ML/AI 개발자를 위한 단계별 Python 최적화

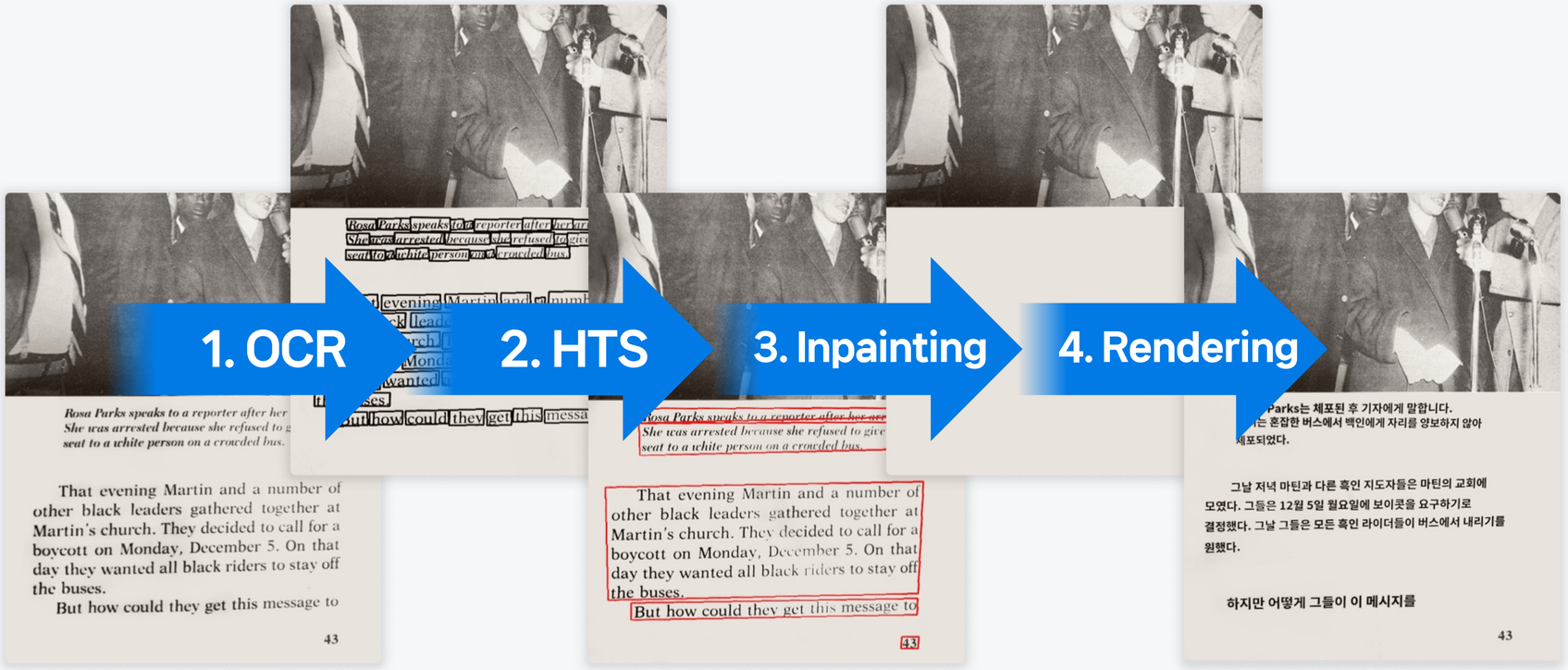
문주혁
NAVER Cloud / Papago

NAVER DEVIEW 2023

0. 배경 및 문제

0.1 파파고 이미지 번역

NAVER
DEVIEW
2023



Deview 2019: 문자인식(OCR), 얼마나 정확하지? (문자인식 성능을 정확하게 측정하는 방법

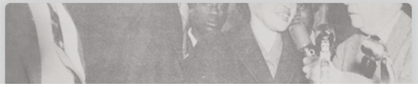
Deview 2020: 외국어가 읽힌다 딱! (파파고 이미지 번역

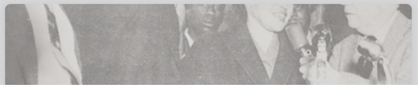
Deview 2021: 더 감쪽같은 이미지 번역을 위해 (파파고 이미지/AR 번역 개발기

Deview 2023: '더' 잘 읽히고 자연스러운 이미지 번역을 위해(파파고 텍스트 렌더링 개발기

0.1 파파고 이미지 번역

NAVER
DEVIEW
2023



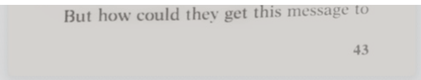


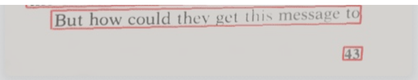
NAVER
CLOUD PLATFORM

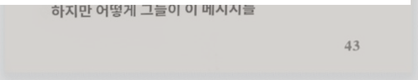
Papago Image Translation

이미지 상의 텍스트를 자동으로 인식하고 여러 나라의 언어로 번역합니다.

https://www.ncloud.com/product/aiService/papagoImageTranslation



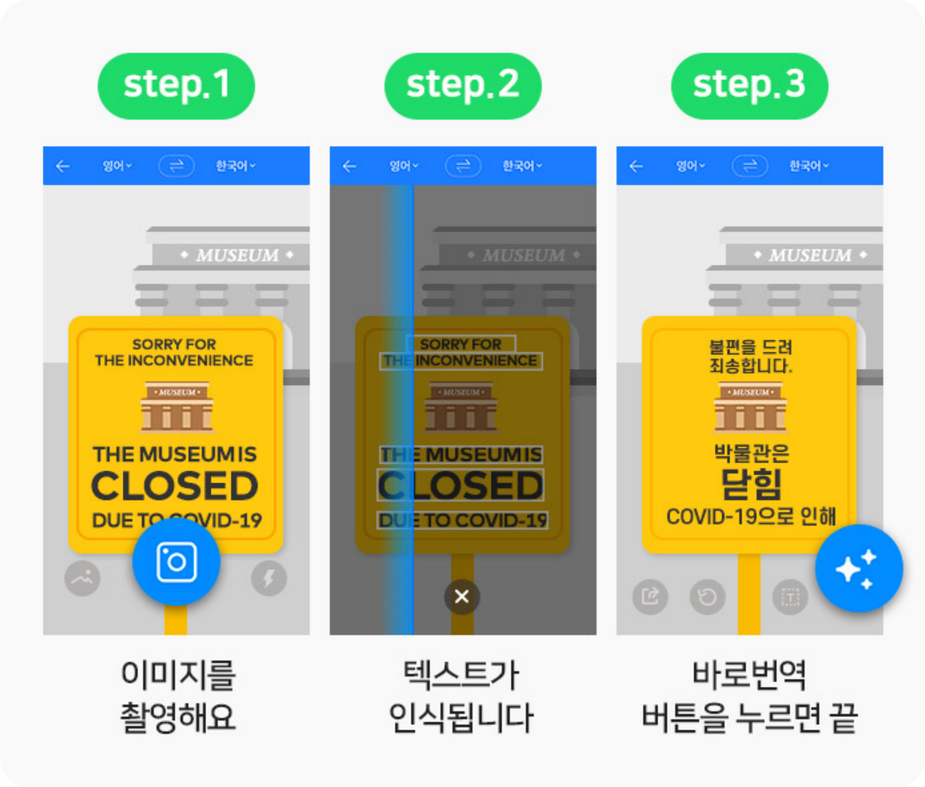




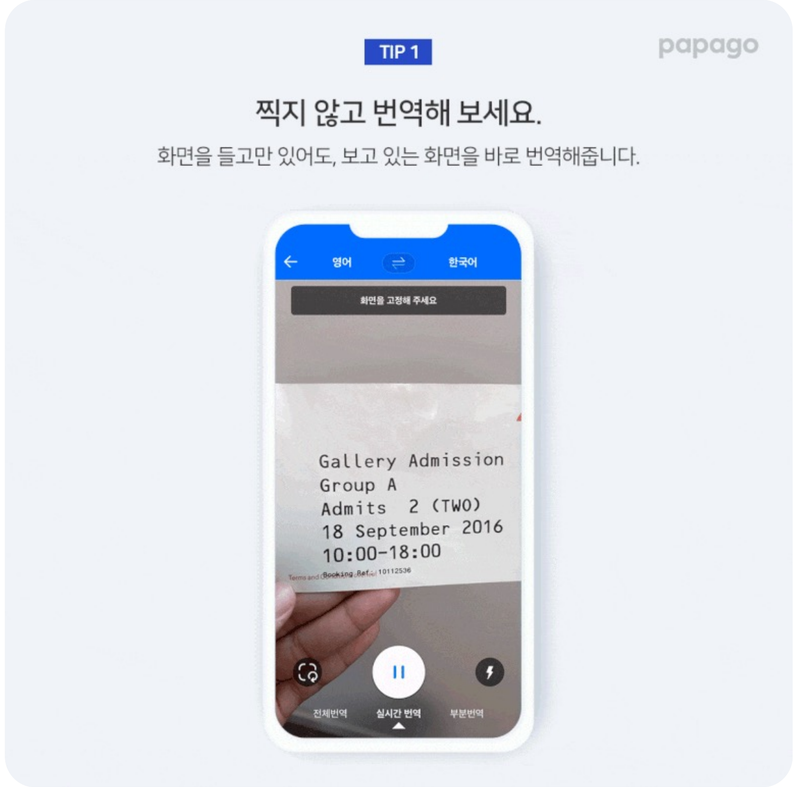
Deview 2019: 문자인식(OCR), 얼마나 정확하지? (문자인식 성능을 정확하게 측정하는 방법
Deview 2020: 외국어가 읽힌다 딱! (파파고 이미지 번역
Deview 2021: 더 감쪽같은 이미지 번역을 위해 (파파고 이미지/AR 번역 개발기
Deview 2023: '더' 잘 읽히고 자연스러운 이미지 번역을 위해(파파고 텍스트 렌더링 개발기

0.1 파파고 이미지 번역

NAVER
DEVIEW
2023



이미지 바로 번역

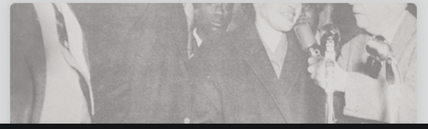


실시간 번역

Deview 2019: 듀
Deview 2020: 외국어가 읽힌다 짝! (파파고 이미지 번역
Deview 2021: 더 감쪽같은 이미지 번역을 위해 (파파고 이미지/AR 번역 개발기
Deview 2023: '더' 잘 읽히고 자연스러운 이미지 번역을 위해(파파고 텍스트 렌더링 개발기

