터 새로운 Gaussian image pyramid를 얻어서 더해가는 식으로 blending을 진행하게 된다.

#4.

Homogrpahy는 평면의 2D image translation을 설명할 수 있는 가장 일반적인 model로 homogeneous coordinate에서 다음과 같이 정의가 된다.

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography는 degree of freedom이 8이며 따라서 이를 결정하기 위해서는 최소 4개의 matching points들이 필요하다. 우리가 원하는 것은 위의 homography matrix H를 구하는 것이다.

이는 4개의 point 쌍들로부터 계산이 가능하며, 만약 homography matrix H를 구하고 나면 어떠한 point로부터 translation이 가능해지게 된다. 위의 식으로부터 우리는 다음과 같이 pair에 대해서 식을 정리할 수 있다.

$$wx' = ax + by + c$$

$$wy' = dx + ey + f$$

$$w = gx + hy + 1$$

이 식을 다시 정리하면 다음과 같다.

$$x' = \frac{ax + by + c}{gx + hy + 1}$$

$$y' = \frac{dx + ey + f}{g + hy + 1}$$

이 식을 동일하게 다음과 같이 정리할 수 있다.

$$ax + by + c - x'(gx + hy + 1) = 0$$

$$dx + ey + f - y'(gx + hy + 1) = 0$$

그러나 이렇게 정리한 식을 만족시키는 H를 구하는 것은 일반적으로 불가능하다. Noise나 error가 존재할 수 있기 때문에, 이 대신에 다음과 같이 squared error의 합을 최소로 만드는 H를 추정하고자 한다. 그리고 이 형태는 다음을 만족하는 A와 H의 inner product 꼴로 나타낼 수 있다.

$$\min_H \sum_i \{ax_i + by_i + c - x'_i(gx_i + hy_i + 1)\}^2 + \{dx_i + ey_i + f - y'_i(gx_i + hy_i + 1)\}^2$$

$$= (AH)^T AH = H^T A^T AH$$

$$\text{where } A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & \vdots & & & \vdots & & & \vdots & \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4 x_4 & -x'_4 y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4 x_4 & -y'_4 y_4 & -y'_4 \end{bmatrix}, H = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ 1 \end{bmatrix}$$

그러면 다음의 least square problem을 푸는 것과 같아진다.

$$H^* = \underset{H}{\operatorname{argmin}}\, H^T A^T AH$$

$$\text{where } A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & \vdots & & & \vdots & & & \vdots & \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4 x_4 & -x'_4 y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4 x_4 & -y'_4 y_4 & -y'_4 \end{bmatrix}$$

이 문제는 이제 eigen decomposition으로 풀면 되고 여기서 H^*는 가장 작은 eigenvalue에 대응되는 eigenvector를 구하는 것과 같다. 그러기 위해서 다시 제대로 정의를 하면 다음과 같다.

$$H^* = \underset{H}{\operatorname{argmin}} H^T A^T A H \text{ subject to } \|H\| = 1$$

여기서 위와 같은 조건이 없으면 H=0 인 solution으로 H라는 matrix 자체가 homogeneous이기에 scale 자체가 중요하지 않아서 1로 두게 된다. 이제 e_i를 $A^T A$의 i번째 eigenvector로 두고, λ_i를 여기에 대응하는 eigenvalue라고 할 것이다. 그러면 어떠한 vector x 도 eigenvector의 linear combination으로 나타낼 수가 있다. Eigenvector들은 orthogonal한 성질이 있다.

$$\mathrm{x} = \mu_1 e_1 + \cdots + \mu_9 e_9 \text{ s.t. } \sum_i \mu_i^2 = 1$$

이렇게 되면 $H^T A^T A H$는 가장 작은 eigenvlaue λ_1에 의해서 bound 된다.

$$H^T A^T A H - e_1^T A^T A e_1 = (\lambda_1 \mu_1)^2 + \cdots (\lambda_9 \mu_9)^2 - \lambda_1^2 \geq \lambda_1^2 (\mu_1^2 + \cdots \mu_9^2 - 1) = 0$$

$H^T A^T A H - e_1^T A^T A e_1 \geq 0$이기 때문에 $e_1^T A^T A e_1$는 $H^T A^T A H$의 lower bound가 된다. 따라서 $H^* = e_1$이 된다. 즉, 가장 작은 eigenvalue에 대응하는 eigenvector가 solution이 되는 것이다. 이러한 식으로 우리는 4개의 point 쌍이 주어지게 되면 linear system을 유도해서 homography H를 구할 수가 있다.

