

딥러닝 및 머신러닝을 이용한 이미지 속 음식 칼로리 계산 시스템



저자1 201424460 박정민
저자2 201524402 강나훈
저자3 201524501 예병준

지도교수 감진규

목 차

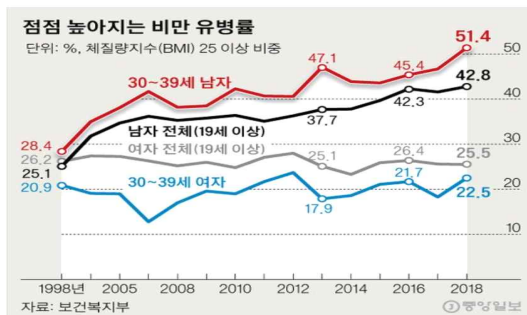
1. 서론	3
1.1. 연구 배경	3
1.2. 기존 문제점	3
1.3. 연구 목표	4
1.3.1. 세부 목표	5
1.4. 요구 조건 분석	5
1.5. 현실적 제약사항	6
2. 배경 지식	7
2.1. openCV	7
2.1.1. 주요함수	7
2.2. SVM	9
2.3. CNN	9
2.4. Flask	11
3. 개발 내용	12
3.1. 설계도	12
3.2. 트레이닝셋 구축	13
3.3. 음식 이미지 전처리	13
3.4. 음식 이미지 특징 추출	21
3.4.1. 음식 색상 추출	21
3.4.2. 음식 질감 추출	21
3.4.3. 음식 모양 추출	23
3.5. 학습	24
3.5.1. 머신러닝(SVM) 모델 학습	24
3.5.2. 딥러닝(CNN) 모델 학습	30
3.6. 칼로리 계산	34
3.7. 칼로리 출력	38
3.8. 영양성분 초과 알림 (산업체 멘토링 반영)	41
4. 과제 수행 결과 및 분석	42
4.1. 개발 환경	42

4.2. 개발 결과.....	42
5. 결론 및 향후 개선 방향.....	46
5.1. 활용 방안.....	46
5.2. 한계점.....	46
5.3. 향후 개선 방향.....	47
6. 역할 분담 및 개발 일정.....	48
6.1. 역할 분담.....	48
6.2. 개발 일정.....	49
7. 참고 문헌.....	50

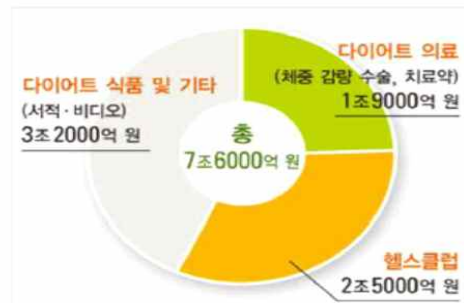
1. 서론

1.1. 연구 배경

과체중이나 비만으로 고민이 많은 사람들이 증가하고 있다. 이는 개인의 문제이기도 하지만 심각한 사회문제 중 하나이기도 하다. 따라서 과체중이나 비만을 해결하기 위한 다이어트는 현대인들에게 필수적인 부분이라고 할 수 있다. 또한 단순히 체중을 줄이기 위한 목적이 아닌 멋진 몸매를 가지기 위한 수단으로도 사용된다. 이처럼 다이어트의 시장 규모가 커지고 있고 관심도가 증가하고 있다. 이에 맞추어 다이어트의 가장 기본이라고 할 수 있는 칼로리에 대해 사용자가 음식에 대한 사진을 찍는 것만으로 음식의 칼로리를 쉽게 알 수 있게 하는 프로그램을 기획하였다.



[그림 1] 비만 비율 도표(중앙일보 자료)



[그림 2] 국내 다이어트 시장규모(폴무원 자료)

