

2015학년도 2학기

컴퓨터 시스템 기초 팀프로젝트

KNN을 통한 문자인식 프로그램 설계 및 최적화



박형민 201120225
이유진 201021654
허경민 201021687
박승현 201122186
이원희 201320190

Contents

1. 서론	3
1.1 프로젝트 동기 & 목표	3
2. 본론	3
2.1 개념	3
2.1.1 광학 문자 인식 (OCR) 정의	3
2.1.2 데이터 전처리 과정	4
2.1.3 KNN 알고리즘	5
2.2 프로그램 개요	7
2.2.1 외부 모듈	7
2.2.2 함수 기능 명세	8
2.1.3 Training set	12
2.3 프로그램 수행 결과	12
2.4 성능 측정	13
2.4.1 성능 측정에 사용된 이미지	13
2.4.2 프로파일링 수행 결과	
2.4.3 수행시간 측정	
2.5 성능 최적화	14
3. 결론	15
4. 별첨-소스코드	15
5. 참조	15 □

1. 서론

1.1 프로젝트 동기 & 목표

전자문서와 같은 디지털 시스템이 사무 및 행정 업무에 본격적으로 도입되기 이전엔 많은 문서들이 수기로 작성되거나 인쇄물의 형태로 보관되었다. 하지만 최근들어 전자정부, 스마트 팩토리 등의 도입으로 과거에 생산된 수기/인쇄 데이터를 디지털 자료로 변환할 필요가 대두되었고, 이를 사람이 일일이 옮기는 데에 지나치게 많은 시간과 비용을 요구되고 있는 상황이다. 이에 대안으로 사용될 수 있는 광학 문자 인식(OCR)은 지면이나

이미지 속의 문자를 인식해 자동으로 디지털 텍스트로 변환하도록 하여 데이터 입력의 부담을 줄여주고, 정보의 저장 및 접근을 용이하게 해 주는 기술이다.

현재 이 기술은 영구 보존 문서의 전산화, 자동차 번호판 인식 등 여러 분야에서 활용되고 있으며, 우리는 본 프로젝트에서 KNN 알고리즘을 활용하여 아날로그 문서를 디지털 텍스트로 변환하는 프로그램을 파이썬으로 구현하고, 수행 시간 측정 및 profiling 과정을 거쳐 문자 분류에 걸리는 연산 시간을 최적화하는 것을 이번 프로젝트의 목표로 하였다.

2. 본론

2.1 개념

2.1.1 광학 문자 인식 (OCR) 정의

광학 문자 인식 (OCR : Optical Character Recognition)은 수기로 작성되거나 인쇄로 생성된 문자를 영상화 한 후 한글, 영문, 숫자 폰트에 대해 편집 가능한 텍스트로 변환하고 저장할 수 있게 해주는 기술이다.

1931년 이스라엘의 물리학자이자 발명가인 Emanuel Goldberg가 개발한 "Statistical machine" 특허가 공개되었다.¹ 이는 마이크로필름에 저장된 광학 데이터를 패턴 매칭을 통해 읽는 기술이었다. 이 특허는 IBM에 의해 취득되었으며 이후 OCR 의 기초가 되었다. 현재 OCR은 의료, 행정, 보안 등 여러 분야에서 자동차 번호판 자동 인식, 필기 인식 등으로 광범위하게 쓰이고 있다.

2.1.2 데이터 전처리 과정

문자인식 시스템에서 인식률을 높이기 위한 이미지의 전처리 과정은 반드시 선행되어야 하는 중요한 작업이다. 이러한 선행 작업들은 영상화된 문서 이미지에서 문자의 영역을 정확히 결정 할 수 있게 하여, 원본 영상을 그대로 사용하는 것에 비해 문자의 인식률을 개선한다.

● 노이즈 제거

¹ Goldberg, Emanuel. "Statistical machine." U.S. Patent No. 1,838,389. 29 Dec. 1931.

이미지의 잡음은 이미지가 촬영되는 과정에서 광량과 촬영장비의 성능 등 다양한 원인으로 인해 필연적으로 발생한다. 이렇게 발생하는 노이즈를 제거하기 위해서 잡음을 제거하는 필터 알고리즘을 사용한다. 이번 프로젝트에서는 Gaussian filter를 이용하여 잡음을 제거하는 전처리 작업을 진행 하였다.

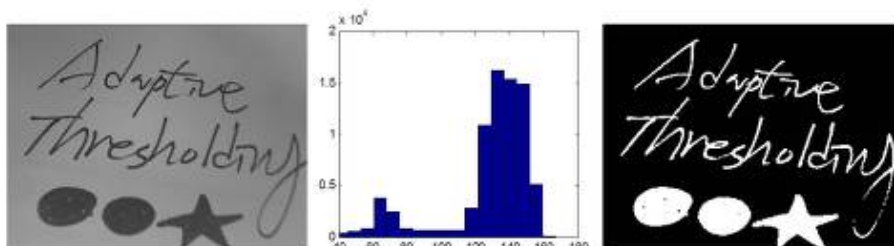


【 원본(좌) Gaussian Smoothing 적용 후(우) 】

Gaussian smoothing이란 Gaussian function 을 이용하는 이미지 잡음제거 처리과정이다.² 이 과정을 통해 이미지에서 불필요하게 문자로 오인될 수 있는 노이즈와 의미없는 디테일을 블러 처리하게 된다.