0.1 파파고 이미지 번역

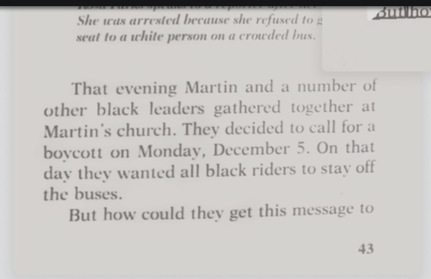
NAVER
DEVIEW
2023

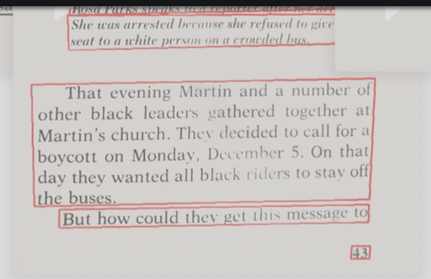


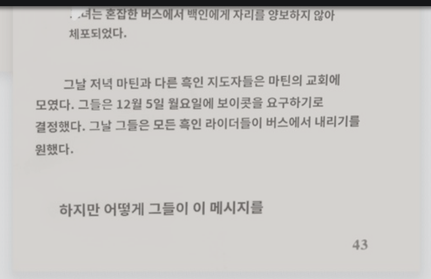
서버 환경: X86_64 (+ CUDA)

개발자: Computer Vision 전공

사용 언어: Python + OpenCV + NumPy (+ PyTorch)







Deview 2019: 문자인식(OCR), 얼마나 정확하지? (문자인식 성능을 정확하게 측정하는 방법

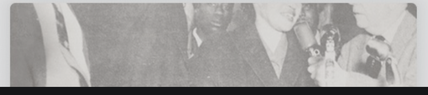
Deview 2020: 외국어가 읽힌다 딱! (파파고 이미지 번역

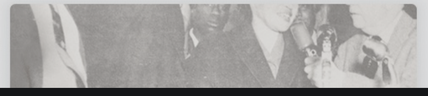
Deview 2021: 더 감쪽같은 이미지 번역을 위해 (파파고 이미지/AR 번역 개발기

Deview 2023: '더' 잘 읽히고 자연스러운 이미지 번역을 위해(파파고 텍스트 렌더링 개발기

0.1 파파고 이미지 번역

NAVER
DEVIEW
2023





한 장의 이미지를 번역하기 위해

입력 형태가 각기 다른 여러 ML/AI 모델

+

ML로 대체되지 못한 렌더링

That evening Martin and a number of other black leaders gathered together at Martin's church. They decided to call for a boycott on Monday, December 5. On that day they wanted all black riders to stay off the buses.

But how could they get this message to

43

That evening Martin and a number of other black leaders gathered together at Martin's church. They decided to call for a boycott on Monday, December 5. On that day they wanted all black riders to stay off the buses.

But how could they get this message to

43

그날 저녁 마틴과 다른 흑인 지도자들은 마틴의 교회에 모였다. 그들은 12월 5일 월요일에 보이콧을 요구하기로 결정했다. 그날 그들은 모든 흑인 라이더들이 버스에서 내리기를 원했다.

하지만 어떻게 그들이 이 메시지를

43

Deview 2019: 문자인식(OCR), 얼마나 정확하지? (문자인식 성능을 정확하게 측정하는 방법

Deview 2020: 외국어가 읽힌다 딱! (파파고 이미지 번역

Deview 2021: 더 감쪽같은 이미지 번역을 위해 (파파고 이미지/AR 번역 개발기

Deview 2023: '더' 잘 읽히고 자연스러운 이미지 번역을 위해(파파고 텍스트 렌더링 개발기

NOT INCLUDED

1. GPU 프로파일링

2. ML 모델 최적화

3. Quantization & Transformation 최적화

4. 네트워크 통신 최적화

CONTENTS

1. Python 프로파일링: line profiler 활용하기
2. Python Level 최적화: NumPy, OpenCV 더 빠르게 쓸 수 없는가
3. Semi-C Level 최적화: Cython, Numba 언제 유용한가
4. C/C++ Level 최적화: Python/C API 어디까지 가능한가

+. 코드 외적인 요소 최적화

* 참고 *
모든 예제는 실제 파파고와는 무관한 예제 샘플 코드입니다.

문제 요약

GPU 최적화도 중요하지만,

ML/AI 개발자가 직접 해결해야하는

CPU 최적화 요소가 많이 있다.

1. Python 프로파일링

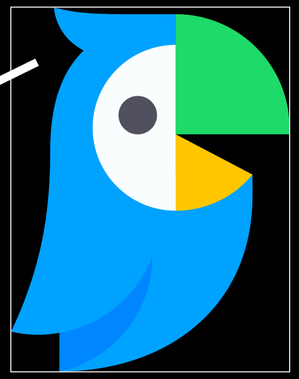
1.1 어디가 가장 오래 걸리나요?

NAVER
DEVIEW
2023

예제. OpenCV/NumPy를 이용한 메인 색상 추출: 이미지 준비

```python
import cv2
import numpy as np

def extract_main_colors_naive(image_path, N=6):
    image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
```

alpha값에 의해 렌더링이 안될 뿐
주로 0 혹은 255 값으로 채워짐



Papago.png

(1254, 958, 4)

color_source

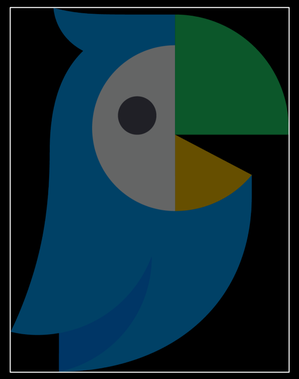
(788348, 3)

1.1 어디가 가장 오래 걸리나요?

NAVER
DEVIEW
2023

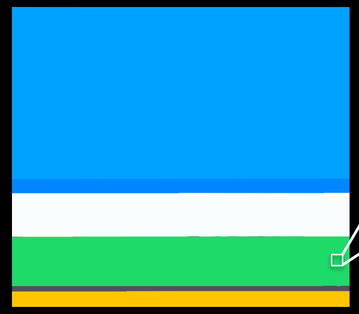
예제. OpenCV/NumPy를 이용한 메인 색상 추출: 픽셀 카운팅

```
packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
```



Papago.png

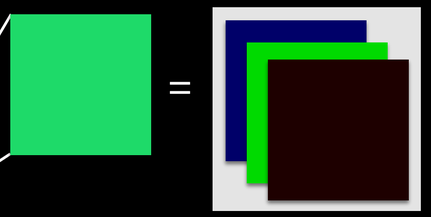
(1254, 958, 4)



color_source

(788348, 3)

(105, 218, 30) → 105218030



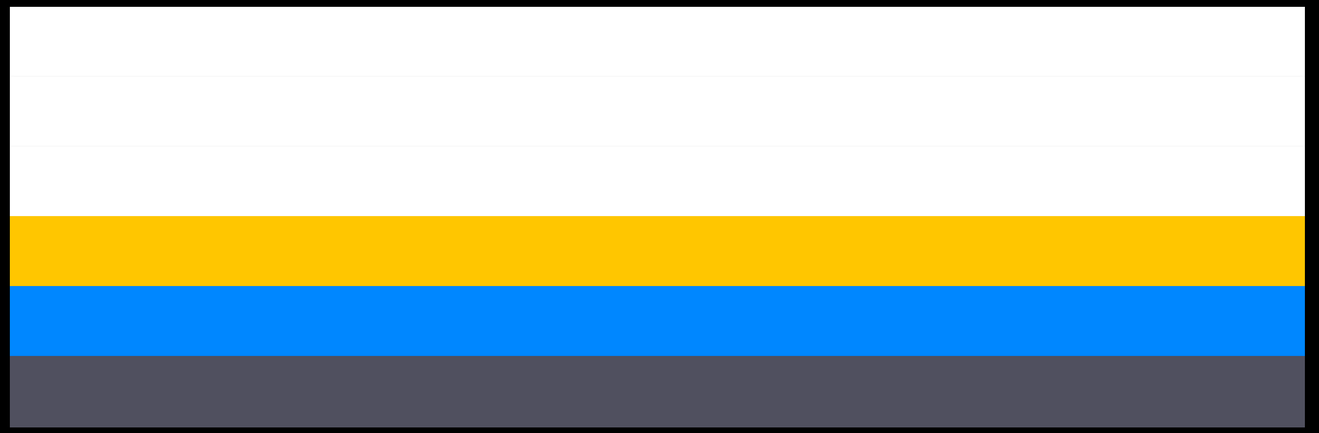
{105: 1, 218: 1, 30: 1, ...} → {105218030: 1, ...}

1.1 어디가 가장 오래 걸리나요?

NAVER
DEVIEW
2023

예제. OpenCV/NumPy를 이용한 메인 색상 추출: 패널 생성

```
panel_h = 64
panel_w = 512
color_panels = []
for color in top_N_colors:
    b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
    color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
    color_panels.append(color_panel)
color_panels = np.concatenate(color_panels, axis=0)
```



1.1 어디가 가장 오래 걸리나요?

NAVER
DEVIEW
2023

문제. 어느 라인이 가장 오래 걸릴까요?

```python
def extract_main_colors_naive(image_path, N=6):
    panel_h = 64
    panel_w = 512

    image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)

    packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
    unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]

    color_panels = []
    for color in top_N_colors:
        b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
        color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)

        color_panels.append(color_panel)
    color_panels = np.concatenate(color_panels, axis=0)

    return color_panels.astype(np.uint8)
```

이미지 로딩

투명 픽셀 제거

픽셀 카운팅

패널 생성

측정 해봐야 알겠죠..

1.1 어디가 가장 오래 걸리나요?