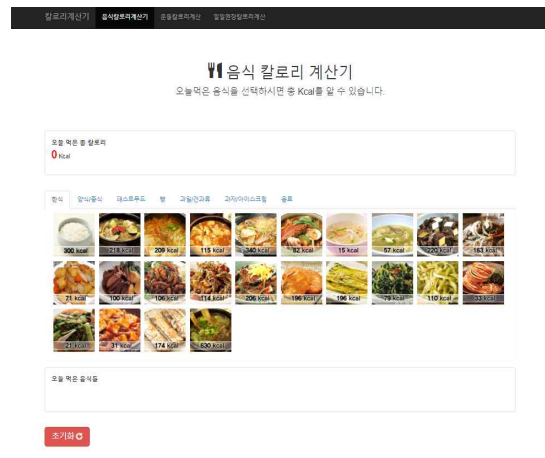
본 프로젝트는 사진에 있는 음식들에 대해 영상인식을 이용하여 칼로리를 계산하는 것을 목표로 한다. 음식사진의 영상특징을 이용해서 음식의 종류나 양을 추정하고 이에 기반 하여 칼로리를 계산해서 사용자가 칼로리를 쉽게 알 수 있게 하여 본인의 식습관을 효율적으로 관리 할 수 있을 것으로 기대된다.

1.2. 기존 문제점

현재 음식의 칼로리를 알기 위해서는 제품의 포장지에 기재된 칼로리 수치를 확인하거나 음식에 대한 칼로리 정보를 인터넷에서 검색하는 방법으로 칼로리를 확인 할 수 있다.



[그림 3] 기존의 칼로리 계산 프로그램 1



[그림 4] 기존의 칼로리 계산 프로그램 2

마찬가지로 현재 존재하는 칼로리 관련 프로그램들은 사용자가 본인이 직접 섭취할 음식들에 대한 정보를 입력 하고 입력한 음식들에 대한 칼로리를 계산 해주는 방식이거나 날짜별로 섭취한 음식들에 대한 칼로리를 기록하여 보관하는 형태가 대부분이다. 이러한 방식은 사용자가 일일이 정보를 입력해야하고, 무슨 음식인지 모를 경우 칼로리를 알 수 없다. 그리고 대부분 칼로리를 출력할 때 100g 당, 1인분 당 몇 칼로리 혹은 1회 적정섭취량 당 몇 칼로리와 같은 형태로 칼로리를 알려주는데 실제로 사용자가 섭취할 양에 대한 정확한 칼로리 계산은 불가능하다.

이처럼 현재의 관련 기술들은 많은 불편함과 문제점을 가지고 있다.

1.3. 연구 목표

영상 인식을 통해 이미지에서 음식들을 추출해내고 추출한 음식들에 대한 정보와 칼로리를 계산하는 프로그램을 구상한다. 사용자가 일일이 음식에 대한 칼로리를 조사 하는 것이 아닌 음식에 대한 사진을 찍는 것만으로도 칼로리에 대한 정보를 직관적으로 알 수 있게 하고 음식의 종류를 몰라도 사진만 있으면 음식에 대한 정보를 알 수 있게 하는 것이 본 연구의 목표이다.

1.3.1 세부 목표

- (1) 이미지의 정보를 저장하는 트레이닝셋 구축
 - a. 음식 이미지 파일, 음식 이름, 음식 칼로리와 영양 정보 등을 저장
 - b. 사용자가 칼로리를 계산하고 싶은 이미지를 넣으면 특정 정보를 모아놓은 트레이닝셋과 비교하기 위한 목적
(ex. 칼로리를 계산하고 싶은 이미지의 특징을 치킨 이미지 10개, 사과 이미지 10개를 넣은 트레이닝셋과 비교)
- (2) 사진에서 음식 이미지를 특정할 수 있는 정보를 추출
 - a. 형상, 질감 및 색상을 기준으로 이미지 특징을 추출
 - b. 사용자가 섭취하는 음식의 실제 크기 추정
- (3) 머신러닝 지도학습 알고리즘을 이용한 프로그램의 정확도 향상
 - a. SVM(Support Vector Machine) 알고리즘 공부하여 적용
- (4) 딥러닝 알고리즘을 이용하여 이미지 분류
 - a. CNN(Convolution Neural Network) 알고리즘 공부하여 적용
- (5) 웹 애플리케이션을 만들어 칼로리 출력
 - a. 웹 애플리케이션을 구축해 사용자를 위한 서비스 제공
 - b. 사용자가 입력한 이미지에 대해 음식의 칼로리와 영양성분 출력

1.4. 요구 조건 분석

- (1) 음식 이미지, 칼로리, 영양성분에 관한 정보를 수집해야 한다.
- (2) 이미지에서 각각의 특징들에 대한 색상, 질감, 형태를 추출해야 한다.
- (3) 딥러닝, 머신러닝을 이용해서 추출한 특징들을 학습, 분류해야 한다.
- (4) 테스트를 위해서 테스트셋을 구성하고 특징들을 추출해야 한다.
- (5) 음식의 부피를 예상하여 음식의 질량을 추정해야 한다.
- (6) 음식의 칼로리와 영양성분을 계산해야 한다.
- (7) 웹 애플리케이션을 적절하게 만들어 결과를 출력해야 한다.