이 밖의 잡음제거 기법으로 Total variation denoising 방법도 있다. 이 방법은 총 변량의 최소화를 통해 원래의 이미지에 근접한 값을 찾을 수 있게 한다. 이 노이즈 제거 방법은 기존의 이미지 잡음제거 방법보다 더욱 예리한 윤곽을 원본과 같이 유지할 수 있게 하는 특징이 있다.³

● 문자영역 분할



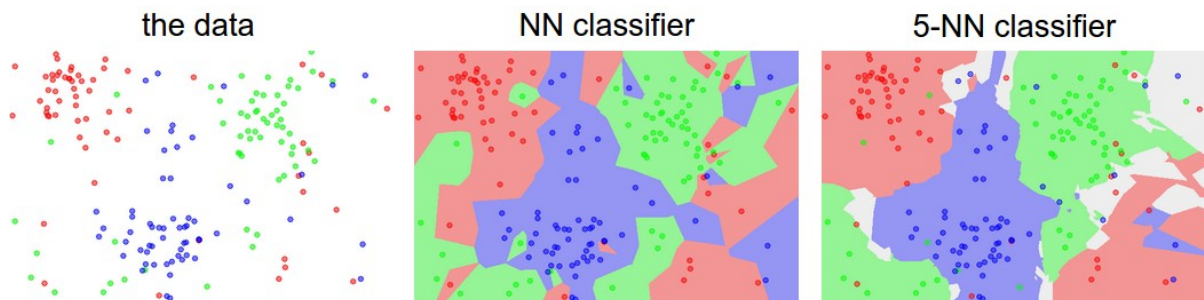
² Lin, Hsin-Chih, Ling-Ling Wang, and Shi-Nine Yang. "Automatic determination of the spread parameter in Gaussian smoothing." *Pattern Recognition Letters* 17.12 (1996): 1247-1252.

³ Strong, David, and Tony Chan. "Edge-preserving and scale-dependent properties of total variation regularization." *Inverse problems* 19.6 (2003): S165.

문자인식에 있어서 문자부분과 문자 외 영역의 분할은, 필요한 정보가 존재하는 영역을 구분 짓게 하여 문자의 인식률을 높일 수 있게 한다. 이러한 방법 중에서 우리는 이미지의 대비를 높여 흑, 백의 두 영역으로 나누는 이진화를 이용하였다.

특히 이번 프로젝트에선 Adaptive Thresholding 기법을 사용하였는데, 이 방법은 기존 Simple Threshold에서 오브젝트를 이진화 할 임계점을 하나만 설정한 경우에 영상의 지역별 명암 차이로 인해 영상의 일부가 제대로 식별되지 않는 문제점을 해결하기 위해 각 지역 별 임계점을 가변적으로 변화시켜 이미지의 명암비가 일정하지 않은 경우에도 robust한 이진화 성능을 보장하는 기법이다. ⁴

2.1.3 KNN(K Nearest Neighbor) 알고리즘



위 이미지는 3 가지 색 분류로 라벨링된 학습용 데이터 포인트들을 가정하고 NN 분류, 5-NN 분류를 이용하여 각 색깔로 분류되는 영역을 나타낸 사진이다. 가장 왼쪽의 이미지는 학습용 데이터포인트들을 나타낸 것이고, 두번째와 세번째 이미지는 가장 가까운 데이터 포인트만으로 분류를 수행하는 NN-알고리즘과 K 개의 최근접 포인트를 기반으로 분류하는 KNN(k=5)의 decision boundaries 를 표현한 플롯이다. NN-알고리즘의 경우 일부 outlier 들에 의해 분류 영역에 작은 island 들이 생기지만, KNN 알고리즘의 경우 각 분류 영역의 경계가 훨씬 부드러워지는것을 관찰할 수 있다.⁵

⁴ Bradley, Derek, and Gerhard Roth. "Adaptive thresholding using the integral image." *Journal of graphics, gpu, and game tools* 12.2 (2007): 13-21.

⁵ Li, Fei-Fei, and Andrej Karpathy. "CS231n: Convolutional Neural Networks for Visual Recognition." (2015).

이미지 전처리와 문자 추정 오브젝트 분류를 마친 후 마지막으로 문자 분류 단계인데, 여기서 각각의 문자로 추정된 오브젝트들이 실제로 어느 문자인지 가려내는데 쓰이는 기계학습 알고리즘이 이 KNN 이다.

● 표 1 KNN 알고리즘의 특징

특징	설명
최고인접 다수결 유사도(거리) 기반 Lazy Learning 기법	기존 데이터 중 가장 유사한 K개의 데이터를 측정하여 분류 유클리디언 거리, 마할라노비스 거리, 코사인 유사도 등 활용 새로운 입력 값이 들어온 후 분류 시작
단순유연성	모형이 단순하며 파라미터의 가정이 거의 없음

이 알고리즘에선 우선 인식하고자 하는 대상을 벡터화하여 미리 학습용으로 저장된 학습 데이터들의 벡터들과 유클리드 거리를 계산, 그 거리가 가장 짧은 근처 벡터 K개를 추려낸다. 이 다음 그 최근접 이웃 벡터들에 맵핑된 실제 대상들의 비율을 기반으로 어느 대상으로 분류할지를 판단하게 된다. 여기서는 미리 실제 문자와 맵핑된 문자 이미지들을 학습시키고, 입력된 문자 이미지가 어느 문자의 이미지들과 가장 근접한 유클리드 거리를 가지는지 테스트하여 문자의 종류를 가려낸다.

