



# 렌.고.쿠 는 어디에 숨었나

## □ 목 적

드라마, 영화 등에서 종종 볼 수 있는 이미지 O.L (overlap) 모방 중에 발생한 에러와 발생 원인 및 해결 과정을 공유

\* 활용 수단 수학적 모델: 행렬, Linearity, 부등식  
공학적 도구: Google Colab, Python (3.10.12), PIL, numpy, IPython 등  
이미지 파일: 애니메이션 캐릭터 두 장

## □ 모 델 링

### ○ 행렬, Linearity 활용 이미지 합성 및 분해 가능 여부 검토

#### ✓ Superposition Principle

어떤 행렬  $A_{m \times n}$ ,  $B_{k \times l}$  가  $m = k$ ,  $n = l$  인 같은 꼴일 때, 행렬의 덧셈이 정의되고 두 행렬  $A$ ,  $B$  의  $(i, j)$  성분의 합은 두 행렬의 합  $A + B$  의  $(i, j)$  성분과 같다.

$$A_{ij} + B_{ij} = (A + B)_{ij} \text{ (단, } 1 \leq i \leq m, 1 \leq j \leq n \text{)}$$

따라서, 다음 등식이 성립한다.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} + \begin{pmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix} = \begin{pmatrix} (1+10) & (2+11) & (3+12) \\ (4+13) & (5+14) & (6+15) \\ (7+16) & (8+17) & (9+18) \end{pmatrix}$$

#### ✓ Homogeneity

또한, 행렬의 실수배에 관한 정의에 따라 어떤 행렬  $A$  의  $t$  배는 행렬  $A$  의 모든 성분에  $t$  를 곱한 성분을 새로운 성분으로 하는 행렬과 같다.

$$(tA)_{ij} = t(A_{ij})$$

따라서, 다음 등식이 성립한다.

$$t \times \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} (t \times 1) & (t \times 2) & (t \times 3) \\ (t \times 4) & (t \times 5) & (t \times 6) \\ (t \times 7) & (t \times 8) & (t \times 9) \end{pmatrix}$$



## □ 모델링

### ○ 행렬, Linearity 활용 이미지 합성 및 분해 가능 여부 검토 (계속)

행렬이 Superposition Principle 과 Homogeneity 를 모두 만족하여 Linearity 를 가지므로 같은 풀인 임의의 두 행렬  $A, B$  와 임의의 실수  $t$  에 대하여 다음 등식이 성립한다.

$$t(A + B) = tA + tB \Leftrightarrow t(A + B)_{ij} = t(A)_{ij} + t(B)_{ij}$$

임의의 두 행렬의 합이 이루는 각 성분은 합을 이루는 각 행렬의 성분에 따라 변하지만 Linearity 를 가지므로 합의 각 성분에는 그 성분을 이루는 두 행렬의 성분이 독립적으로 보존되며, 적절한 방법을 쓰면 다시 두 행렬의 각 성분으로 분해가 가능하고 이러한 성질은 실수배를 한 임의의 두 행렬의 합에도 동일하게 성립한다.

만약 임의의 두 행렬  $A, B$  가 서로 다른 이미지를 나타내는 행렬이라면 이들의 합으로 이루어진 새로운 행렬  $C$  를 만들 수 있고 다시 두 이미지로 분해도 가능할 것이다. 또한 어떤 이미지 행렬의 실수배를 통해 그 이미지의 색상을 자유롭게 조절할 수 있을 것이다.

### ○ 부등식 활용 합성 이미지 의미 해석

0 이상의 정수  $a$  와 1보다 큰 양수  $b$  의 곱  $ab$  에 대해 다음 부등식이 성립한다.

$$a \leq ba$$

여기에서  $a$  대신 어떤 이미지 행렬  $A$  를 생각할 때,  $bA$  의 임의의 성분  $(bA)_{ij}$  은 항상 0 이상의 정수이고 이에 대응하는  $A$  의 성분  $A_{ij}$  보다 항상 크거나 같다. 따라서 다음 부등식이 성립한다.