NAVER
DEVIEW
2023

우리의 프로파일링

```
start_t = time.time()
image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
print("imread: ", time.time()-start_t)

start_t = time.time()
color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
print("indexing: ", time.time()-start_t)

start_t = time.time()
packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
print("counting: ", time.time()-start_t)
```

이미지 로딩

투명 픽셀 제거

픽셀 카운팅

투명 픽셀 제거

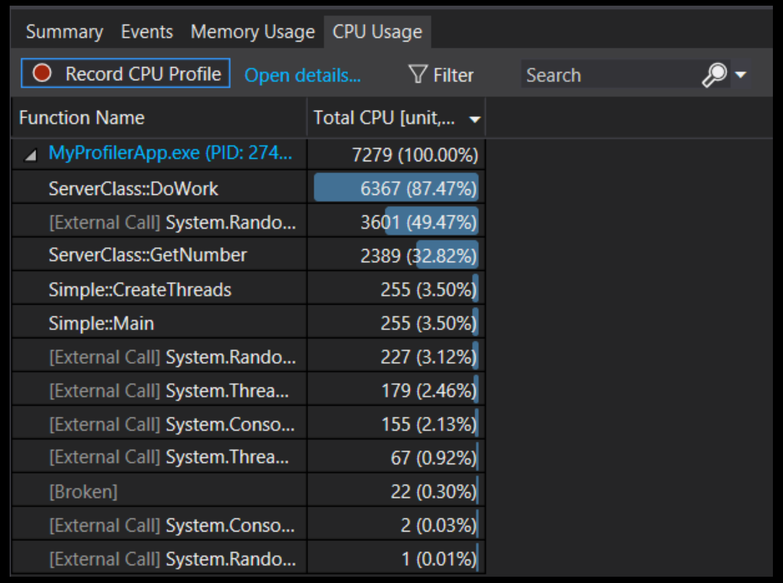
픽셀 카운팅

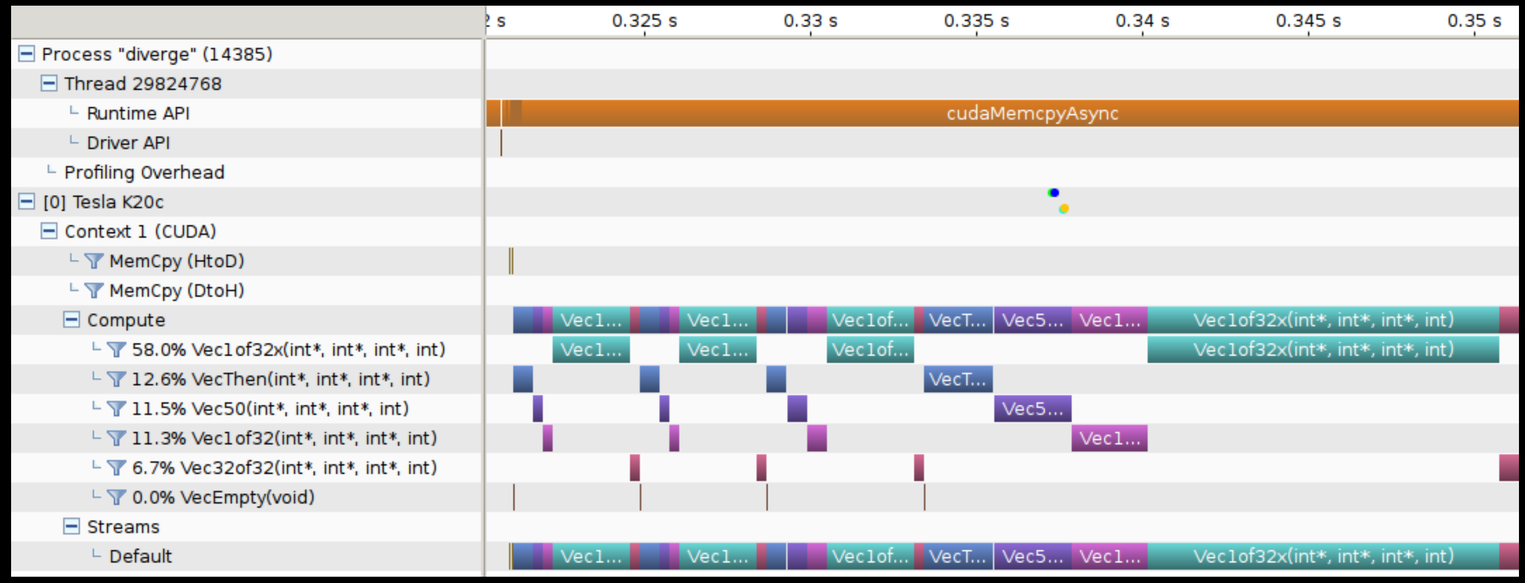
> 한 10번 하다보면 귀찮아서 안함

1.2 프로파일링 진입 장벽

NAVER
DEVIEW
2023

프로파일링 하면 떠오르는 이미지





> 설치부터 어려운 IDE/Profiler

> Low-level function/instruction을 분석해줘도 그게 어디서 발생한건지 잘 안보임

1.3 Line Profiler

NAVER
DEVIEW
2023

- Line-by-line profiling for Python
- 설치: pip install line_profiler
- Linux (x86_64, i686)
- OSX (x86_64) / Apple Silicon (arm)의 경우 Rosetta 필요
- Windows (x86, x86_64)

1.3 Line Profiler

NAVER
DEVIEW
2023

타겟 함수 def 위에 @profile

```python
import cv2
import numpy as np

@profile
def extract_main_colors(image_path, N=6):
    panel_h = 64
    panel_w = 512

    image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)

    packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
    unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]

    color_panels = []
    for color in top_N_colors:
        b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
        color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)

        color_panels.append(color_panel)
    color_panels = np.concatenate(color_panels, axis=0)

    return color_panels.astype(np.uint8)
```

1.3 Line Profiler (terminal)

NAVER DEVIEW 2023

python my_code.py 대신

실행: kernprof –l –v my_code.py

```
> kernprof -l -v A_line_profiler.py
Wrote profile results to A_line_profiler.py.lprof
Timer unit: 1e-06 s

Total time: 0.0608794 s
File: A_line_profiler.py
Function: extract_main_colors at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           @profile
     5                                           def extract_main_colors(image_path, N=6):
     6         1          1.9      1.9      0.0      panel_h = 64
     7         1          0.3      0.3      0.0      panel_w = 512
     8
     9         1      16680.1  16680.1     27.4      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    10         1      22492.2  22492.2     36.9      color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    11
    12         1       4926.6   4926.6      8.1      packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
    13         1      12041.2  12041.2     19.8      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    14         1         20.1     20.1      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    15
    16         1          0.1      0.1      0.0      color_panels = []
    17         6          2.5      0.4      0.0      for color in top_N_colors:
    18         6         10.1      1.7      0.0          b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
    19         6       3078.8    513.1      5.1          color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
    20
    21         6          3.6      0.6      0.0          color_panels.append(color_panel)
    22         1       1465.6   1465.6      2.4      color_panels = np.concatenate(color_panels, axis=0)
    23
    24         1        156.1    156.1      0.3      return color_panels.astype(np.uint8)
```

이미지 로딩

투명 픽셀 제거

픽셀 카운팅

패널 생성

1.3 Line Profiler (jupyter notebook)

NAVER DEVIEW 2023

프로파일러 로딩

프로파일러 실행

```
[5]: %load_ext line_profiler

[6]: %lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
File:
Function: extract_main_colors_naive at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           def extract_main_colors_naive(image_path, N=6):
     5       100         77.1      0.8      0.0      panel_h = 64
     6       100         14.6      0.1      0.0      panel_w = 512
     7
     8       100    1539512.1  15395.1     24.9      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
     9       100    2849188.7  28491.9     46.0      color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    10
    11       100     442498.6   4425.0      7.1      packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
    12       100    1110719.8  11107.2     17.9      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    13       100       1329.0     13.3      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    14
    15       100         23.5      0.2      0.0      color_panels = []
    16       600        278.2      0.5      0.0      for color in top_N_colors:
    17       600       1024.0      1.7      0.0          b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
    18       600     220806.9    368.0      3.6          color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
    19
    20       600        243.7      0.4      0.0          color_panels.append(color_panel)
    21       100      16234.6    162.3      0.3      color_panels = np.concatenate(color_panels, axis=0)
    22
    23       100      11001.4    110.0      0.2      return color_panels.astype(np.uint8)
```

타겟 함수

실행 내용

1장 요약

프로파일링

우리 수준에 맞는 툴로

완벽하지 않아도 일단 시작해보자.

2. Python level 최적화

2.1 수정할 부분 타겟팅

NAVER
DEVIEW
2023

```
[5]: %load_ext line_profiler

[6]: %lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
File:
Function: extract_main_colors_naive at line 4

Line #      Hits         Time  Per Hit   % Time  Line
==============================================================
     4                                           def e
     5       100         77.1      0.8      0.0      p
     6       100         14.6      0.1      0.0      p
     7
     8       100    1539512.1  15395.1     24.9      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
     9       100    2849188.7  28491.9     46.0      color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    10
    11       100     442498.6   4425.0      7.1      packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
    12       100    1110719.8  11107.2     17.9      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    13       100       1329.0     13.3      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    14
    15       100         23.5      0.2      0.0      color_panels = []
    16       600        278.2      0.5      0.0      for color in top_N_colors:
    17       600       1024.0      1.7      0.0          b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
    18       600     220806.9    368.0      3.6          color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
    19
    20       600        243.7      0.4      0.0          color_panels.append(color_panel)
    21       100      16234.6    162.3      0.3      color_panels = np.concatenate(color_panels, axis=0)
    22
    23       100      11001.4    110.0      0.2      return color_panels.astype(np.uint8)
```

이미지 로딩 25%

투명 픽셀 제거 46%

2.1 수정할 부분 타겟팅

NAVER
DEVIEW
2023

```
[5]: %load_ext line_profiler

[6]: %lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
File:
Function: extract_main_colors_naive at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           def extract_main_colors_naive(image_path, N=6):
     5       100         77.1      0.8      0.0      panel_h = 64
     6       100         14.6      0.1      0.0
     7
     8       100    1539512.1  15395.1     24.9
     9       100    2849188.7  28491.9     46.0                                                  )
    10
    11       100     442498.6   4425.0      7.1      packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
    12       100    1110719.8  11107.2     17.9      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    13       100       1329.0     13.3      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    14
    15       100         23.5      0.2      0.0      color_panels = []
    16       600        278.2      0.5      0.0      for color in top_N_colors:
    17       600       1024.0      1.7      0.0          b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
    18       600     220806.9    368.0      3.6          color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
    19
    20       600        243.7      0.4      0.0          color_panels.append(color_panel)
    21       100      16234.6    162.3      0.3      color_panels = np.concatenate(color_panels, axis=0)
    22
    23       100      11001.4    110.0      0.2      return color_panels.astype(np.uint8)
```

전체 시간의 30% 수준이지만
개선이 쉬워 보이는 부분

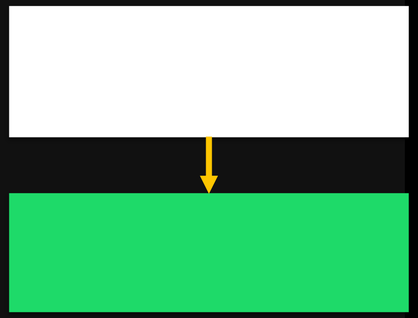
2.2 동일 출력, 빠른 연산 조사

NAVER
DEVIEW
2023

```python
packed_image = color_source[:,0] * 1e6 + color_source[:,1] * 1e3 + color_source[:,2]
unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
```