● 표 2 KNN 알고리즘의 장점 및 단점

장점	설명
학습이 간단	- 모형이 단순하고 구현이 쉬움
유연한 경계	- 모수(parameter) 및 데이터에 대한 가정이 거의 없음 - 거리의 변형, 가중치 적용이 용이함
모델의 유연성	- 유클리디언, 코사인유사도, 가중치 적용, 정규화 적용 용이 - 데이터에 대한 가정을 반영하여 변형하기에 간편
높은 정확도	- 변형한 데이터의 training data set 기반 분류기 검증 용이 - 사례기반(instance based)으로 높은 정확성 - 훈련 데이터가 클수록 클러스터 매칭의 정확성 높아짐
단점	설명
K 선정 어려움	- K 수에 따라 알고리즘의 성능을 좌우

	- under/over fitting의 trade off 문제 발생 요인
공간 예측 부정확	- 공간 정보 예측모델에서는 특정 이벤트의 발생이 일정하지 않고, 영향 변수 많아 적용이 어려움
거리계산 복잡성	- 모든 데이터와의 유사도, 거리 측정 수행 필요 - 명목변수 및 결측치를 따로 처리 필요 - 기존데이터의 실측값, instance에 크게 의존
고비용	- 모든 데이터를 메모리 기반 연산, 거리 측정 필요 - 데이터가 커질수록 메모리 및 연산시간 증가 문제
노이즈에 약함	- 노이즈로 인해 큰 K 설정을 필요로 함 - 민감하고 작은 데이터가 무시되는 under fitting 문제 야기

● KNN 알고리즘 선정 이유

1. 문자인식률을 높이는 방향으로 진행하여 높은 정확성을 보장하며 노이즈에 Robust 하게 대처 할 수 있는 알고리즘 일 것
2. instance based learning 으로 generalized model 을 생성할 필요 없이 training set 만을 가지고 기계학습을 구현 할 수 있을 것
3. 벡터 연산과 같이 반복적인 산술 연산이 많이 필요한 알고리즘으로 최적화의 필요성이 클 것.

위와 같은 조건을 고려하여 KNN 알고리즘을 사용하였다.

2.2 프로그램 개요

아래에선 프로그램을 구성하는 각 함수들의 기능을 설명한다. 보고서에 옮기기엔 코드 길이가 길거나 크게 중요하지 않은 경우 아래 설명에선 제외하고 소스는 별첨하였다.

2.2.1 외부 모듈

```
import sys
import numpy as np
import cv2
from math import sqrt
import time
```

위는 해당 프로그램에 필요한 각종 외부 모듈을 임포트하는 부분이다. 이미지 프로세싱에는 OpenCV 를 사용하였으며 이미지 자료의 저장과 각종 벡터 연산을 위해 NumPy 또한 활용하였다. 사용된 파이썬 버전은 3.5.1 이다.

2.2.2 함수 기능 명세

- Image2vector_list

이미지 파일 경로를 입력받아 해당 이미지를 로드한 후 이미지 전처리를 하고, 전처리가 완료된 이미지에서 오브젝트로 추정되는 부분의 좌표들을 리스트로 모아 반환한다.

```
image_bw = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_blur = cv2.GaussianBlur(image_bw, (5, 5), 0)
image_threshold = cv2.adaptiveThreshold(image_blur, 255, 1, 1, 11, 2)
```

위 이미지는 해당 함수에서 이미지 전처리를 수행하는 부분이다. 첫번째 라인에서 이미지를 흑백처리한 다음, 두번째 라인에서 Gaussian Smoothing 으로 노이즈를 줄이고 마지막 세번째 라인에서 영상 이진화를 실시하여 이미지를 1차원 벡터로 변환할 준비를 마친다.

```
_, contours, hierarchy = cv2.findContours(cleaned, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
```

위 라인에서는 OpenCV의 findContours 메서드를 이용하여 이미지에서 오브젝트를 탐지한다. findContours 메서드는 border following algorithm⁶을 사용하여 주변지역과 분리되는 오브젝트를 구분하여 그 위치정보 객체를 contours 변수에 반환한다.

- make_file_path_list

각 문자(알파벳 대문자 A~Z)들의 training set 이미지 파일에 대해 적절한 경로를 생성한다.

- make_train_set

⁶ Suzuki, Satoshi. "Topological structural analysis of digitized binary images by border following." Computer Vision, Graphics, and Image Processing 30.1 (1985): 32-46.

make_file_path_list에서 생성된 training set 경로에 대해 각 문자들의 벡터들로 이루어진 매트릭스를 생성한다.

- **most_common**

입력받은 리스트에서 가장 빈번히 존재하는 요소를 반환한다.

- **get_euclid**

```
def get_euclid(trained_char, input_char_vector):
    ssq = 0
    for i, px in enumerate(input_char_vector): # 매 픽셀마다 거리 계산해서 ssq 합산
        ssq += (abs(px-trained_char[0][i]))**2

    return sqrt(ssq)
```

학습된 글자와 입력된 vector를 가지고 각 픽셀마다 거리를 계산하여 *sum of square* 를 합산한 후 두 벡터 간 유클리드 거리인 $\sqrt{\text{sum of square}}$ 를 반환한다.