#5.

(x_c, y_c)를 기준으로 (x, y)를 반시계 방향으로 회전시키켜 (x′,y′)을 얻고자 할 때는 원점을 기준으로 rotation transformation을 한 것과 다른 결과를 가지게 된다. 기준점이 평행이동했다는 관점이 들어가서 rotation과 translation을 함께 진행했다고 봐도 무방하다. 기존의 rotation 관계식은 다음과 같다.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

우리는 원점 (0,0)이 (x_c, y_c)로 이동했다고 생각하고 기존의 좌표를 모두 translation 시켜주고자 한다.

$$\begin{bmatrix} x' - x_c \\ y' - y_c \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \end{bmatrix}$$

이를 전개해서 정리하면 다음과 같다.

$$x' - x_c = \cos\theta(x - x_c) - sin\theta(y - y_c)$$
$$y' - y_c = \sin\theta(x - x_c) + cos\theta(y - y_c)$$

우리는 x′과 y′을 x와 y가 주어졌을 때 구할 수 있는 homography matrix를 찾고자 한다. 그래서 다시 다음과 같이 식을 정리할 수 있다.

$$x' = \cos\theta(x - x_c) - sin\theta(y - y_c) + x_c$$
$$y' = \sin\theta(x - x_c) + cos\theta(y - y_c) + y_c$$

이를 matrix representation으로 나타내고자 2D transformation이지만 3개의 항으로 계산이 이루어져 임의로 차원을 하나 늘리고자 한다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta(x - x_c) - sin\theta(y - y_c) + x_c \\ \sin\theta(x - x_c) + cos\theta(y - y_c) + y_c \\ 1 \end{bmatrix}$$

덧셈 연산을 하나씩 분해하면서 matrix를 decomposition 할 것이다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta(x - x_c) - sin\theta(y - y_c) \\ \sin\theta(x - x_c) + cos\theta(y - y_c) \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -sin\theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -sin\theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

θ가 30도라서 이를 대입하면 다음과 같다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\text{where } \mathrm{R} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix}$$

이렇게 원점이 아닌 좌표로부터 반시계 방향으로 rotation 시키는 homography R을 위와 같이 구할 수가 있게 된다.

#6.

RANSAC을 통해서 먼저 2개의 image를 가장 잘 연결해주는 transition을 찾고자 한다. 2개의 image와 match들이 있을 때 이 중에서 무작위로 하나의 match를 고르고 선택된 match로부터 translation vector를 계산하게 된다. 그러면 알고리즘으로부터 계산이 된 translation vector와 일치하는 inlier의 개수를 센다. 만약 우리가 찾은 translation vector가 좋은 경우에는 최대한 많은 match들이 존재한다는 것이 설명 가능하고 반대로 좋지 않을 경우에는 소수의 match만이 설명 가능해진다. RANSAC은 이러한 과정을 모든 inlier에 대해서 반복하고 가장 많은 inlier들을 집합으로 묶어서 가지게 된다. 총 N번의 반복이 끝난 후에는 가장 많은 inlier가 포함된 집합을 가지고 있을 것이고, 이로부터 알고리즘은 translation vector의 평균을 계산하게 된다. 따라서 최종적으로는 가장 큰 집합을 설명하는 하나의 vector가 생기게 된다. 이렇게 RANSAC을 진행시키기 위해서는 N을 잘 설정해서 몇번 반복할 것인지를 정해야 한다. RANSCA이 잘 동작하려면 N번의 시도 중에서 적어도 한번은 inlier들에서만 data가 뽑혀야 하는데, 이러한 확률은 N을 키우면 키울수록 증가하게 될 것이다. 그렇다고 무한으로 계속 RANSAC을 반복할 수는 없기 때문에 이를 확률적으

로 결정해주게 된다. 그래서 수학적 확률로 가장 좋은 결과를 찾을 수 있도록 N을 설정하게 되고, 이로 인해서 효율적으로 outlier를 제거하는데에 RANSAC을 사용할 수 있을 것이다.

#7.