(B, G, R) → BGR
(105, 218, 30) → 105218030

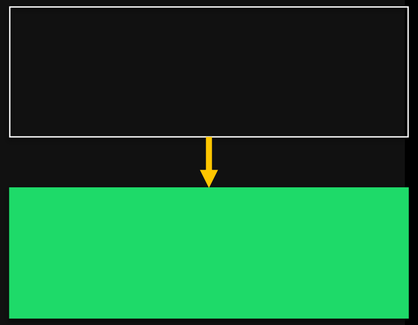
```python
color_panels = []
for color in top_N_colors:
    b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
    color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)

    color_panels.append(color_panel)
color_panels = np.concatenate(color_panels, axis=0)

return color_panels.astype(np.uint8)
```



```python
packed_image = np.left_shift(color_source[:,0], 16) + np.left_shift(color_source[:,1], 8) + color_source[:,2]
unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
```

(0x69, 0xDA, 0x1E)
→ 0x69DA1E

```python
color_panels = np.empty((panel_h*N, panel_w, 3), np.uint8)
R = np.bitwise_and(top_N_colors, 0xff)
G = np.right_shift(np.bitwise_and(top_N_colors, 0xff00), 8)
B = np.right_shift(np.bitwise_and(top_N_colors, 0xff0000), 16)
for idx, (b,g,r) in enumerate(zip(B, G, R)):
    color_panels[idx*panel_h:(idx+1)*panel_h, :, 0].fill(b)
    color_panels[idx*panel_h:(idx+1)*panel_h, :, 1].fill(g)
    color_panels[idx*panel_h:(idx+1)*panel_h, :, 2].fill(r)
return color_panels
```



2.3 결과 관찰

NAVER DEVIEW 2023

```
[6]: %lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
```

약 -18% 감소

```
[7]: %lprun -u 1e-6 -f extra range(100): extract_main_colors_fast(image_path)

Timer unit: 1e-06 s

Total time: 5.0982 s
File:
Function: extract_main_colors_fast at line 25

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    25                                           def extract_main_colors_fast(image_pa
    26       100         44.7      0.4      0.0      panel_h = 64
    27       100         18.9      0.2      0.0      panel_w = 512
    28
    29       100    1525827.3  15258.3     29.9      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    30       100    2836582.0  28365.8     55.6      color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    31
    32       100     146607.9   1466.1      2.9      packed_image = np.left_shift(color_source[:,0], 16) + np.left_shift(color_source[:,1], 8) + color_source[:,2]
    33       100     567684.8   5676.8     11.1      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    34       100        991.5      9.9      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    35
    36       100        189.2      1.9      0.0      color_panels = np.empty((panel_h*N, panel_w, 3), np.uint8)
    37       100        375.3      3.8      0.0      R = np.bitwise_and(top_N_colors, 0xff)
    38       100        385.6      3.9      0.0      G = np.right_shift(np.bitwise_and(top_N_colors, 0xff00), 8)
    39       100        269.5      2.7      0.0      B = np.right_shift(np.bitwise_and(top_N_colors, 0xff0000), 16)
    40       600        667.2      1.1      0.0      for idx, (b,g,r) in enumerate(zip(B, G, R)):
    41       600       6476.1     10.8      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 0].fill(b)
    42       600       6064.2     10.1      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 1].fill(g)
    43       600       6001.2     10.0      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 2].fill(r)
    44       100         14.3      0.1      0.0      return color_panels
```

각 파트가 차지하는 시간 비중 변화 관찰

70.9% → 85.5%

25.9% → 17.7%

4.3% → 0.5%

2.3 결과 관찰

NAVER DEVIEW 2023

```
[6]: %lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
```

```
[7]: %lprun -u 1e-6 -f extract_main_colors_fast for _ in range(100): extract_main_colors_fast(image_path)

Timer unit: 1e-06 s

Total time: 5.0982 s
File:
Function: extract_main_colors_fast at line 25

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    25                                           def extract_main_colors_fast(image_path, N=6):
    26       100         44.7      0.4      0.0      panel_h = 64
    27       100         18.9      0.2      0.0      panel_w = 512
    28
    29       100    1525827.3  15258.3     29.9      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    30       100    2836582.0  28365.8     55.6      color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    31
    32       100     146607.9   1466.1      2.9      packed_image = np.left_shift(color_source[:,0], 16) + np.left_shift(color_source[:,1], 8) + color_source[:,2]
    33       100     567684.8   5676.8     11.1      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    34       100        991.5      9.9      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    35
    36       100        189.2      1.9      0.0      color_panels = np.empty((panel_h*N, panel_w, 3), np.uint8)
    37       100        375.3      3.8      0.0      R = np.bitwise_and(top_N_colors, 0xff)
    38       100        385.6      3.9      0.0      G = np.right_shift(np.bitwise_and(top_N_colors, 0xff00), 8)
    39       100        269.5      2.7      0.0      B = np.right_shift(np.bitwise_and(top_N_colors, 0xff0000), 16)
    40       600        667.2      1.1      0.0      for idx, (b,g,r) in enumerate(zip(B, G, R)):
    41       600       6476.1     10.8      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 0].fill(b)
    42       600       6064.2     10.1      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 1].fill(g)
    43       600       6001.2     10.0      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 2].fill(r)
    44       100         14.3      0.1      0.0      return color_panels
```

70.9% → 85.5%

25.9% → 17.7%

더 이상 고려하지 않음

4.3% → 0.5%

2.3 결과 관찰

NAVER
DEVIEW
2023

```
[6]: %lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
```

```
[7]: %lprun -u 1e-6 -f extract_main_colors_fast for _ in range(100): extract_main_colors_fast(image_path)

Timer unit: 1e-06 s

Total time: 5.0982 s
File:
Function: extract_main_colors_fast at line 25

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    25                                           def extract_main_colors_fast(image_path, N=6):
    26       100         44.7      0.4      0.0      panel_h = 64
    27       100         18.9      0.2      0.0      panel_w = 512
    28
    29       100    1525827.3  15258.3     29.9      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    30       100    2836582.0  28365.8     55.6      color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    31
    32       100     146607.9   1466.1      2.9      packed_image = np.left_shift(color_source[:,0], 16) + np.left_shift(color_source[:,1], 8) + color_source[:,2]
    33       100     567684.8   5676.8     11.1      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    34       100        991.5      9.9      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    35
    36       100        189.2      1.9      0.0      color_panels = np.empty((panel_h*N, panel_w, 3), np.uint8)
    37       100        375.3      3.8      0.0      R = np.bitwise_and(top_N_colors, 0xff)
    38       100        385.6      3.9      0.0      G = np.right_shift(np.bitwise_and(top_N_colors, 0xff00), 8)
    39       100        269.5      2.7      0.0      B = np.right_shift(np.bitwise_and(top_N_colors, 0xff0000), 16)
    40       600        667.2      1.1      0.0      for idx, (b,g,r) in enumerate(zip(B, G, R)):
    41       600       6476.1     10.8      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 0].fill(b)
    42       600       6064.2     10.1      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 1].fill(g)
    43       600       6001.2     10.0      0.1          color_panels[idx*panel_h:(idx+1)*panel_h, :, 2].fill(r)
    44       100         14.3      0.1      0.0      return color_panels
```

이제는 정말 고쳐야겠다...

70.9% → 85.5%

25.9% → 17.7%

4.3% → 0.5%

2.4 만족스러울 때까지 반복

NAVER DEVIEW 2023

개선 전

```
%lprun -u 1e-6 -f extract_main_colors_naive for _ in range(100): extract_main_colors_naive(image_path)

Timer unit: 1e-06 s

Total time: 6.19295 s
```

1차 개선

```
%lprun -u 1e-6 -f extract_main_colors_fast for _ in range(100): extract_main_colors_fast(image_path)

Timer unit: 1e-06 s

Total time: 5.0982 s
```

```
%lprun -u 1e-6 -f extract_main_colors_faster for _ in range(100): extract_main_colors_faster(image_path)

Timer unit: 1e-06 s

Total time: 2.31268 s
File: /home1/irteam/users/juhyeok.mun/Deview2023/python_level/A_fatster_color_extraction.py
Function: extract_main_colors_faster at line 46

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    46                                           def extract_main_colors_faster(image_path, N=6):
    47       100         29.4      0.3      0.0      panel_h = 64
    48       100         14.5      0.1      0.0      panel_w = 512
    49
    50       100    1387415.2  13874.2     60.0      image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    51       100      37432.1    374.3      1.6      roi = image[:,:,-1].ravel() != 0
    52
    53                                               packed_image = np.left_shift(image[:, :, 0].ravel().astype(np.uint32)[roi], 16) \
    54                                                            + np.left_shift(image[:, :, 1].ravel().astype(np.uint32)[roi], 8) \
    55       100     301337.1   3013.4     13.0                  + image[:, :, 2].ravel().astype(np.uint32)[roi]
    56       100     565732.3   5657.3     24.5      unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    57       100        780.1      7.8      0.0      top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]
    58
    59       100        152.1      1.5      0.0      color_panels = np.empty((panel_h*N, panel_w, 3), np.uint8)
    60       100        288.1      2.9      0.0      R = np.bitwise_and(top_N_colors, 0xff)
    61       100        311.8      3.1      0.0      G = np.right_shift(np.bitwise_and(top_N_colors, 0xff00), 8)
    62       100        270.5      2.7      0.0      B = np.right_shift(np.bitwise_and(top_N_colors, 0xff0000), 16)
    63       600        582.8      1.0      0.0      for idx, (b,g,r) in enumerate(zip(B, G, R)):
    64       600       6246.2     10.4      0.3          color_panels[idx*panel_h:(idx+1)*panel_h, :, 0].fill(b)
    65       600       6064.7     10.1      0.3          color_panels[idx*panel_h:(idx+1)*panel_h, :, 1].fill(g)
    66       600       6006.7     10.0      0.3          color_panels[idx*panel_h:(idx+1)*panel_h, :, 2].fill(r)
    67       100         14.5      0.1      0.0      return color_panels
```

최종 개선 후
61.9ms → 23.1ms (-62.7%)

24.9% → 60.0% 이미지 로딩 점유율 증가

이제 이 코드는
작성자만 읽을 수 있습니다.