- **ocr**

```
def ocr(file_path, trained_list, k, SHOW_PROGRESS=False):
    input_vectors = image2vector_list(file_path, SHOW_LOOP=False, SHOW_RECOGNIZED=False).tolist()

    result_str = ""

    for input_char_vector in input_vectors: # 추출된 각 문자마다
        proximate_k = [] # [{"A", 10}, {"A", 20}, ...]
        for trained_char in trained_list: # 모든 학습셋 문자들에 대해서 비교
            euclid = get_euclid(trained_char, input_char_vector)

            if len(proximate_k) < k:
                proximate_k.append([trained_char[-1], euclid])
            else:
                proximate_k = sorted(proximate_k, key=lambda x:x[1])
                proximate_k[-1] = [trained_char[-1], euclid]

        votes = [x[0] for x in proximate_k]
        elect = most_common(votes)
        if SHOW_PROGRESS:
            print(elect)
        result_str += elect

    return result_str
```

문자인식을 수행할 이미지 경로를 받아 추출된 문자들을 문자열로 변환하여 반환하는 부분이다. 여기에서 벡터간 거리를 계산하는 함수 `get_euclid`가 호출되어 입력된 이미지의 벡터와 기존의 학습셋 벡터들간의 유클리드 거리를 비교하여 문자인식을 수행한다.

- `If __name__ == "main": (...)`

해당 파이썬 스크립트가 실행된 경우 실제 파일경로와 함께 위 함수들을 호출하여 결과를 측정된 프로세스타임과 함께 출력한다

2.2.3 Training set



학습 데이터는 다음과 같은 이미지 파일 26개를 사용하였다. 사진은 A에 해당되는 학습 데이터용 이미지이고 실제 소스에는 A~Z까지 모두 이런 이미지 파일이 있으며, 이를 기반으로 학습한다.

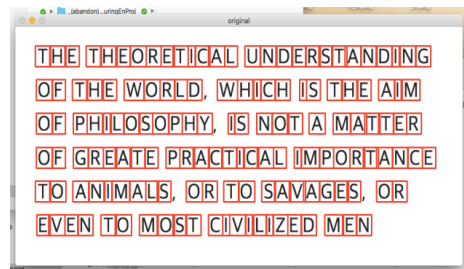
2.3 프로그램 수행 결과

다음과 같이 버트런드 러셀의 글귀를 캡처한 이미지를 대상으로 프로그램을 실행해 보았다.

【 OCR 테스트를 위해 사용한 이미지 】

THE THEORETICAL UNDERSTANDING
OF THE WORLD, WHICH IS THE AIM
OF PHILOSOPHY, IS NOT A MATTER
OF GREATE PRACTICAL IMPORTANCE
TO ANIMALS, OR TO SAVAGES, OR
EVEN TO MOST CIVILIZED MEN

【 문자로 추정되는 오브젝트 위치가 추출된 결과 】



【 문자 인식이 완료되어 문자열이 출력된 화면 】

```
E
N
THE THEORETICAL UNDERSTANDING OF THE WORLD WHICH IS THE AIM OF PHILOSOPHY IS NOT A MATTER OF GREATE PRACTICAL IMPORTANCE TO ANIMALS OR TO SAVAGES OR EVEN TO MOST CIVILIZED MEN
process time: 45.967560999999996
Process finished with exit code 0
```

2.4 성능 측정

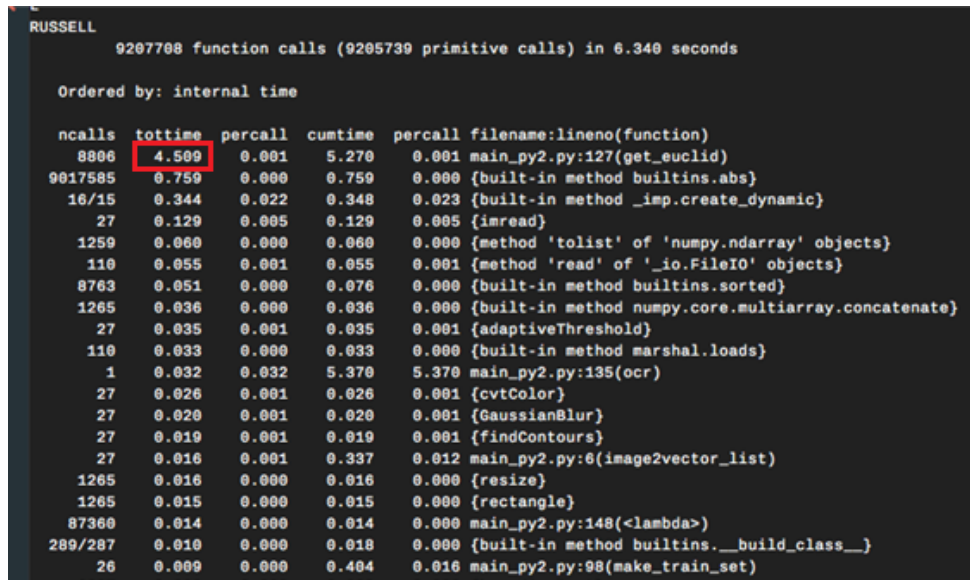
2.4.1 성능 측정에 사용된 이미지

RUSSELL

기존의 OCR 테스트에서 사용한 이미지를 이용하여 프로그램의 성능을 측정 했을 때, 문자인식 수행은 정상적으로 이뤄지나, 연속적인 테스트를 수행하기에는 시간이 너무 오래 걸리는 관계로 문자의 수가 더 적은 이미지로 성능 측정을 시도하고, 이를 개선해 보았다.