1.5. 현실적 제약사항

(1) 제한된 하드웨어 용량 및 성능에 따른 실행속도로 인해 매우 많은 양의 트레이닝을 갖추기 제한됨

→ epoch의 양을 10개부터 시작하여 오버피팅이 발생하지 않는 적정 수준의 양까지 점차 늘려 나갈 예정

(2) 국 요리 또는 음료수 같은 고체가 아닌 액체형태의 음식에 대해서 정확한 분류가 제한됨

→ 고체와 액체형태의 음식물에 대한 트레이닝셋을 각각 따로 구축하고 학습

(3) 텐서플로우-gpu 사용을 위한 그래픽카드의 부재로 최적화된 딥러닝 환경 설정을 갖추 수 없음

-> 속도가 조금 느린 텐서플로우-cpu를 사용

2. 배경 지식

2.1. openCV

OpenCV(Open Source Computer Vision)은 실시간 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리이다. 영상 관련 라이브러리로서 사실상 표준의 지위를 가지고 있다. 조금이라도 영상처리가 들어간다면 필수적으로 사용하게 되는 라이브러리이다.



[그림 5] openCV 로고

2.1.1. openCV 주요함수

(1) cv2.adaptiveThreshold

cv2.adaptiveThreshold(img, value, adaptive method, threshold Type, block size, C)

- img: grayscale 이미지
- value: 계산된 임계값과 thresholdType에 의해 픽셀에 적용될 최댓값
- adaptive method: 사용할 임계값 계산 알고리즘

(1) cv2.ADAPTIVE_THRESH_GAUSSIAN_C : X, Y를 중심으로 block Size * block Size 안에 있는 픽셀 값의 평균에서 C를 뺀 값을 임계값으로 한다.

(2) cv2.ADAPTIVE_THRESH_MEAN_C : X, Y를 중심으로 block Size * block Size 안에 있는 Gaussian 윈도우 기반 가중치들의 합에서 C를 뺀 값을 임계값으로 한다.

- blocksize: block * block size에 각각 다른 임계값이 적용된다.
- C: 보정 상수로서 adaptive에 계산된 값에서 양수면 빼주고 음수면 더해준다.
- 반환 값 : 이진화된 이미지

(2) cv2.calcHist

cv2.calcHist(img, channel, mask, histSize, ranges)

- img : 입력 이미지
- channel : 처리할 채널 (1채널:[0], 2채널:[0,1], 3채널:[0,1,2])
- mask: 마스크에 지정한 픽셀만 히스토그램 계산
- histSize : 계급(bin)의 수, 채널의 수에 따라 다르게 표현
- ranges : 각 픽셀이 가질수 있는 값의 범위

(3) cv2.getGaborKernel

cv2.getGaborKernel(ksize, sigma, theta, lambda, gamma, psi, ktype)

- ksize : Gabor Kernel 사이즈
- sigma : Gaussian function의 표준편차로써 Kernel의 너비 결정
- theta : Gabor Filter가 추출하는 Edge의 방향을 결정.
- lambda : Gabor Filter Kernel의 정현파 파장의 크기 결정
- gamma : 중심으로부터 Gabor Filter가 얼마나 이동하는지를 나타내는 값

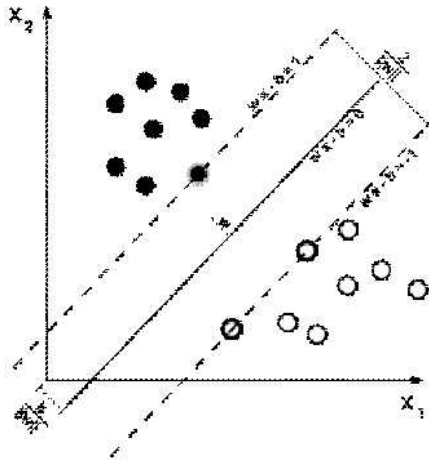
(4) cv2.findContours

cv2.findContours(src, mode, method [, contours, hierarchy, offset])