$$A_{ij} \leq (bA)_{ij}$$

따라서, 모든 성분이  $[0, 255]$  에 속하는 정수를 갖는 어떤 이미지 행렬  $A$  와 1보다 큰 양수  $b$  의 곱  $bA$  의 각 성분은 대응하는  $A$  의 성분과 같거나 큰 성분이므로 행렬  $bA$  는 행렬  $A$  가 나타내는 이미지가 갖는 색보다 더 짙은 색을 갖는 이미지를 나타내는 행렬이다. 하지만 O.L 은 중첩된 두 이미지 중 하나는 본래 이미지보다 옅게, 나머지 하나는 옅은 상태에서 본래 이미지로의 변화 과정이므로 1보다 큰 양수  $b$  를 이미지 행렬에 곱하는 것은 O.L 현상을 나타내는 수학적 표현이라 볼 수 없다.

이번에는 닫힌 구간  $[0, 1]$  에서 정의된 임의의 실수  $t$  를 생각해 보자. 이때  $1 - t$  와  $t$  의 부등식을 대조하여 극한을 관찰하면 다음과 같다.

$$\begin{cases} 0 \leq t \leq 1 \\ 0 \leq 1-t \leq 1 \end{cases}$$

$$\begin{aligned} t \rightarrow 1- & \Rightarrow 1-t \rightarrow 0+ \\ t \rightarrow 0+ & \Rightarrow 1-t \rightarrow 1- \end{aligned}$$



## □ 모델링

### ○ 부등식 활용 합성 이미지 의미 해석 (계속)

위의 내용을 임의의 두 행렬의 각 성분에 적용하여 살펴보면 다음과 같다.

$$\begin{cases} 0 \leq t(A)_{ij} \leq A_{ij} \\ 0 \leq ((1-t)B)_{ij} \leq B_{ij} \end{cases}$$

$$\begin{aligned} t &\rightarrow 1- \Rightarrow (1-t) &\rightarrow 0+ \\ (tA)_{ij} &\rightarrow A_{ij}- \Rightarrow ((1-t)B)_{ij} &\rightarrow 0+ \end{aligned}$$

$$\begin{aligned} t &\rightarrow 0+ \Rightarrow (1-t) &\rightarrow 1- \\ (tA)_{ij} &\rightarrow 0+ \Rightarrow ((1-t)B)_{ij} &\rightarrow B_{ij}- \end{aligned}$$

이를 해석하면 다음과 같은 사실을 확인할 수 있다.

$t \rightarrow 1-$  일 때, 이미지 행렬  $tA$  가 나타내는 이미지는  
색이 짙어지며 이미지  $A$  에 근접하고,  
이때  $1-t \rightarrow 0+$  이므로 이미지 행렬  $(1-t)B$  가 나타내는 이미지는  
색이 옅어지며  $O$  (투명한 상태)에 근접한다.

또한  $t \rightarrow 0+$  일 때, 이미지 행렬  $tA$  가 나타내는 이미지는  
색이 옅어지며  $O$  (투명한 상태)에 근접하고,  
이때  $1-t \rightarrow 1-$  이므로 이미지 행렬  $(1-t)B$  가 나타내는 이미지는  
색이 짙어지며 이미지  $B$  에 근접한다.

이제 이미지 행렬  $tA$  와  $(1-t)B$  의 합성 행렬  $C$  에 대해 이 행렬의 임의의 성분  $C_{ij}$  의 성질을 살펴보자. 그렇다면 다음이 성립함을 알 수 있다.

$$\begin{cases} 0 \leq t \leq 1, A_{ij} \in \mathbb{Z}, A_{ij} \in [0, 255] \\ 0 \leq (1-t) \leq 1, B_{ij} \in \mathbb{Z}, B_{ij} \in [0, 255] \end{cases} \Rightarrow \begin{cases} 0 \leq (tA)_{ij} \leq 255 \\ 0 \leq ((1-t)B)_{ij} \leq 255 \end{cases}$$

$$\begin{cases} t = 1 \text{ 일 때, } (tA)_{ij} \text{ 는 최댓값 } 255, ((1-t)B)_{ij} \text{ 는 최솟값 } 0 \\ t = 0 \text{ 일 때, } (tA)_{ij} \text{ 는 최솟값 } 0, ((1-t)B)_{ij} \text{ 는 최댓값 } 255 \end{cases} \Rightarrow 0 \leq C_{ij} \leq 255$$