2.4.2 프로파일링 수행결과

【 프로파일링 결과 】



```
RUSSELL
9207708 function calls (9205739 primitive calls) in 6.348 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
8806    4.509    0.001    5.270    0.001 main_py2.py:127(get_euclid)
9017585 0.759    0.000    0.759    0.000 {built-in method builtins.abs}
16/15   0.344    0.022    0.348    0.023 {built-in method _imp.create_dynamic}
27      0.129    0.005    0.129    0.005 {imread}
1259    0.060    0.000    0.060    0.000 {method 'tolist' of 'numpy.ndarray' objects}
110     0.055    0.001    0.055    0.001 {method 'read' of '_io.FileIO' objects}
8763    0.051    0.000    0.076    0.000 {built-in method builtins.sorted}
1265    0.036    0.000    0.036    0.000 {built-in method numpy.core.multiarray.concatenate}
27      0.035    0.001    0.035    0.001 {adaptiveThreshold}
110     0.033    0.000    0.033    0.000 {built-in method marshal.loads}
1       0.032    0.032    5.370    5.370 main_py2.py:135(ocr)
27      0.026    0.001    0.026    0.001 {cvtColor}
27      0.020    0.001    0.020    0.001 {GaussianBlur}
27      0.019    0.001    0.019    0.001 {findContours}
27      0.016    0.001    0.337    0.012 main_py2.py:6(image2vector_list)
1265    0.016    0.000    0.016    0.000 {resize}
1265    0.015    0.000    0.015    0.000 {rectangle}
87360   0.014    0.000    0.014    0.000 main_py2.py:148(<lambda>)
289/287 0.010    0.000    0.018    0.000 {built-in method builtins.__build_class__}
26      0.009    0.000    0.404    0.016 main_py2.py:98(make_train_set)
```

(각 칼럼별 명세)

- ncalls: 해당 루틴(루틴이란 함수 혹은 함수와 유사한 역할을 하는 논리 조각을 의미)이 호출된 횟수
- tottime: 해당 루틴의 실행시간에서 서브루틴이 차지하는 부분을 제외한 순수한 실행시간을 모두 합친 것
- percall(첫번째): 해당 루틴이 한번 실행될 때 소요되는 평균 시간(서브루틴 제외 순수 시간) (tottime/ncalls)
- cumtime: 특정 루틴의 서브루틴을 제외하지 않은 소요시간을 모두 합친 것
- percall(두번째): 해당 루틴이 한번 실행될 때 소요되는 평균 시간(서브루틴 제외하지 않은 시간) (cumtime/ncalls)
- filename:lineno(function): 괄호 안에서 해당 함수 이름을 확인.

프로그램 내의 병목구간은 순수 실행시간을 나타내는 tottime 항목을 기준으로 판단하였다. 이에 해당 프로그램의 병목구간은 get_euclid 함수임을 알 수 있었다. 순수 실행시간이 4.509초로 가장 길었고 다른 함수들의 실행 시간은 각 0.8초미만으로, 병목구간에 비해 매우 작다고 간주할 수 있기 때문이다. 따라서 프로그램의 최적화는 get_euclid 함수의 성능을 개선하는 쪽으로 수행하였다.

2.4.3 수행시간 측정

프로그램의 개선 후 성능과의 정확한 비교를 위해서, 개선하기 전 사이클 타임을 5회 측정하고 평균값을 구했다. 사이클 타임 측정은 파이썬에서 표준 라이브러리로 제공되는 time.process_time() 을 활용하였다. 해당 메서드를 활용한 이유는 프로세스의 sleep time을 제외하고 순수 해당 프로그램의 실행시간만을 측정하기 때문에 비교적

정확한 CPU Time을 알 수 있기 때문이다.⁷

【 개선하기 전 사이클 타임 5회 측정 결과 】

```
L
RUSSELL
process time: 4.027355
Process finished with exit code 0

L
RUSSELL
process time: 4.016128
Process finished with exit code 0

L
RUSSELL
process time: 4.010098999999999
Process finished with exit code 0

L
RUSSELL
process time: 3.956762
Process finished with exit code 0

L
RUSSELL
process time: 4.024786
Process finished with exit code 0
```

5번의 평균으로 약 4.0070262초 가량의 process time이 소요되었다.

2.5 성능 최적화

2.5.1 병목구간 선정

【 get_euclid 함수 】

```
def get_euclid(trained_char, input_char_vec):
    ssq = 0
    for i, px in enumerate(input_char_vector):
        ssq += (abs(px - trained_char[0][i]))**2
    return sqrt(ssq)
```

앞서 확인한 것과 같이 병목구간은 유클리드 거리를 계산하는 함수인 get_euclid 임을 알 수 있다. 따라서 해당 부분의 process time 을 개선하기로 결정했다.

⁷ Python Software Foundation. Python Language Reference, version 3.5.1 Available at https://docs.python.org/3/library/time.html#time.process_time

2.5.2 1차 개선

【 1차 개선된 get_euclid 함수 】

```
def get_euclid(trained_char, input_char_vector):  
    ssq = 0  
    for i, px in enumerate(input_char_vector): #  
        ssq += (abs(px - trained_char[0][i])) # 최적화  
    return sqrt(ssq)
```

원래 유클리드 거리 계산은 벡터의 인덱스가 같은 각 원소간 거리의 제곱합을 이용한다. 하지만 여기서는 이미지가 이진화 되어 각 픽셀, 즉 각 원소의 가능한 값이 1 과 0 밖에 없으므로 같은 인덱스의 각 원소간 거리 또한 0 혹은 1 이 되기 때문에 굳이 제곱할 필요가 없다. 따라서 위 함수를 다음과 같이 수정하였다.