- mode : 컨투어 제공 방식
 - cv2.RETR_EXTERNAL : 가장 바깥쪽 라인만 제공
 - cv2.RETR_LIST : 모든 라인을 계층없이 제공
 - cv2.RETR_CCOMP : 모든 라인을 2계층으로 제공
 - cv2.RETR_TREE : 모든 라인의 모든 계층 정보를 트리 구조로 제공
- method : 근사 값 방식 선택
 - cv2.CHAIN_APPROX_NONE : 근사 계산하지 않고 모든 좌표 제공
 - cv2.CHAIN_APPROX_SIMPLE : 컨투어 꼭짓점 좌표만 제공
 - cv2.CHAIN_APPROX_TC89_L1 : Teh_Chin 알고리즘으로 좌표 개수 축소
 - cv2.CHAIN_APPROX_TC89_KCOS : Teh_Chin 알고리즘으로 좌표 개수 축소

2.2. SVM

SVM(Support Vector Machine)은 기계 학습의 분야 중 하나로 패턴 인식, 자료 분석을 위한 지도 학습 모델이며, 주로 분류와 회귀 분석을 위해 사용한다.



주어진 데이터 집합을 바탕으로 하여 새로운 데이터가 어느 카테고리에 속할지 판단하는 비확률적 이진 선형 분류 모델을 만든다.

선형 SVM 이 흰 원과 검은색 원을 직선으로 분리한다.

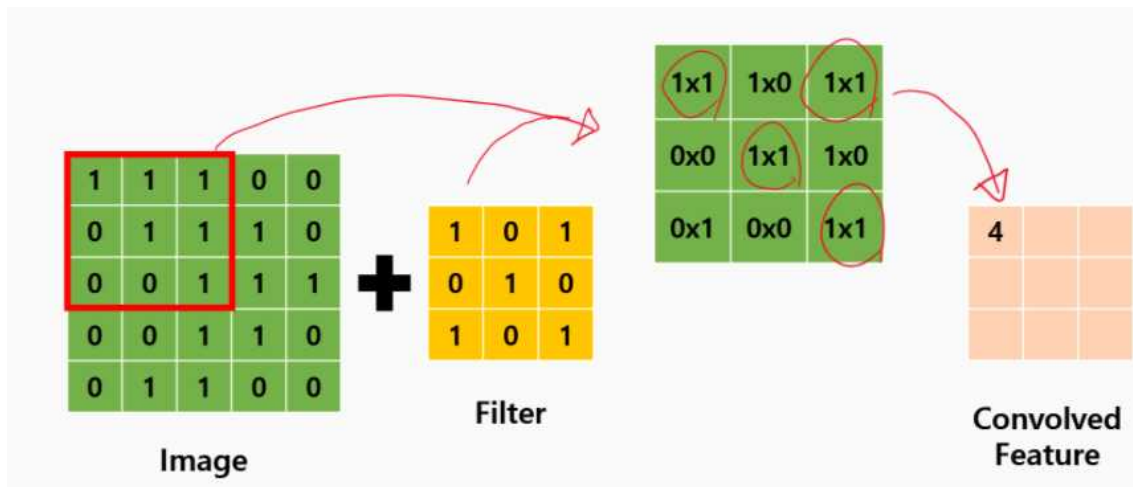
[그림 6] SVM 예시

이 SVM 모델을 활용하여 특정 음식의 특징 벡터 데이터를 구분한다.