2장 요약

Python 패키지

같은 출력을 내는 여러 문법을 비교해

병목구간을 더 빠르게 바꿔보자

3. Semi-C level 최적화

3.1 Cython 및 Numba

NAVER
DEVIEW
2023

블로그 등에서 쉽게 접하는 Cython 예제

```python
def cumulative_sum_python(N):
    sum_value = 0
    for i in range(N):
        sum_value += i
    return sum_value
```

```
%timeit -r 7 -n 10 cumulative_sum_python(100000000)
3.47 s ± 29.1 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```python
def cumulative_sum_cython_typed(int N):
    cdef long sum_value = 0
    cdef int i = 0
    for i in range(N):
        sum_value += i
    return sum_value
```

이게 그래서 정말 내 코드도 빠르게 해줄까?

```
%timeit -r 7 -n 10 cumulative_sum_cython_typed(100000000)
14.6 ms ± 59.6 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

3.1 Cython 및 Numba

NAVER
DEVIEW
2023

NumPy를 활용하는 내 코드

```python
def extract_main_colors_naive(image_path, N=6):
    panel_h = 64
    panel_w = 512

    image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)

    packed_image = color_source[:,0] * 1e6 \
                   + color_source[:,1] * 1e3 \
                   + color_source[:,2]
    unique_colors, pixel_counts = np.unique(packed_image,
                                            return_counts=True)
    top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]

    color_panels = []
    for color in top_N_colors:
        b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
        color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
        color_panels.append(color_panel)
    return color_panels
```

1장에서 만든 그 예제
(2장의 개선 내용 적용 전)

```
%timeit extract_main_colors_naive(image_path)
55.8 ms ± 52.1 µs per loop (mean ± std. dev.
```

NumPy와 같이 사용하는
방법은 공식 문서* 참고

* Working with NumPy: https://cython.readthedocs.io/en/stable/src/tutorial/numpy.html

3.1 Cython 및 Numba

내 코드에 Cython 적용해보기

```
def extract_main_colors_naive_cython(str image_path, int N=6):
    cdef int panel_h = 64
    cdef int panel_w = 512

    cdef np.ndarray[np.uint8_t, ndim=3] image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    cdef np.ndarray[np.uint32_t, ndim=2] color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    cdef np.ndarray[np.uint32_t, ndim=1] packed_image = (color_source[:,0] * 1e6 \
                                                       + color_source[:,1] * 1e3 \
                                                       + color_source[:,2]).astype(np.uint32)
    cdef np.ndarray[np.uint32_t, ndim=1] unique_colors
    cdef np.ndarray[np.int64_t, ndim=1] pixel_counts
    unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    cdef np.ndarray[np.uint32_t, ndim=1] top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]

    cdef list color_panels = []
    cdef unsigned int color
    cdef int b, g, r
    cdef np.ndarray[np.uint8_t, ndim=3] color_panel
    for color in top_N_colors:
        b, g, r = int(color//1e6), int(color//1e3 % 1e3), int(color % 1e3)
        color_panel = np.ones((panel_h, panel_w, 3), np.uint8) * np.array((b, g, r), np.uint8)
        color_panels.append(color_panel)
    return color_panels
```

귀찮고 지저분하지만
속도를 극적으로 개선해줄거니까...

NAVER
DEVIEW
2023

3.1 Cython 및 Numba

NAVER
DEVIEW
2023

내 코드에 Cython을 적용한 결과 비교

```python
def extract_main_colors_naive(image_path, N=6):
    panel_h = 64
    panel_w = 512

    image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)

    packed_image = color_source[:,0] * 1e6 \
                   + color_source[:,1] * 1e3 \
                   + color_source[:,2]
    unique_colors, pixel_counts = np.unique(packed_image,
                                            return_counts=True)
    top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]

    color_panels = []
    for color in top_N_colors:
        b, g, r = color//1e6, color//1e3 % 1e3, color % 1e3
        color_panel = np.ones((panel_h, panel_w, 3)) * (b, g, r)
        color_panels.append(color_panel)
    return color_panels
```

```
%timeit extract_main_colors_naive(image_path)
```

```
55.8 ms ± 52.1 µs per loop (mean ± std. dev. of 7 runs, 10 loops ea
```

```python
def extract_main_colors_naive_cython(str image_path, int N=6):
    cdef int panel_h = 64
    cdef int panel_w = 512

    cdef np.ndarray[np.uint8_t, ndim=3] image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    cdef np.ndarray[np.uint32_t, ndim=2] color_source = (image[image[:,:,-1] != 0, :3]).astype(np.uint32)
    cdef np.ndarray[np.uint32_t, ndim=1] packed_image = (color_source[:,0] * 1e6 \
                                                         + color_source[:,1] * 1e3 \
                                                         + color_source[:,2]).astype(np.uint32)
    cdef np.ndarray[np.uint32_t, ndim=1] unique_colors
    cdef np.ndarray[np.int64_t, ndim=1] pixel_counts
    unique_colors, pixel_counts = np.unique(packed_image, return_counts=True)
    cdef np.ndarray[np.uint32_t, ndim=1] top_N_colors = unique_colors[np.argsort(pixel_counts)[::-1][:N]]

    cdef list color_panels = []
    cdef unsigned int color
    cdef int b, g, r
    cdef np.ndarray[np.uint8_t, ndim=3] color_panel
    for color in top_N_colors:
        b, g, r = int(color//1e6), int(color//1e3 % 1e3), int(color % 1e3)
        color_panel = np.ones((panel_h, panel_w, 3), np.uint8) * np.array((b, g, r), np.uint8)
        color_panels.append(color_panel)
    return color_panels
```

```
%timeit extract_main_colors_naive_cython(image_path)
```

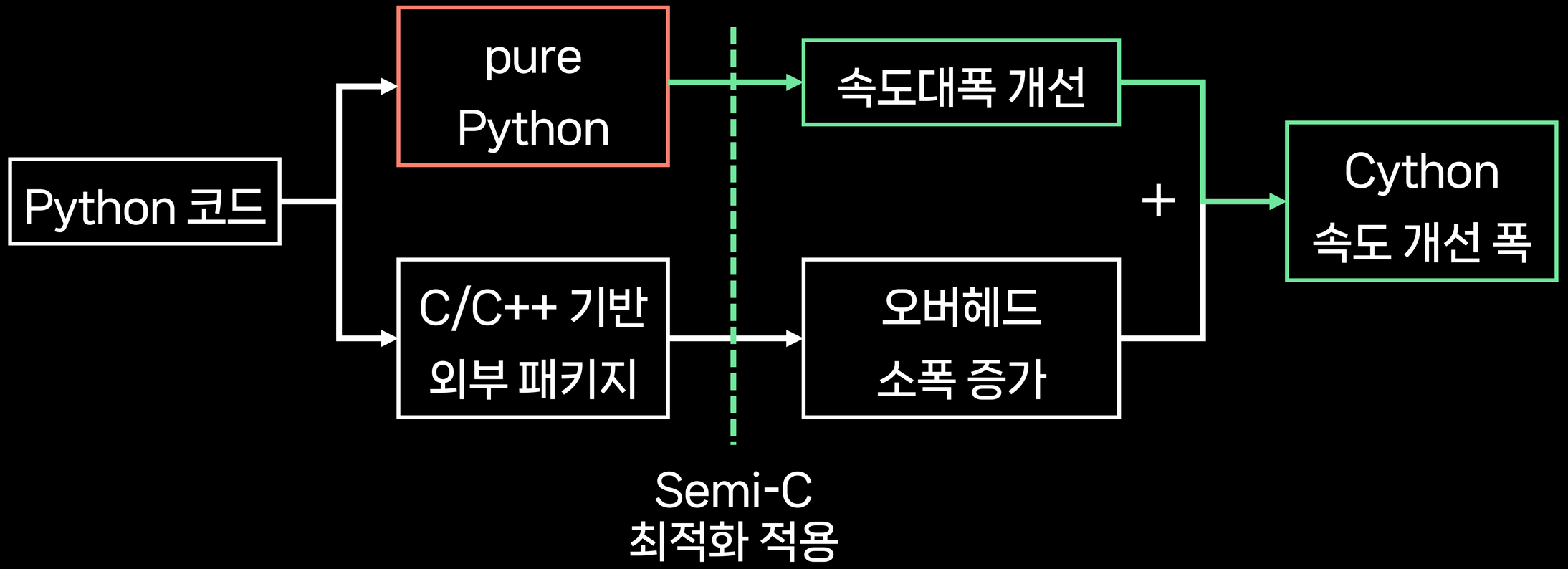
```
55.6 ms ± 161 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

55.8ms → 55.6ms ???

3.2 유용한 경우

NAVER
DEVIEW
2023

코드 내 pure Python 구현의 비중이 너무 큰 경우



3장 요약

Cython/Numba

써야하는 상황이 따로 있다.

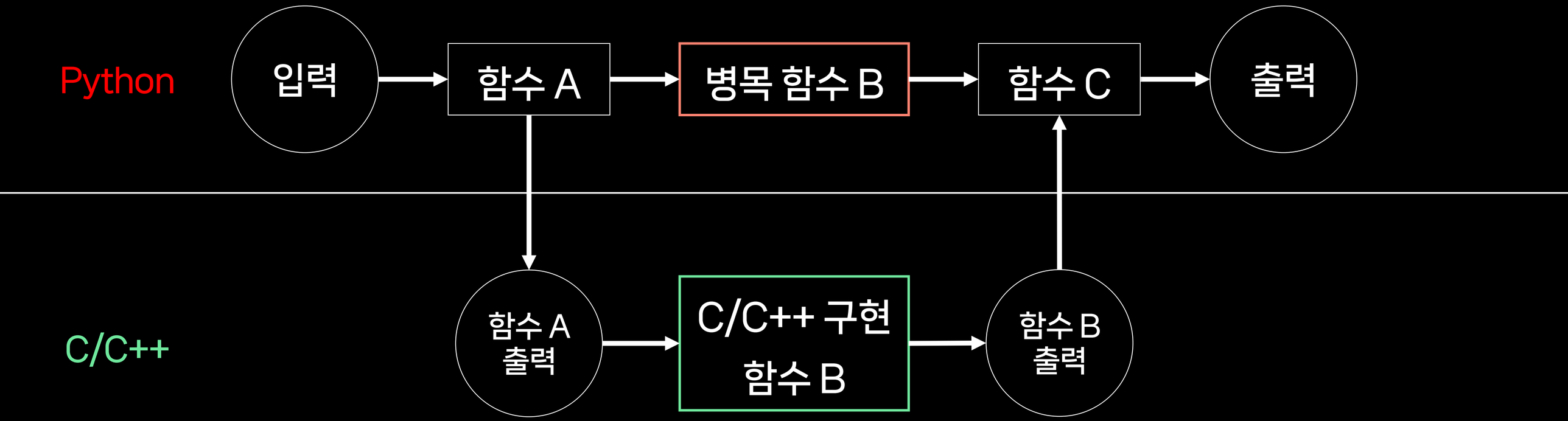
우리를 위한 툴이 아니라면 넘어가자.