2.5.3 2차 개선

【 Profiling 수행 결과 2 】

```
RUSSELL  
process time: 4.454815  
9207711 function calls (9205742 primitive calls) in 5.111 seconds  
  
Ordered by: internal time  
  
ncalls  tottime  percall  cumtime  percall  filename:lineno(function)  
8806    3.318    0.000    4.072    0.000  main_py2.py:128(get_euclid)  
9017585 0.752    0.000    0.752    0.000  {built-in method builtins.abs}  
16/15   0.340    0.021    0.344    0.023  {built-in method _imp.create_dynamic}  
27      0.126    0.005    0.126    0.005  {imread}  
110     0.057    0.001    0.057    0.001  {method 'read' of '_io.FileIO' objects}  
1259    0.049    0.000    0.049    0.000  {method 'tolist' of 'numpy.ndarray' objects}  
8763    0.045    0.000    0.070    0.000  {built-in method builtins.sorted}  
27      0.037    0.001    0.037    0.001  {adaptiveThreshold}  
110     0.034    0.000    0.034    0.000  {built-in method marshal.loads}  
1265    0.031    0.000    0.031    0.000  {built-in method numpy.core.multiarray.concatenate}  
27      0.028    0.001    0.028    0.001  {cvtColor}  
1       0.027    0.027    4.161    4.161  main_py2.py:136(ocr)  
27      0.018    0.001    0.018    0.001  {findContours}  
27      0.017    0.001    0.017    0.001  {GaussianBlur}  
87360   0.014    0.000    0.014    0.000  main_py2.py:149(<lambda>)  
1265    0.014    0.000    0.014    0.000  {rectangle}  
27      0.014    0.001    0.321    0.012  main_py2.py:7(image2vector_list)  
1265    0.013    0.000    0.013    0.000  {resize}
```

【 1차 개선 후 사이클 타임 5회 측정 결과 】

```
L
RUSSELL
process time: 2.849538
Process finished with exit code 0

L
RUSSELL
process time: 2.9342880000000005
Process finished with exit code 0

L
RUSSELL
process time: 3.039354
Process finished with exit code 0
```

```
RUSSELL
process time: 2.9433170000000004
Process finished with exit code 0

L
RUSSELL
process time: 2.882231
Process finished with exit code 0
```

5 번의 수행 결과의 평균값으로 약 2.9297456 초 가량의 process time 이 소요되었다



개선 후 5 회의 사이클타임 평균이 4.0070262 초에서 2.9297456 초로 감소했다. 이것은 $(4.0070262 - 2.9297456) / 4.0070262 * 100 = 26.8848 \%$ 성능 증가를 가져왔다.

```
def get_euclid(trained_char, input_char_vector):
    ssq = 0
    for i, px in enumerate(input_char_vector): #
        ssq += (abs(px - trained_char[0][i])) # 최적화
    return sqrt(ssq)
```

Profiling 결과 여전히 get_euclid 함수가 가장 많은 프로세스 타임을 소모하고 있기 때문에 계속해서 해당 부분에 대한 개선을 수행했다.

【 개선 전 】

```
for trained_char in trained_list: # 모든 학습셋 문자열에 대해서
    euclid = get_euclid(trained_char, input_char_vector)
```

get_euclid 함수의 2차 개선 전 호출 방식이다. trained_char 리스트를 그대로 인자에 넘겨주고 있다.

```
def get_euclid(trained_char, input_char_vector):
    ssq = 0
    for i, px in enumerate(input_char_vector): # 매 픽셀
        ssq += (abs(px - trained_char[0][i])) # 최적화
    return sqrt(ssq)
```

trained_char 리스트의 첫 번째 원소만 필요하지만 매 루프마다 trained_char 리스트의 전체가 취급되고 있음을 확인할 수 있다.

【 개선 후 】

```
for trained_char in trained_list: # 모든 학습셋 문자열에 대해서
    euclid = get_euclid(trained_char[0], input_char_vector)
```

get_euclid 함수의 2차 개선 후 호출 방식이다. trained_char 리스트의 첫 번째 원소만 인자에 넘겨주고 있다

```
def get_euclid(trained_char_vec, input_char_vector):
    ssq = 0
    for i, px in enumerate(input_char_vector): # 매 픽셀
        ssq += (abs(px - trained_char_vec[i])) # 최적화
    return sqrt(ssq)
```

get_euclid 함수의 2차 개선 후 모습이다. trained_char 리스트에서 해당 함수에서 필요로 하는 첫 번째 원소만을 trained_char_vec로 받아 취급한다.

2.5.4 개선 후 성능

【 Profiling 수행 결과 3 】

```
RUSSELL
process time: 4.086282000000001
9206479 function calls (9204510 primitive calls) in 4.768 seconds

Ordered by: internal time
```

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
8806	2.954	0.000	3.685	0.000	main_py2.py:129(get_euclid)
9017585	0.729	0.000	0.729	0.000	{built-in method builtins.abs}
16/15	0.359	0.022	0.365	0.024	{built-in method _imp.create_dynamic}
27	0.132	0.005	0.132	0.005	{imread}
110	0.057	0.001	0.057	0.001	{method 'read' of '_io.FileIO' objects}
27	0.054	0.002	0.054	0.002	{method 'tolist' of 'numpy.ndarray' objects}
8763	0.042	0.000	0.066	0.000	{built-in method builtins.sorted}
27	0.037	0.001	0.037	0.001	{adaptiveThreshold}
1265	0.035	0.000	0.035	0.000	{built-in method numpy.core.multiarray.concatenate}
110	0.033	0.000	0.033	0.000	{built-in method marshal.loads}
27	0.030	0.001	0.030	0.001	{cvtColor}
1	0.025	0.025	3.768	3.768	main_py2.py:136(ocr)
27	0.020	0.001	0.020	0.001	{findContours}
27	0.020	0.001	0.020	0.001	{GaussianBlur}
27	0.016	0.001	0.343	0.013	main_py2.py:7(image2vector_list)
26	0.015	0.001	0.410	0.016	main_py2.py:99(make_train_set)