그리고  $A_{ij}, B_{ij} \in \mathbb{Z}$  이고  $\mathbb{Z}$  는 덧셈에 대해 닫혀있으므로  $C_{ij} \in \mathbb{Z}$  이다. 따라서,  
 $C_{ij} \in [0, 255], \mathbb{Z}$  이므로 합성 행렬  $C$  는 이미지를 나타내는 행렬이다.



## □ 모델링

### ○ 부등식 활용 합성 이미지 의미 해석 (계속)

합성 행렬  $C$ 는 이미지를 나타내는 행렬이다. 이 행렬은 **Linearity**를 가지므로 각 성분은 이미지  $A$ 와  $B$ 의 성분을 온전히 가지고 있고 적절한 방법을 통해 이미지 행렬  $A$ 와  $B$ 의 성분을 분해할 수 있으며  $C$ 의 경우,  $t$ 의 변화가 그 목표를 이루는 실마리가 된다.

즉,

$$\begin{aligned} t \rightarrow 1- &\Rightarrow (1-t) \rightarrow 0+ \text{ 일 때 } C \text{는 } A \text{에 근접하고,} \\ t \rightarrow 0+ &\Rightarrow (1-t) \rightarrow 1- \text{ 일 때 } C \text{는 } B \text{에 근접한다.} \end{aligned}$$

따라서, 이미지 행렬  $C$ 의  $t$ 에 변화를 주면 두 이미지의 O.L을 표현할 수 있을 것이다.

## 결론

O.L 모델링 결과 도출된 모델은 아래와 같고, 모델 구현 전에 시뮬레이션을 통해 O.L 모방에 적합한지 확인할 필요가 있다.



## O.L 모델

- $t \in [0, 1]$  이고 실수인  $t$ 가 존재한다.
- 행렬은 서로 같은 꼴이다.
- 행렬의 각 성분은  $\{x \mid x \in [0, 255], x \in \mathbb{Z}\}$ 에 속한다.
- 임의의 두 이미지 행렬  $A, B$ 의 합성 행렬  $C$ 는 아래와 같다.

$$C = tA + (1-t)B$$

## □ 시뮬레이션 : GeoGebra Classic 6 활용 ➡ 모델 적합

## □ 모델 구현 : 기본 모델 ⇨ 반응형 모델



## □ 모델 구현(계속)

### ○ 기본 모델

1. 모듈, 패키지, 라이브러리 준비 ⇨ 이미지 준비 ⇨ 변수 구성
2. 이미지 행렬 변환과 Resizing ⇨ 행렬 합성 및 dtype 변환  
⇨ 가중치에 따른 합성 이미지 표시 기능 구성
3. 테스트 및 코드 개선

#### • Step 1: 준비 단계

- 모듈, 패키지, 라이브러리 준비

```
import numpy as np # Colab 환경
from PIL import Image
from time import sleep
from google.colab import files
from IPython.display import display
```

- 이미지 준비

렌고쿠



탄지로



< 출처 > 렌고쿠 <https://m.blog.naver.com/mgc4007/222253482149>  
탄지로 [https://m.blog.naver.com/grace\\_seonmi/221982737601](https://m.blog.naver.com/grace_seonmi/221982737601)



```
#@markdown **이미지를선택하여 업로드해 주세요.**
```

```
# 이미지 파일 업로드
files.upload().keys()
```

- 변수 구성

```
t = 0          # 가중치
rengoku_img = Image.open('/content/렌고쿠.jpg')
tanjiro_img = Image.open('/content/탄지로.jpg')
```

## • Step 2: 구성 단계

- 이미지 행렬 변환과 Resizing

```
# rengoku, tanjuro 이미지 행렬 변환
rengoku_arr = np.array(rengoku_img)
tanjiro_arr = np.array(tanjiro_img)

# rengoku 행렬을 tanjiro 행렬과 같은 꼴로
rengoku_arr = np.array(rengoku_img.crop((0, 0, 468, 703)))
```