2.3. CNN

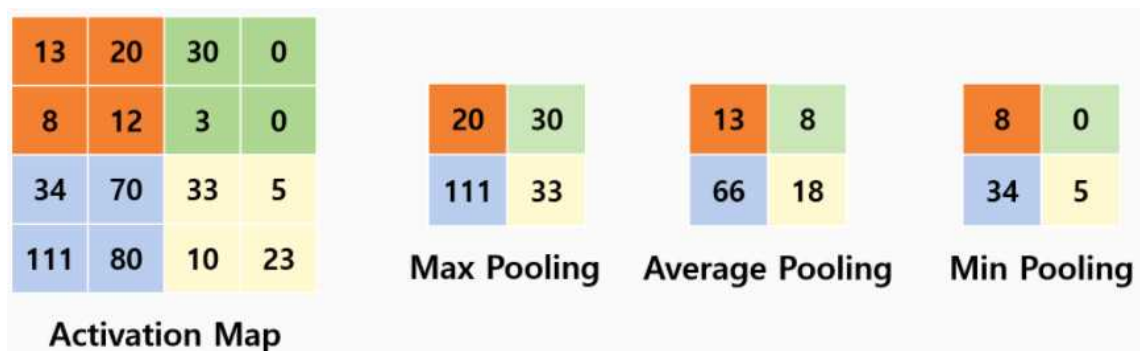
CNN(Convolutional Neural Network)은 딥러닝 방법의 일종으로 이미지 분류를 위한 인공지능에 가장 많이 사용되고 뛰어난 방법이다.

CNN은 크게 전처리와 분류 두 단계로 구성이 된다. 전처리 단계는 convolution, polling 이란 두 작업으로 이루어지며 전처리 단계 즉, 이 두 단계를 거치는 목적은 전처리라는 이름에 걸맞게 분류기에 들어오는 데이터를 정확한 분류를 위해 사전 처리를 하는 것이다.



[그림 7] convolution

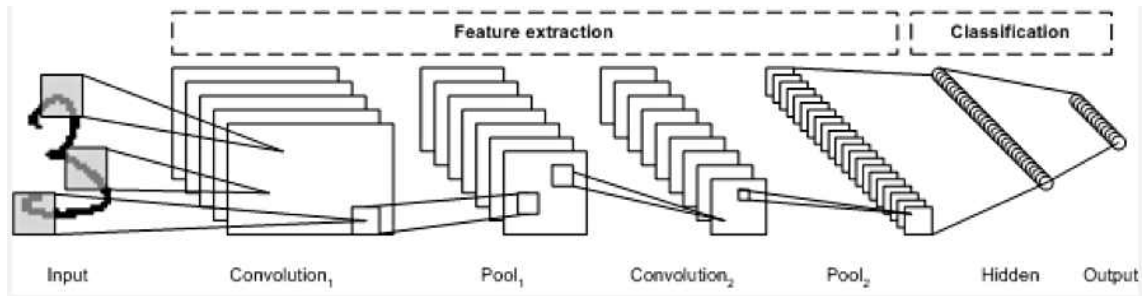
convolution이란 들어온 데이터의 특징을 추출하는 작업이다. 여러 개의 학습된 가중치 값을 가지고 있는 필터를 통해 해당 이미지에서 분류에 도움이 될 만한 특징들을 추출하는 작업을 하게 된다.



[그림 8] pooling

pooling이란 convolution을 통해 이미지의 특징이 정돈된 상태에서 가장 핵심적인 부분을 추출하는 것이다. 여기서도 필터를 통해 데이터를 추출하게 된다. 하지만 convolution에서 필터의 가중치를 통해 계산하는 것과는 달리 그냥 해당 이미지의 filter 사이즈에서 가장 큰 값을 추출하게 된다.

분류 단계에서는 이렇게 전처리를 한 데이터를 가지고 실제 이제 다층의 퍼셉트론에 데이터를 집어넣어서 분류 작업을 하게 된다. 분류 단계에는 Flatten Layer 와 Softmax Layer 가 존재한다. Flatten Layer는 데이터 타입을 Fully Connected 네트워크 형태로 변경하고 입력데이터의 형태 변경만 수행한다. Softmax Layer는 분류 작업 수행을 수행한다.



[그림 9] CNN 전체구조

2.4. Flask

Flask란 파이썬을 기반으로 하는 WSGI 마이크로 웹 프레임워크이다. 여기서 WSGI란 Web Server Gateway Interface로 파이썬 스크립트가 웹 서버와 통신하기 위한 인터페이스를 지칭하는 말이다. 그리고 마이크로 웹 프레임워크란 기존보다 확장성 및 가벼운 구조에 중점을 둔 프레임워크를 뜻한다. 내부 구성으로는 WSGI용 Library인 Werkzeug와 HTML에 데이터를 렌더링하는 엔진인 Jinja2 template으로 구성되어 있다.