【 2차 개선 후 사이클 타임 5회 측정 결과 】

```
L
RUSSELL
process time: 2.585824

Process finished with exit code 0
```

```
L
RUSSELL
process time: 2.702329

Process finished with exit code 0
```

```
L
RUSSELL
process time: 2.689236

Process finished with exit code 0
```

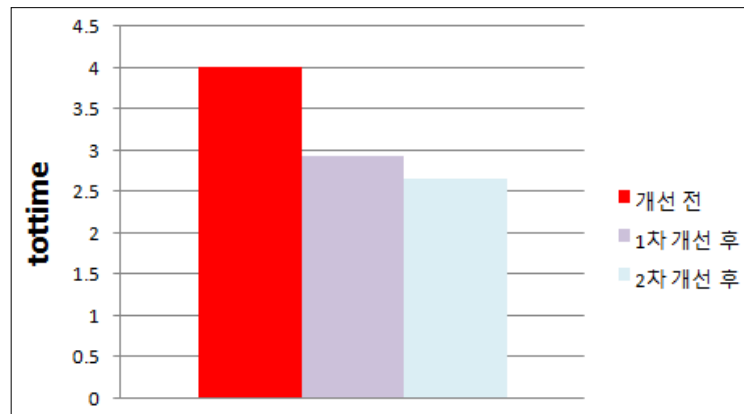
```
L
RUSSELL
process time: 2.7347470000000005

Process finished with exit code 0
```

```
L
RUSSELL
process time: 2.631445

Process finished with exit code 0
```

5 번의 수행 결과의 평균값으로 약 2,6687162 초 가량의 process time 이 소요되었다.



2차 개선 후 5회의 사이클타임 평균이 2.9297456 초에서 2.6687162 초로
 $(2.9297456 - 2.6687162) / 2.9297456 * 100 = 8.91 \%$ 성능 증가를 가져왔다.

결과적으로 process time 이 두 번의 개선 끝에 4.0070262 초에서 2.6687162 초로 감소했고, $(4.0070262 - 2.6687162) / 4.0070262 * 100 = 33.40 \%$ 의 성능증가를 보였다.

3. 결론

4. 별첨-소스코드

```

1 import sys
2 import numpy as np
3 import cv2
4 from math import sqrt
5 import time
6
7 def image2vector_list(file_name,
8                       RESOLUTION=32,
9                       AREA_SIZE=10,
10                      CHAR_HEIGHT=10,
11                      MARGIN=5,
12                      X_SCALE=1,
13                      Y_SCALE=15,
14                      SHOW_LOOP=False,

```

```

15             SHOW_RECOGNIZED=False,
16             SHOW_VECTORS=False):
17
18         image = cv2.imread(file_name)
19
20         ##### 전처리 시작
21
22         image_bw = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
23         image_blur = cv2.GaussianBlur(image_bw, (5, 5), 0)
24         image_threshold = cv2.adaptiveThreshold(image_blur, 255, 1
25 , 1, 11, 2)
26         cleaned = image_threshold.copy()
27
28         ##### 전처리 끝
29
30         ##### 오브젝트 탐색
31         _, contours, hierarchy = cv2.findContours(cleaned, cv2.RETR
32 _LIST, cv2.CHAIN_APPROX_SIMPLE)
33
34         countours_filtered = filter(lambda contour:cv2.contourArea(
35 contour) > AREA_SIZE and cv2.boundingRect(contour)[3] > CHAR_HE
36 IGHT, contours)
37         countours_sorted = sorted(countours_filtered, key=lambda co
38 ntour:cv2.boundingRect(contour)[0]*X_SCALE + cv2.boundingRect(c
39 ontour)[1]*Y_SCALE, reverse=False)
40
41         vectors = np.empty((0, RESOLUTION**2))
42         keys = [i for i in range(48,58)]
43
44         ex=[0,0,0,0]
45         count = 0
46         for index, contour in enumerate(countours_sorted):
47             [x, y, w, h] = cv2.boundingRect(contour) # x,y: top-
48 left point w, h: with, height
49             [x_ex, y_ex, w_ex, h_ex] = ex
50             nested = x-x_ex >= 0 and y-y_ex>=0 and (x+w)-
51 (x_ex+w_ex) <= 0 and (y+h)-(y_ex+h_ex) <= 0
52             go_back = x-x_ex < 0 and abs(y-y_ex) < h*0.8
53             # nested = False
54
55             if not nested and not go_back:

```

```

49         cv2.rectangle(image, (x - MARGIN, y -
MARGIN), (x + w + MARGIN, y + h + MARGIN), (0, 0, 255), 2)
50         roi = cleaned[y - MARGIN:y + h + MARGIN, x -
MARGIN:x + w + MARGIN]
51         roismall = cv2.resize(roi, (RESOLUTION, RESOLUTION))
52
53         sample = roismall.reshape((1, RESOLUTION**2)).astype
e(bool) # roismall 32*32 => 1*1024
54         vectors = np.append(vectors, sample, 0)
55
56         if SHOW_LOOP:
57             cv2.imshow('norm', image)
58             key = cv2.waitKey(0)
59
60             if key == 27: # (escape to quit)
61                 sys.exit()
62             elif key in keys:
63                 sample = roismall.reshape((1,100))
64                 samples = np.append(samples, sample, 0)
65
66         ex = [x, y, w, h]
67         count += 1
68
69
70     if SHOW_RECOGNIZED:
71         cv2.imshow('original', image)
72         print("print esc to continue")
73         if 27 == cv2.waitKey(0):
74             pass
75
76     if SHOW_VECTORS == True:
77         for sample in vectors:
78             print("=====")
79             for i in range(RESOLUTION):
80                 for j in range(RESOLUTION):
81                     if sample[i*RESOLUTION+j] == True:
82                         print("#", end="")
83                     else:
84                         print(" ", end="")
85
86             print("")
87
88     return vectors

```

```

89
90
91 def make_file_path_list(capitals):
92     result = []
93     for capital in capitals:
94         result.append("train_image/" + capital + ".png")
95     return result
96
97
98 def make_train_set(file_path, flag):
99     vector_list = image2vector_list(file_path,
100                                     Y_SCALE=9,
101                                     AREA_SIZE=12,
102                                     CHAR_HEIGHT=12,
103                                     SHOW_LOOP=False,
104                                     SHOW_RECOGNIZED=False,
105                                     SHOW_VECTORS=False,
106                                     RESOLUTION=32)
107
108     result = []
109     vector_list = vector_list.tolist()
110     for vector in vector_list:
111         result.append([vector, flag])
112
113     return result
114
115
116 def most_common(input_list):
117     elems = tuple(input_list)
118     biggest_count = 0
119     most_common_elem = ""
120     for elem in elems:
121         count = input_list.count(elem)
122         if count > biggest_count:
123             biggest_count = count
124             most_common_elem = elem
125     return most_common_elem
126
127
128 def get_euclid(trained_char_vec, input_char_vector):
129     ssq = 0
130     for i, px in enumerate(input_char_vector):
131         ssq += (abs(px-trained_char_vec[i]))

```

```

132     return sqrt(ssq)
133
134
135 def ocr(file_path, trained_list, k, SHOW_PROGRESS=False):
136     input_vectors = image2vector_list(file_path, SHOW_LOOP=False,
137     SHOW_RECOGNIZED=False).tolist()
138
139     result_str = ""
140
141     for input_char_vector in input_vectors:
142         proximate_k = [] # [{"A", 10}, {"A", 20}, ...]
143         for trained_char in trained_list:
144             euclid = get_euclid(trained_char[0], input_char_vector)
145
146             if len(proximate_k) < k:
147                 proximate_k.append([trained_char[-1], euclid])
148             else:
149                 proximate_k = sorted(proximate_k, key=lambda x:
150                 x[1])
151                 proximate_k[-1] = [trained_char[-1], euclid]
152
153             votes = [x[0] for x in proximate_k]
154             elect = most_common(votes)
155             if SHOW_PROGRESS:
156                 print(elect)
157             result_str += elect
158
159     return result_str
160
161 if __name__ == '__main__':
162     import matplotlib.pyplot as pp
163     image_file = 'russell.png'
164
165     chars = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J",
166     "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W",
167     "X", "Y", "Z"]
168     training_file_path_list = make_file_path_list(chars)
169     trained = []
170     for index, path in enumerate(training_file_path_list):
171         trained.extend(make_train_set(path, chars[index]))

```

```

169
170
171     result_list = []
172     count = 100
173     for i in range(count):
174         print("index: " + str(i+1))
175         t1 = time.process_time()
176         result = ocr("russell_short.png", trained, 10, SHOW_PRO
GRESS=False)
177         print(result)
178         t2 = time.process_time()
179         process_time = t2 - t1
180         print("process time: " + str(process_time))
181         result_list.append({"str": result, "time": process_time
    })
182
183     result_list_sorted = sorted(result_list, key=lambda k: k['t
ime'])
184
185     for item in result_list_sorted:
186         print(item)
187
188     val = 0.
189     ar = np.array([x['time'] for x in result_list_sorted])
190     pp.plot(ar, np.zeros_like(ar) + val, 'x')
191     pp.show()

```

5. 참조

- 1) Goldberg, Emanuel. "Statistical machine." U.S. Patent No. 1,838,389. 29 Dec. 1931.
- 2) Lin, Hsin-Chih, Ling-Ling Wang, and Shi-Nine Yang. "Automatic determination of the spread parameter in Gaussian smoothing." Pattern Recognition Letters 17.12 (1996): 1247-1252.
- 3) Strong, David, and Tony Chan. "Edge-preserving and

scale-dependent properties of total variation regularization." *Inverse problems* 19.6 (2003): S165.

- 4) Bradley, Derek, and Gerhard Roth. "Adaptive thresholding using the integral image." *Journal of graphics, gpu, and game tools* 12.2 (2007): 13–21.
- 5) Li, Fei-Fei, and Andrej Karpathy. "CS231n: Convolutional Neural Networks for Visual Recognition." (2015).
- 6) Suzuki, Satoshi. "Topological structural analysis of digitized binary images by border following." *Computer Vision, Graphics, and Image Processing* 30.1 (1985): 32–46.
- 7) Python Software Foundation. Python Language Reference, version 3.5.1 Available at https://docs.python.org/3/library/time.html#time.process_time