- 행렬 합성 및 dtype 변환

```
# 행렬 합성
composite_img_arr = (t * rengoku_arr + \
                    (1- t) * tanjiro_arr)

# dtype 변환
composite_img_arr = composite_img_arr.astype(np.uint8)
```

- 가중치에 따른 합성 이미지 표시 기능 구성

```
# 가중치에 따른 합성 이미지 출력 기능
for i in range(11):
    t += 0.1
    display(composite_img_arr)
    sleep(1)
```



· Step 3 : 테스트 및 코드 개선

- 모듈 테스트: 코드 작성 진행하며 확인 ☞ 오류 발생 없음
- 모델 테스트: 모델이 예상 밖의 결과 출력

가중치가 변해도 합성 이미지가 변하지 않는 현상

Weight: 0.1



Weight: 0.2



Weight: 0.3



Weight: 0.4



Weight: 0.5



Weight: 0.6



Weight: 0.7



Weight: 0.8



Weight: 0.9

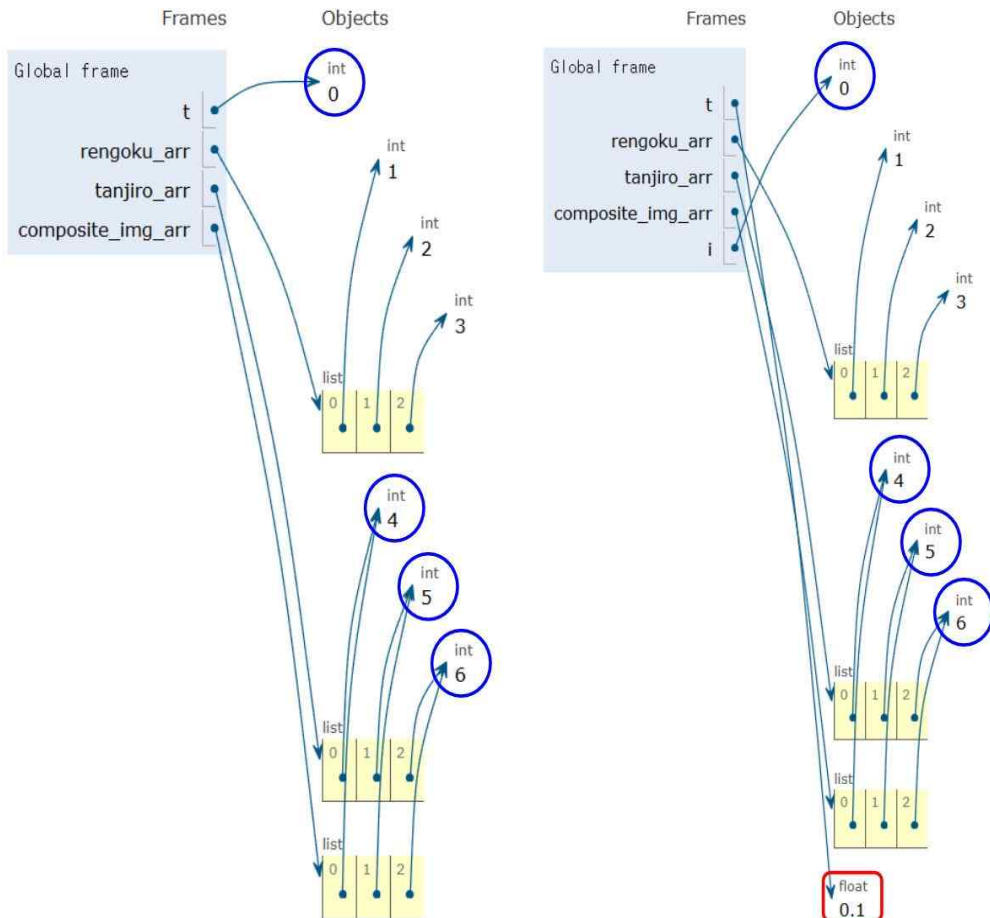




## ✓ 원인 분석

for 진입 전 합성 이미지

for 진입 후 합성 이미지



## 분석 결과

`composite_img_arr` 는 for 에 진입 후 변하는 가중치 `t` 가 가리키는 객체를 참조하지 않는다.