Opening은 개체의 윤곽선을 부드럽게 하는 등에 사용하기 위해서 먼저 erosion을 수행하고 dilation을 수행하게 된다. Closing은 반대로 dilation을 수행하고 erosion을 수행한다. 따라서 다음과 같이 정의할 수 있다.

Opening: $(A \ominus B) \oplus B$

Closing: $(A \oplus B) \ominus B$

#8.

Binary image에서 작은 크기의 white dot과 같은 noise를 지우기 위해서는 opening을 적용하면 된다. Opening은 erision을 수행하고 dilation을 진행하기 때문에 ersion을 통해서 하얀 noise를 지우고 다시 dilation을 통해서 그 공간을 채울 수가 있다.

#9.

Median filter의 경우 특정 구간에서의 pixel 값들을 작은 값부터 큰 값으로 정렬했을 때, 중간 값을 골라서 output image의 pixel 값으로 결정하게 된다. 3 x 3 median filter를 (2,2)의 위치에 적용하기 위해서는 주변까지 해서 총 9개의 pixel을 보고 이 중에서 중간 값으로 대체하면 된다. 그래서 84, 60, 67, 114, 243, 103, 48, 93, 159를 정렬시키면 48, 60, 67, 84, 93, 103, 114, 159, 243이 되어 중간에 있는 값인 93을 선택해서 filtering 결과 (2,2) 위치에 적용하면 된다.

#10.

Image deblurring의 목적은 blurred image가 주어졌을 때 latent sharp image로 복원하고자 하는 것이다. Image deblurring의 classical한 방법으로는 inverse problem으로 접근하는 방법이 있다. 그러기 위해서는 image degradation model이 필요하며, 이 문제에서 image degradation model은 image blur이다. 그래서 이를 image blur model이라 부르고 다음과 같이 일반적으로 정의해서 사용하곤 한다.

$$B = K * L + N$$

B는 blurred image, K는 blur kernel, L은 latent sharp image, 그리고 N은 noise이다. 이러한 model

이 가장 기본적인 image deblurring model이고 이는 무엇이 주어지느냐에 따라서 blind deconvolution problem과 non-blind deconvolution problem으로 나뉘게 된다. 기본적으로 image deblurring process는 convolution 연산을 기반으로 진행된다.

먼저, blind deconvolution은 blur kernel에 대해서 모르기 때문에 blind라고 부른다. 그래서 우리는 image blur model에서 K, L, N 중 어느 것도 알지 못하는 상태가 된다. 보통 blur는 카메라의 흔들림이에 의해 발생하고 대부분 알지 못하는 상태가 기본적이다. B를 보더라도 실제로는 카메라가 얼마나 흔들렸는지 알 수 없다. 카메라의 흔들림 등의 원인으로 B가 만들어지는데 이를 다시 L로 복원하기 위해서는 blind deconvolution problem을 풀어야 한다.

Non-blind deconvolution은 blur kernel에 대해서 기본적으로 알고 있는 상태이다. 그래서 K의 존재를 알고 L을 복원하고자 한다. 그래서 이 문제에서는 K는 알고 L과 N만 모르는 상태가 된다. 대부분의 경우에는 K를 알수가 있다. 카메라의 흔들림 외에도 L을 카메라로 촬영했을 때 K를 얻었다면 lens에 의해서 생기기도 한다. Lens같은 경우 image의 해상도에 영향을 주고 low-pass filter와 같이 사용이 되어 blur를 만들 수 있기 때문이다. 그래서 대부분의 경우에는 lens에 의한 blur를 없애고 싶어하고 이 경우에는 chart image와 같은 수단을 사용해서 lens에 의한 blur를 예측할 수가 있게 된다. 결국 K를 아는지 모르는지에 따라 blind인지 non-blind인지 나뉘게 되는 것이다.