4. C level 최적화

4.1 Python/C API

NAVER
DEVIEW
2023

= Python ↔ C/C++ 데이터 인터페이스



* Python/C API Reference Manual: https://docs.python.org/3/c-api/intro.html

4.1 Python/C API

NAVER DEVIEW 2023

Python/C API 예제?

```c
#define PY_SSIZE_T_CLEAN
#include "Python.h"

static PyObject*
cumulative_sum(PyObject *self, PyObject *args){
    unsigned long N;
    if (!PyArg_ParseTuple(args, "k", &N)){
        return NULL;
    }
    unsigned long sum = 0;
    for (size_t i = 0 ; i < N ; i++){
        sum += i;
    }
    PyObject *results = Py_BuildValue("k", sum);
    return results;
}
```

Python > C/C++

C/C++ impl

C/C++ > Python

```python
def cumulative_sum_python(N):
    sum_value = 0
    for i in range(N):
        sum_value += i
    return sum_value
```

```
%timeit cumulative_sum_python(100000000)
3.19 s ± 11.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Python 3.19s

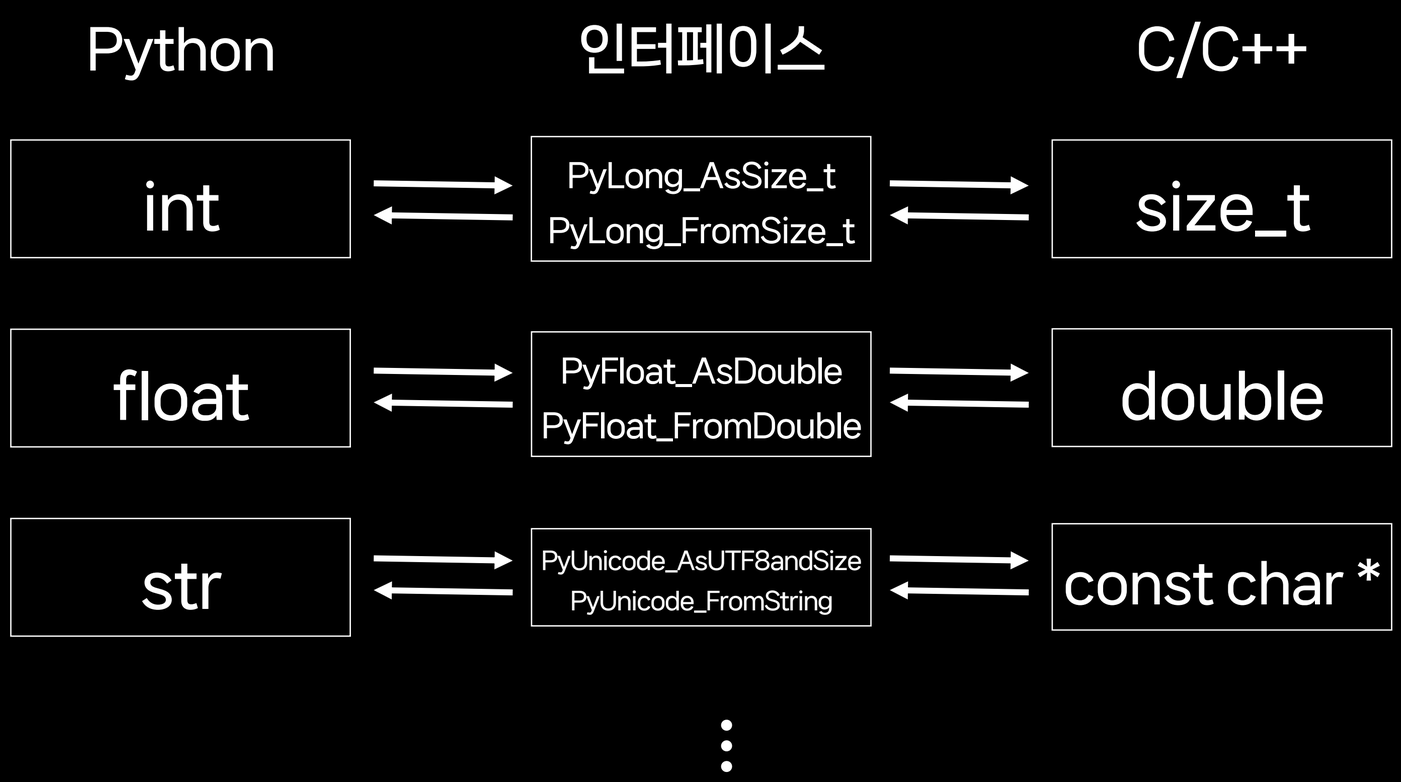
```
%timeit cumulative_sum(100000000)
20.6 ms ± 205 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

C/C++ 20.6ms

4.1 Python/C API

NAVER
DEVIEW
2023

Python/C API 문서의 주요 내용은 Python 기본 타입에 충실



PyArg_ParseTuple

PyList_New

PyDict_New

PyTuple_New

Py_BuildValue

4.2 Python/C API (+ NumPy + OpenCV)

NAVER DEVIEW 2023

우리에게 추가로 필요한 튜토리얼

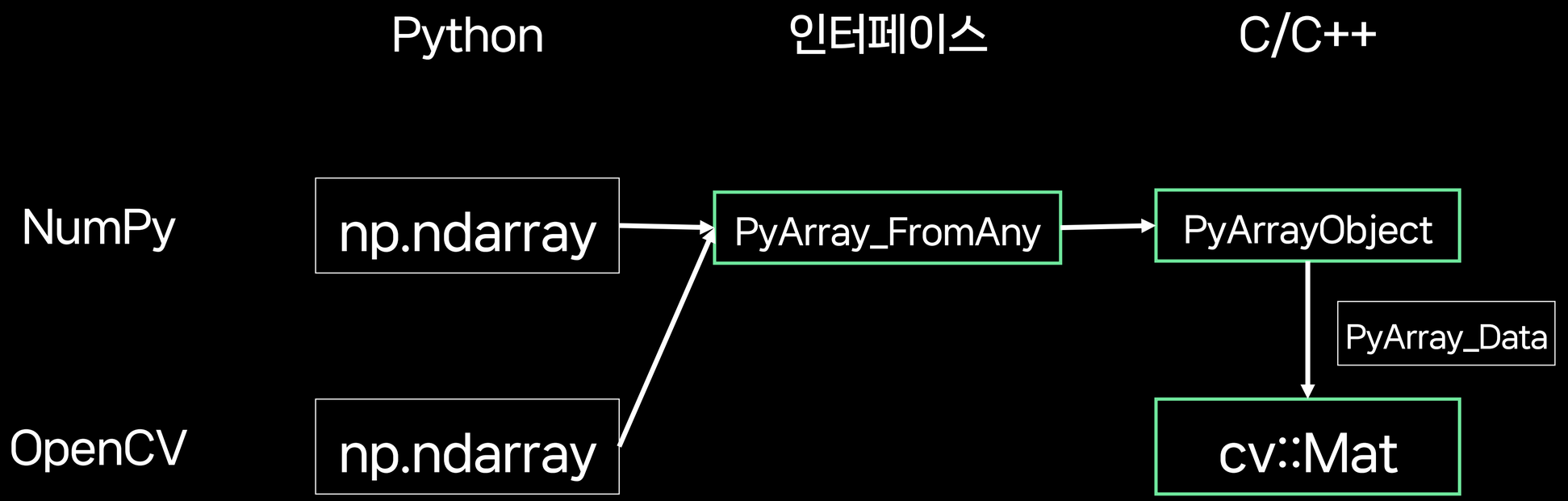
	Python	인터페이스	C/C++
NumPy	np.ndarray	???	???
OpenCV	np.ndarray	???	cv::Mat

* NumPy C-API Reference Manual: https://numpy.org/doc/stable/reference/c-api/index.html