따라서, 변하는 가중치 `t` 가 가리키는 객체를 참조하도록 `t` 의 변화에 따라 `composite_img_arr` 를 재정의하고 표현 위치를 조정할 필요가 있다.



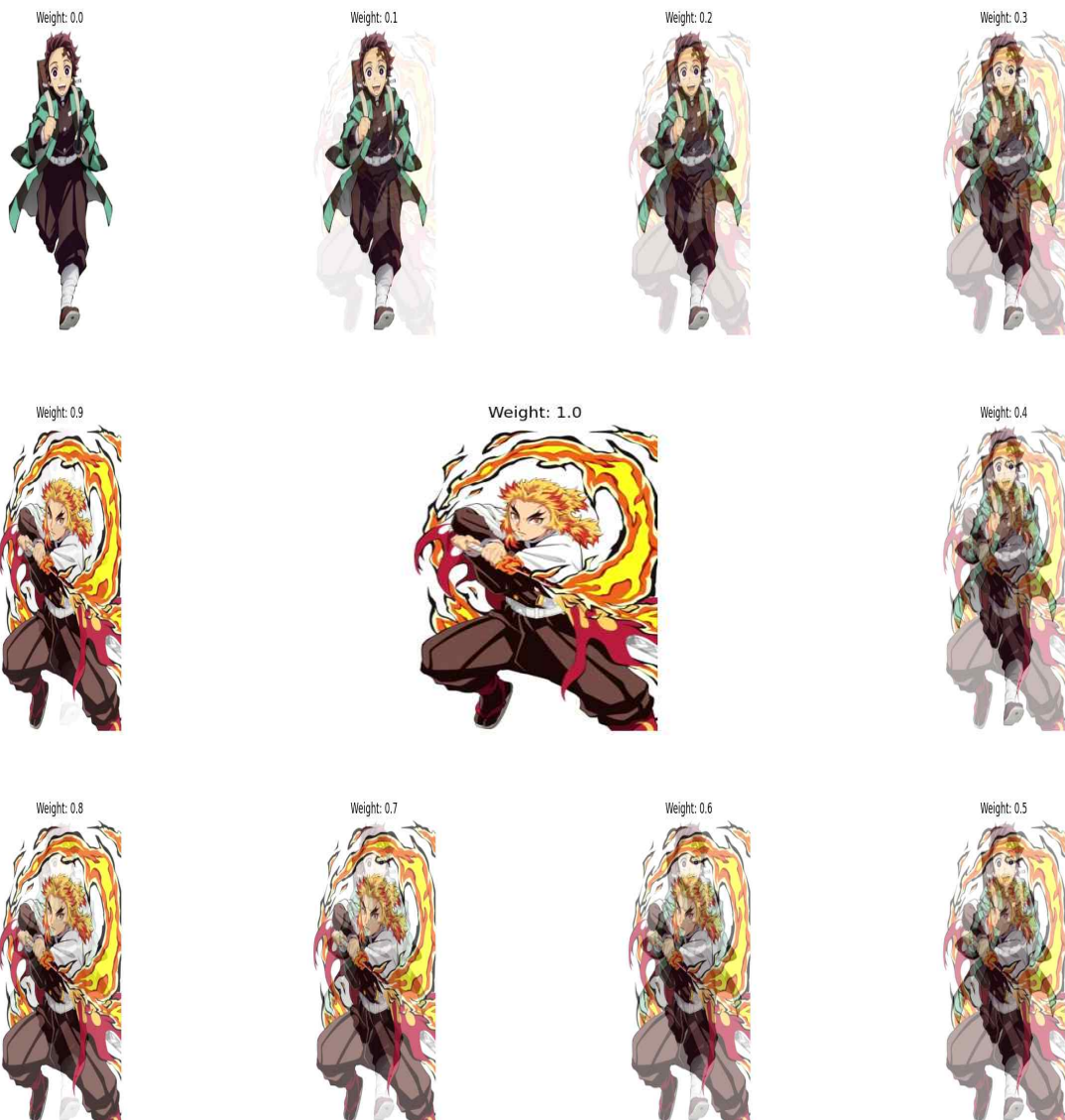


## · Step 3: 테스트 및 코드 개선(계속)

### - 코드 개선

```
# 가중치에 따른 합성 이미지 표시 기능
for i in range(11):
    composite_img_arr = (t * rengoku_arr + \
                        (1- t) * tanjiro_arr).astype(np.uint8)
    display(composite_img_arr)
    t += 0.1
    sleep(1)
```

### - 코드 개선 후 모델 테스트





## ○ 반응형 모델

### 『기본 모델』을 바탕으로

1. 모듈, 패키지, 라이브러리 준비
2. 이미지 행렬 변환과 Resizing ⇨ 행렬 dtype 변환  
⇨ 애니메이션 프레임 목록 생성 ⇨ 애니메이션 표시 기능 구성
3. 테스트 및 코드 개선

#### • Step 1: 준비 단계

- 모듈, 패키지, 라이브러리 준비

```
from cv2 import resize
from IPython.display import HTML
from tqdm import tqdm_notebook
import matplotlib.animation as animation
```

#### • Step 2: 구성 단계

- 이미지 행렬 변환과 Resizing

```
# rengoku, tanjuro 이미지 행렬 변환 Colab 환경
rengoku_arr = np.array(rengoku_img)
tanjiro_arr = np.array(tanjiro_img)

# rengoku 행렬을 tanjiro 행렬과 같은 꼴로
row, col, _ = tanjiro_arr.shape
rengoku_arr = resize(rengoku_arr, dsize = (col, row))
# np.array(rengoku_img.crop((0, 0, 468, 703)))
```

- 행렬 dtype 변환

```
# rengoku, tanjiro 이미지 행렬 dtype 변환
rengoku_arr = rengoku_arr / 255.0
tanjiro_arr = tanjiro_arr / 255.0
```