#11.

$$y = h * x + n$$

우리가 degraded image y와 blur kernel h를 알고 있는 상황에서 원래의 undegraded image x에 blur와 noise를 더하게 되면 이는 non-blind image degradation으로 볼 수 있다. 여기서 x를 찾는데 있어서 적용가능한 방법으로는 inverse filter를 사용하는 inverse problem method가 가능하다. Inverse filtering의 경우 Fourier transform을 위와 같은 image degradation model 적용해서 유도할 수가 있다. 그래서 먼저 inverse filtering을 위한 Fourier transform을 진행시키면 다음과 같이 domain을 바꾸게 된다.

$$\mathrm{Y} = \mathrm{F}^{-1}\{F\{h\}F\{x\}\}$$

여기서 F는 forward Fourier transform function으로 기존의 input image h와 x에 적용한 뒤에 reverse Fourier transform function을 통해서 다시 spatial domain으로 이동시킬 수 있다. 연산도 기존의 convolution 연산에서 이제는 element-wise multiplication 연산으로 대체하게 된다. 이렇게 inverse filtering을 적용하면 다음과 같이 원래의 undegraded image를 찾을 수가 있다.

$$\tilde{\mathrm{x}} = F^{-1}\left(\frac{F\{y\}}{F\{h\}}\right)$$

마찬가지로 blur kernel의 inverse filter도 구할 수가 있다.

$$\tilde{\mathrm{h}} = F^{-1}(\frac{F\{y\}}{F\{x\}})$$

Inverse filtering을 실제로는 잘 사용하지 않는 이유는 바로 만족스러운 결과를 만들지 못하기 때문이다. Blur kernel이 frequency domain에서는 0이나 0에 가까운 값을 가지게 되어 나눗셈 연산 등에서 noise를 증폭시킬 수가 있다.

#12.

이 논문에서는 style tranfer problem을 energy function을 통해서 수식화하여 풀고자 했다.

$$y^* = \underset{y}{\operatorname{argmin}}\{L_{content}(x_c, y) + L_{style}(x_s, y)\}$$

Input으로는 content image와 style image를 사용하고, 이 image들을 기반으로 energy function을 최소로 만드는 y^*을 찾고자하는 것이다. 그래서 energy function은 content loss와 style loss의 합으로 되어 있고, 각 content와 style은 y와의 거리의 차이를 기반으로 정의가 되어 있으며, 이는 꽤 직관적이다. 2개의 항을 최소로 만들어서 x_c와는 동일한 content를 x_s와는 동일한 style을 가지는 y^*을 찾을 수 있다. 이 문제를 풀려면 각각의 representation들이 중요하고 Gram matrix는 style represenation에서 중요하게 사용된다. 주어진 image x에 대해서 style representation은 다음과 같다.

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

G^l은 $N_l \times N_l$ 인 gram matrix로 N_l 은 채널의 크기가 되며, VGG network에서 feature map을 사용해서 적의가 된다. 디테일 하게 l은 l번째 layer의 index이고, F_{ik}^l은 l번째 layer에서의 feature map F^l로부터 i번째 채널에서 k번째 위치에 있는 feature의 값을 의미한다. 각 채널은 feature 값의 2차원 array이고, Gram matrix는 i, j 채널의 F^l의 dot product로 정의가 된다. 그래서 서로 다른 채널들은 서로 다른 local structure를 포착하는데 이때 하나의 local structure가 다른 structure랑 항상 같이 나오는 패턴을 style로 정의할 수 있다. Gram matrix는 이러한 local structure들 사이의 correlation을 계산할 수 있다. Style은 결국 feature들이 여러가지 형태로 존재한다고 했을 때 통계적으로 가장 두드러지게 나타나는 패턴을 이야기하고, Gram matrix는 이러한 feature들로부터 통계적인 부분을 포착할 수 있는 방법 중 하나이다. 서로 다른 feature들 사이의 correlation을 계산해서 상대적으로 빈번하게 나타나는 것을 채택하는 것이다.

#13.

DCGAN에서 latent code가 arithmetic한 성질을 가지고 있어서 이를 통해서 vector를 계산에 사용할 수 있었고, 이는 latent code를 조작함으로써 GAN에 의해서 만들어진 image에 의미를 부여할 수 있다. 이는 latent code에만 적용이 되는 부분이라서 GAN에 의해 만들어진 결과에만 적용할

수 있고, 현실 세계의 image는 latent vector가 아니라서 현실 세계의 image에 적용하기 위해서는 GAN inversion이라는 것이 필요한 것이다. GAN inversion은 target image에 대응되는 latent random vector를 찾는 문제이다. GAN에서 generator는 latent random vector로부터 fake image를 만들어낸다. GAN inversion은 이 과정을 반대로 생각해서 target image로부터 latent random vector를 찾아내게 된다. 이렇게 적절한 latent code를 찾아내게 되면 이를 이용해서 GAN model을 통해 현실 세계의 image를 생성해낼 수 있다.