4.2 Python/C API (+ NumPy + OpenCV)

NAVER DEVIEW 2023

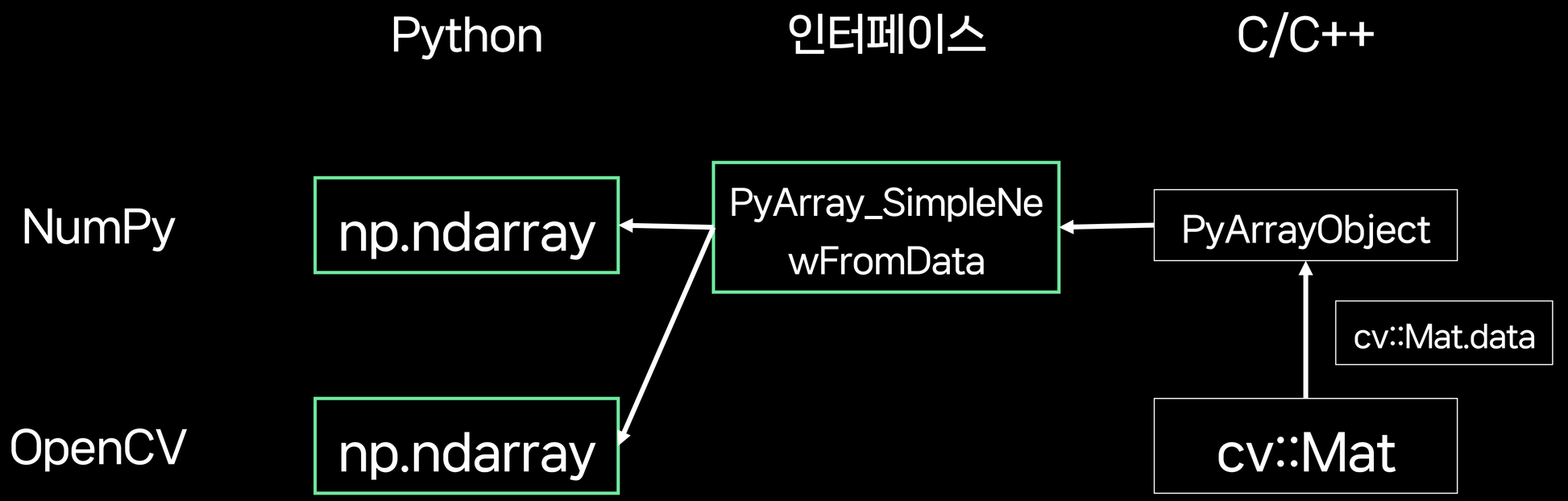
cv::Mat까지 가는 여정



4.2 Python/C API (+ NumPy + OpenCV)

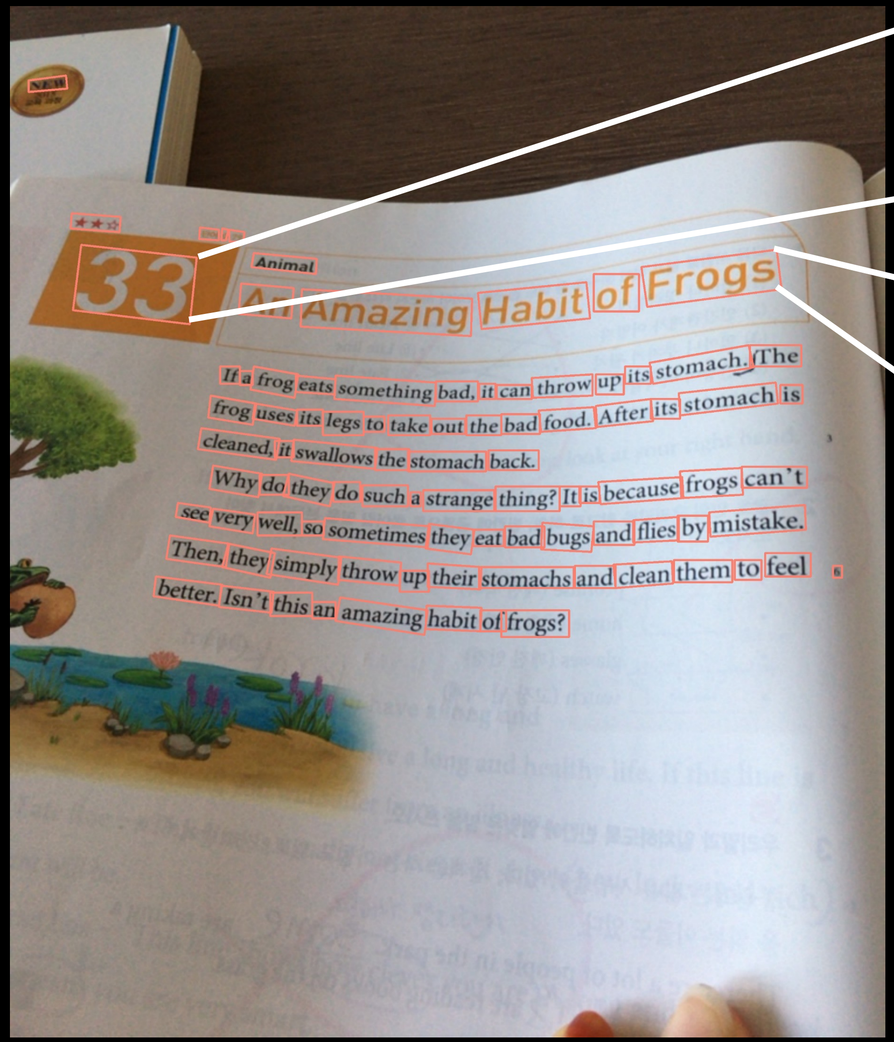
NAVER DEVIEW 2023

np.ndarray로 돌아오는 여정



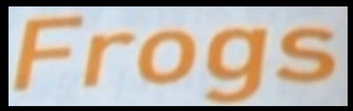
4.3 Python/C API 적용 예시

NAVER
DEVIEW
2023



텍스트
검출 결과

인식이 더 쉬운
정방향 글자



warp를 이용한
실제 crop 결과

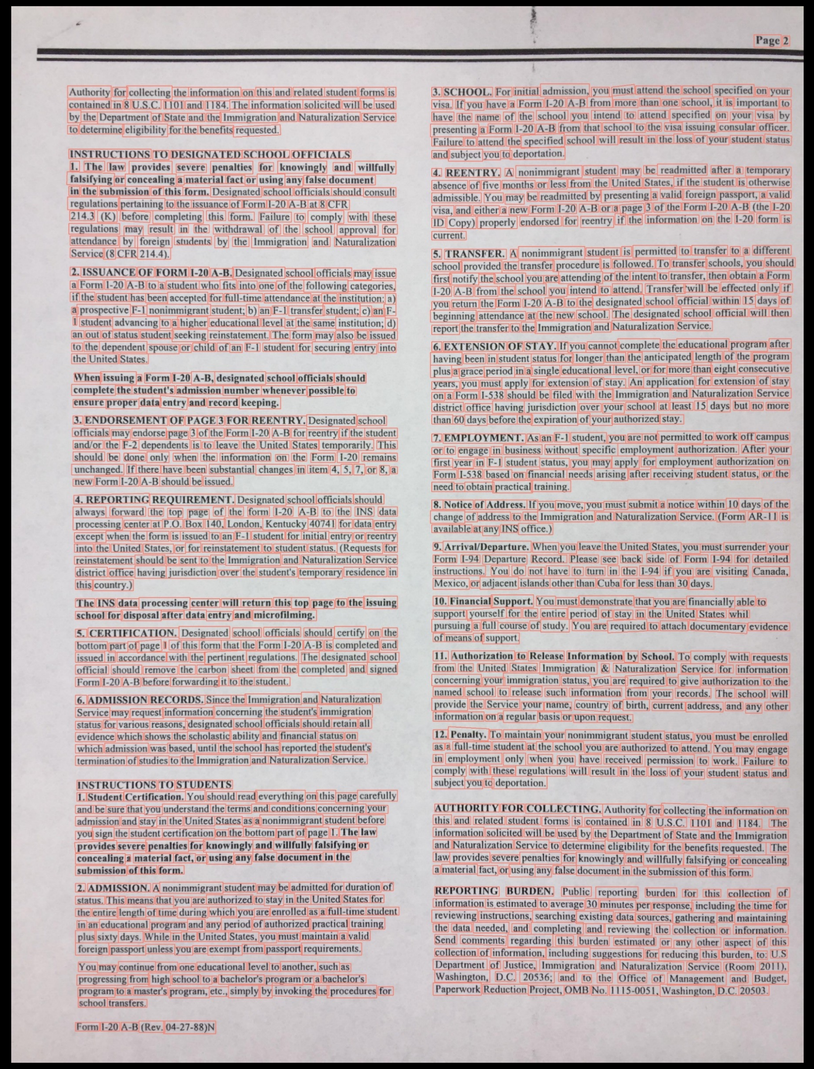
```python
def crop(image, bbox):
    lt, rt, rb, lb = bbox
    widthA = np.sqrt(((rb[0] - lb[0]) ** 2) + ((rb[1] - lb[1]) ** 2))
    widthB = np.sqrt(((rt[0] - lt[0]) ** 2) + ((rt[1] - lt[1]) ** 2))
    heightA = np.sqrt(((rt[0] - rb[0]) ** 2) + ((rt[1] - rb[1]) ** 2))
    heightB = np.sqrt(((lt[0] - lb[0]) ** 2) + ((lt[1] - lb[1]) ** 2))
    maxWidth, maxHeight = max(int(widthA), int(widthB)), max(int(heightA), int(heightB))

    dst = np.array([[0, 0],
                    [maxWidth - 1, 0],
                    [maxWidth - 1, maxHeight - 1],
                    [0, maxHeight - 1]], dtype=np.float32)
    M = cv2.getPerspectiveTransform(bbox.astype(np.float32), dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
    return warped
```

모든 박스에 대해
for loop

```python
for bbox in bboxes:
    cropped_images.append(
        crop(image, bbox)
    )
```

4.3 Python/C API 적용 예시

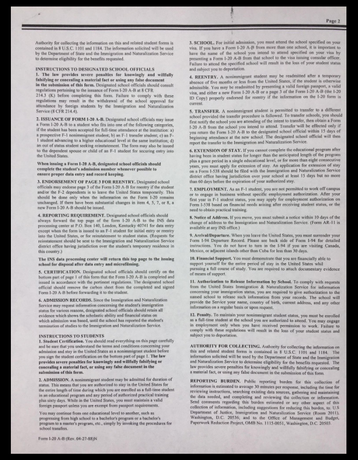
NAVER DEVIEW 2023




```python
def crop(image, bbox):
    ...
    M = cv2.getPerspectiveTransform(bbox.astype(np.float32), dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
    ...

for bbox in bboxes:
    cropped_images.append(
        crop(image, bbox)
    )
```

bboxes.shape
= (1528, 4, 2)

```
%timeit -r 7 -n 100 image_crop_python(image, bboxes)

87.7 ms ± 1.17 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```



각각의 crop이 독립적이므로 **병렬처리**를 하고싶지만,

원본 이미지가 **GIL**에 묶여있어 동시 접근 불가

C/C++에 가져가서 자유롭게 crop 해보자..

4.3 Python/C API 적용 예시

NAVER
DEVIEW
2023

같은 라이브러리를 이용하는 경우 코어 구현이 크게 달라지지 않음

```python
def crop(image, bbox):
    lt, rt, rb, lb = bbox

    widthA = np.sqrt(((rb[0] - lb[0]) ** 2) + ((rb[1] - lb[1]) ** 2))
    widthB = np.sqrt(((rt[0] - lt[0]) ** 2) + ((rt[1] - lt[1]) ** 2))
    maxHeight = max(int(heightA), int(heightB))

    heightA = np.sqrt(((rt[0] - rb[0]) ** 2) + ((rt[1] - rb[1]) ** 2))
    heightB = np.sqrt(((lt[0] - lb[0]) ** 2) + ((lt[1] - lb[1]) ** 2))
    maxWidth = max(int(widthA), int(widthB))

    dst = np.array([[0, 0],
                    [maxWidth - 1, 0],
                    [maxWidth - 1, maxHeight - 1],
                    [0, maxHeight - 1]], dtype=np.float32)
    M = cv2.getPerspectiveTransform(bbox.astype(np.float32), dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
    return warped
```

crop (Python)

```cpp
void crop(cv::InputArray src_, cv::InputArray image_, cv::OutputArray word_image_){
    cv::Mat src = src_.getMat();
    cv::Mat image = image_.getMat();

    float max_width, max_height, width_a, width_b, height_a, height_b;
    cv::Point2f diff;