## · Step 2: 구성 단계 (계속)

- 애니메이션 프레임 목록 생성

```
fig = plt.figure()          # 캔버스 개방
plt.axis('off')             # 축 제거

# 가중치에 따른 이미지 합성, 프레임 생성 및 프레임 목록 갱신
for i in tqdm_notebook(range(NUM_OF_FRAMES + 1)):

    # 가중치 설정
    alpha = i / NUM_OF_FRAMES

    # 이미지 합성
    composite_image_arr = (alpha * rengoku_arr + \
                           (1 - alpha) * tanjiro_arr)

    # 이미지 프레임 생성
    img_frame = [plt.imshow(composite_image_arr, \
                             animated = True)]

    # 프레임 목록에 갱신
    img_frames.append(img_frame)

plt.close()                 # 캔버스 폐쇄

# Overlap 애니메이션 생성
overlap_animation = animation.ArtistAnimation(fig, \
                                              img_frames, interval = 40, blit = True)

# 애니메이션을 jupyter notebook 환경에서 표시
display(HTML(overlap_animation.to_jshtml()))
```



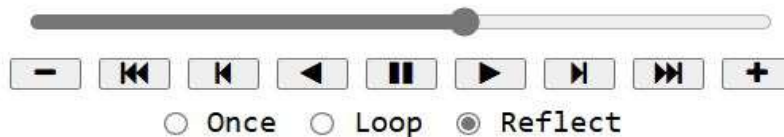
· Step 3: 테스트 및 코드 개선

- 모듈 테스트: 코드 작성 진행 간 모듈 동작 확인하며 오류 수정
- 모델 테스트: 프레임 수 조정 (50 ⇨ 100), interval 조정 (10 ⇨ 40)
- 코드 개선

```
NUM_OF_FRAMES = 100      # 프레임 수

# Overlap 애니메이션 생성
overlap_animation = animation.ArtistAnimation(fig,\
                                             img_frames, interval = 40, blit = True)
```

- 코드 개선 후 모델 테스트



“렌고쿠는 숨지 않았다.”

끝.