    // known as it is faster than cv::norm
    diff = src.at<cv::Point2f>(2) - src.at<cv::Point2f>(3);
    width_a = cv::sqrt(diff.x*diff.x + diff.y*diff.y);
    diff = src.at<cv::Point2f>(1) - src.at<cv::Point2f>(0);
    width_b = cv::sqrt(diff.x*diff.x + diff.y*diff.y);
    max_width = std::max(width_a, width_b);

    diff = src.at<cv::Point2f>(1) - src.at<cv::Point2f>(2);
    height_a = cv::sqrt(diff.x*diff.x + diff.y*diff.y);
    diff = src.at<cv::Point2f>(0) - src.at<cv::Point2f>(3);
    height_b = cv::sqrt(diff.x*diff.x + diff.y*diff.y);
    max_height = std::max(height_a, height_b);

    float dst_data[8] = {0, 0, max_width-1, 0, max_width-1, max_height-1, 0, max_height-1};
    cv::Mat dst(4, 2, CV_32FC1, dst_data);

    cv::Mat M = cv::getPerspectiveTransform(src, dst);
    cv::warpPerspective(image, word_image_.getMatRef(), M, cv::Size(max_width, max_height));
}
```

crop (C/C++)

4.3 Python/C API 적용 예시

NAVER DEVIEW 2023

```
static PyObject*
image_crop(PyObject *self, PyObject *args){
    PyObject *py_image;
    PyObject *py_bboxes;
    if (!PyArg_ParseTuple(args, "OO",
                          &py_image, &py_bboxes)){
        return NULL;
    }
    PyArrayObject *image = (PyArrayObject *)PyArray_FromAny(
                                        py_image,
                                        PyArray_DescrFromType(NPY_UINT8),
                                        3, 3,
                                        NPY_ARRAY_CARRAY, NULL);
    PyArrayObject *bboxes = (PyArrayObject *)PyArray_FromAny(
                                        py_bboxes,
                                        PyArray_DescrFromType(NPY_FLOAT32),
                                        3, 3,
                                        NPY_ARRAY_CARRAY,
                                        NULL);
    if (image == NULL || bboxes == NULL) {
      return Py_None;
    }

    int num_bbox = bboxes->dimensions[0];
    PyObject *py_word_images = PyList_New(num_bbox);
    if (bboxes->dimensions[1] != 4){
        return Py_None;
    }

    cv::Mat image_mat(PyArray_DIM(image, 0), PyArray_DIM(image, 1), CV_8UC3, PyArray_DATA(image));
    cv::Mat bboxes_mat(PyArray_DIM(bboxes, 0), PyArray_DIM(bboxes, 1), CV_32FC2, PyArray_DATA(bboxes));

    std::vector<cv::Mat> word_image_mats;
    for (int k = 0 ; k < num_bbox ; k++){
        cv::Mat src = bboxes_mat.row(k);
        cv::Mat word_image_mat;
        crop(src, image_mat, word_image_mat);
        PyArrayObject *py_word_image = cvimage2pyarray(word_image_mat);
        PyList_SetItem(py_word_images, k, (PyObject *)py_word_image);
    }

    PyObject *results = Py_BuildValue("O", py_word_images);

    Py_XDECREF(image);
    Py_XDECREF(py_word_images);
    Py_XDECREF(py_bboxes);
    return results;
}
```

Python → C/C++

PyArray → cv::Mat

Core implementation

C/C++ → Python

Python

```
%timeit -r 7 -n 100 image_crop_python(image, bboxes)
```

```
87.7 ms ± 1.17 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

C/C++

```
%timeit -r 7 -n 100 image_crop_cpp(image, bboxes)
```

```
45 ms ± 556 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

4.4 C/C++ OpenMP 활용

NAVER DEVIEW 2023

Core implementation 위아래로 OpenMP 구문 추가

```
std::vector<cv::Mat> word_image_mats;
for (int k = 0 ; k < num_bbox ; k++){
    cv::Mat src = bboxes_mat.row(k);
    cv::Mat word_image_mat;
    crop(src, image_mat, word_image_mat);
    PyArrayObject *py_word_image = cvimage2pyarray(word_image_mat);
    PyList_SetItem(py_word_images, k, (PyObject *)py_word_image);
}
```

```
std::vector<cv::Mat> word_image_mats;
std::vector<int> indices;
#pragma omp parallel
{
    std::vector<cv::Mat> word_image_mats_;
    std::vector<int> indices_;
    #pragma omp for nowait
    for (int k = 0 ; k < num_bbox ; k++){
        cv::Mat src = bboxes_mat.row(k);
        cv::Mat word_image_mat;
        crop(src, image_mat, word_image_mat);
        word_image_mats_.push_back(word_image_mat);
        indices_.push_back(k);
    }
    #pragma omp critical
    {
        word_image_mats.insert(word_image_mats.end(), word_image_mats_.begin(), word_image_mats_.end());
        indices.insert(indices.end(), indices_.begin(), indices_.end());
    }
}

for (int k = 0 ; k < num_bbox ; k++){
    PyArrayObject *py_word_image = cvimage2pyarray(word_image_mats[k]);
    PyList_SetItem(py_word_images, indices[k], (PyObject *)py_word_image);
}
```

```
extra_compile_args=["-fopenmp", "-std=c++11"],
extra_link_args=["-lgomp"],
```

추가된 빌드 옵션

Python

```
%timeit -r 7 -n 100 image_crop_python(image, bboxes)
87.7 ms ± 1.17 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

C/C++

```
%timeit -r 7 -n 100 image_crop_cpp(image, bboxes)
45 ms ± 556 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

C/C++ w/ OpenMP

```
%timeit -r 7 -n 100 image_crop_cpp_openmp(image, bboxes)
4.64 ms ± 139 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

87.7ms → **4.64ms (-94.7%)**

4장 요약

Python/C API 꼭 한번 써보라.

진입 장벽은 있지만 개발이 자유로워진다.

이런 때 아니면 언제 C/C++에 입문해볼까?

WRAP UP

NAVER
DEVIEW
2023

1. line profiler를 활용해 병목 구간을 찾고 (프로파일링)

2. Python 패키지의 동일 출력 다른 표현을 비교하고 (Python 최적화)

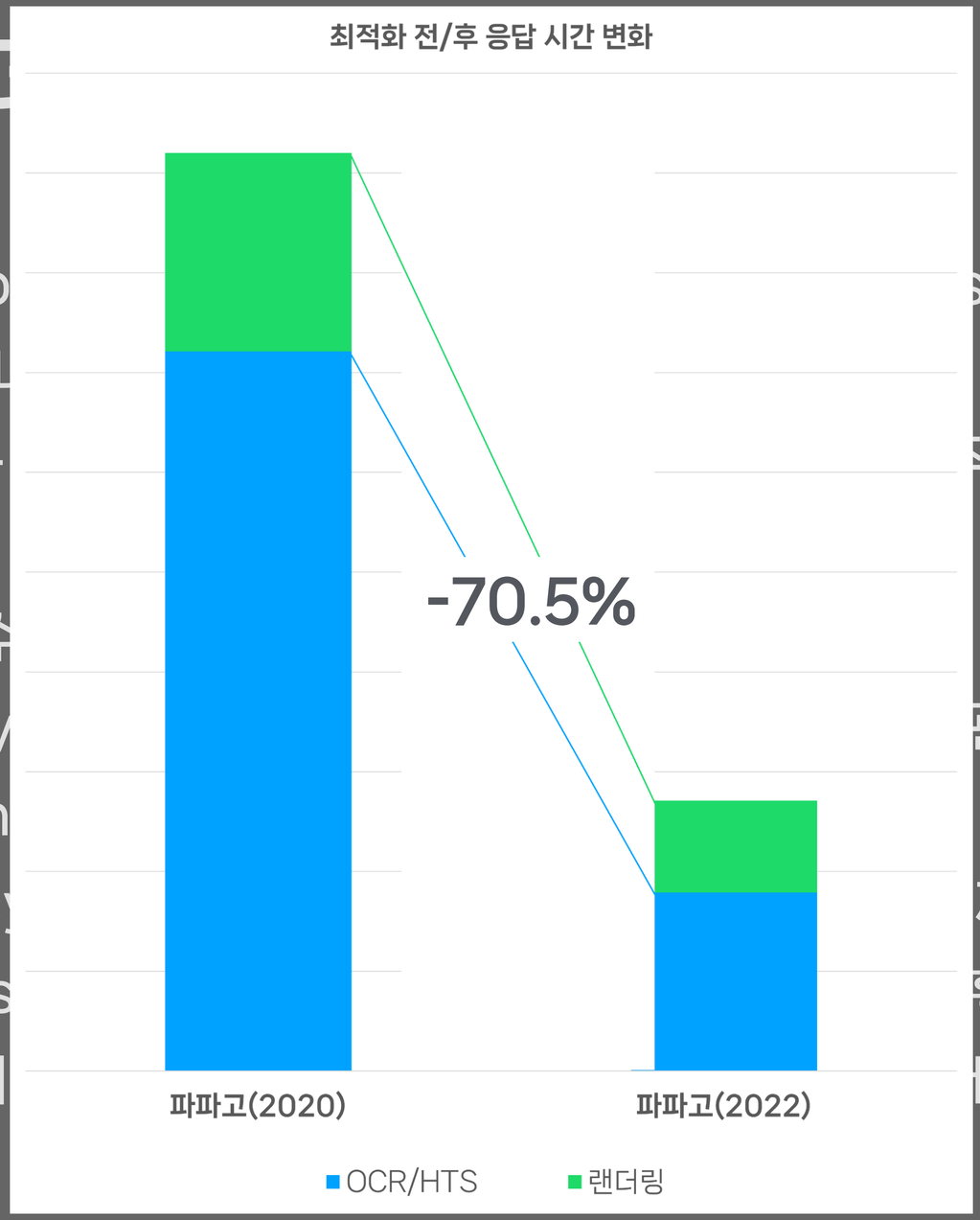
3. Cython, Numba는 유효한 상황인지 검토해본 후 (Semi-C 최적화)

4. Python/C API를 이용해 C/C++에 입문해보자. (C/C++ 최적화)

사실 이것저것 다 이만

NAVER DEVIEW 2023

- Container 환경에서 cp s 환경 변수를 CPU 제한량에 맞추어 꼭 설정하셔야 합니
- omp_num_threads는 직접 사용하지 않은 곳에서도 영향을 미칩니다.
- PyTorch에서 GPU 변수 확인하세요.
- PyList_SetItem vs Py 관리를 달리하는 API를 꼼꼼히 살펴보셔야 mem
- Intel Memory Latency 제가 없는지 확인 해보세요.
- nvidia-smi에서 Persis 확인하세요.
- GPU에 부하가 걸렸을 때 세요.
- ...



Q & A

Thank You