Flask의 장점으로서는 타 파이썬 기반 웹 프레임워크 Django보다 자유도나 가벼움 측면에서 앞선다는 부분이다. Flask 경우에는 프레임워크를 사용자가 원하는 설계 방향으로 구축할 수 있다는 점이 있다. 이러한 장점이 있기 때문에 이 프로젝트를 진행함에 있어 웹 페이지 연동이나 결과 출력 같은 경우 비중을 크게 두지 않기 위해 이미 여러 기능들이 구축되어 있는 Django를 선택하기 보단 기본 구조만 되어 있고 데이터 가져오기나 출력을 위한 기능만을 간결하게 구축하기 위해서 Flask를 이용하였다.

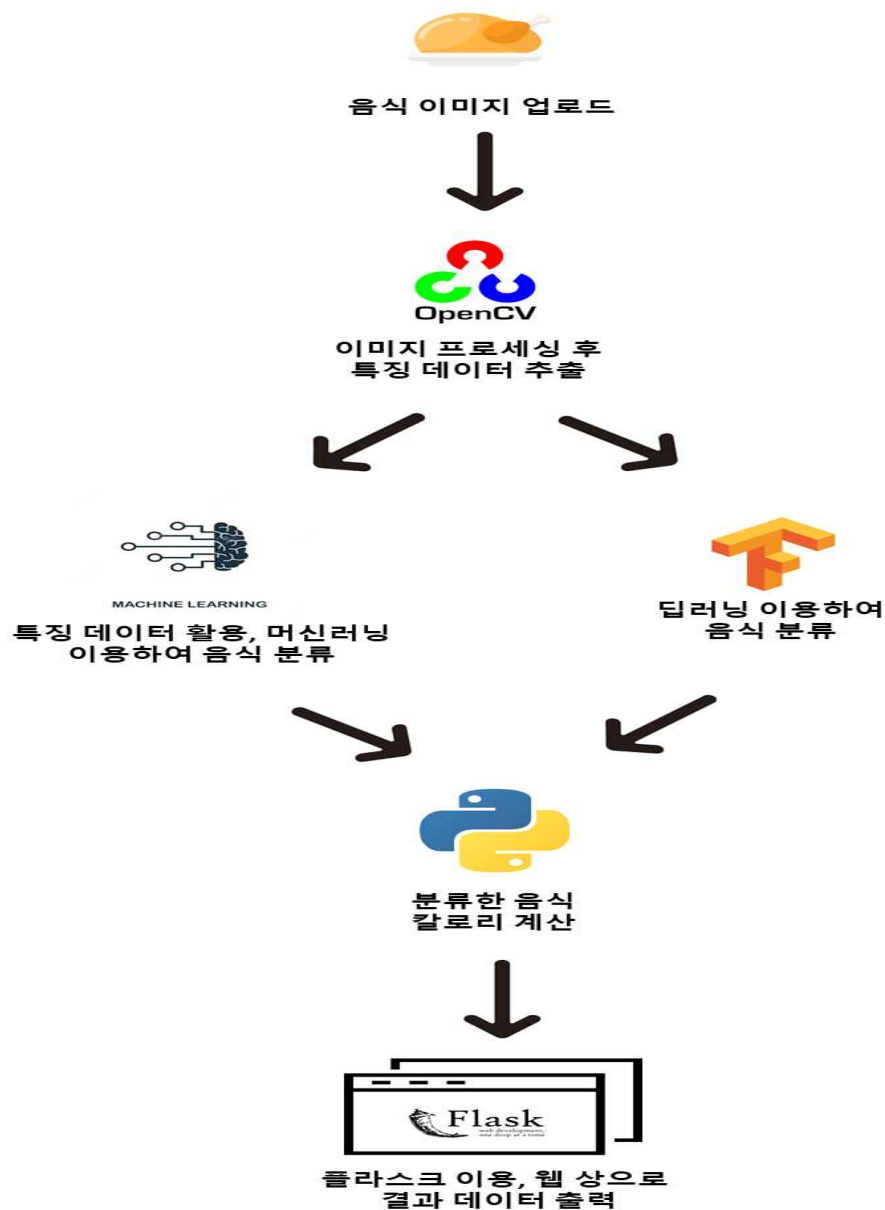


[그림 10] Flask 로고

3. 개발 내용

3.1. 설계도

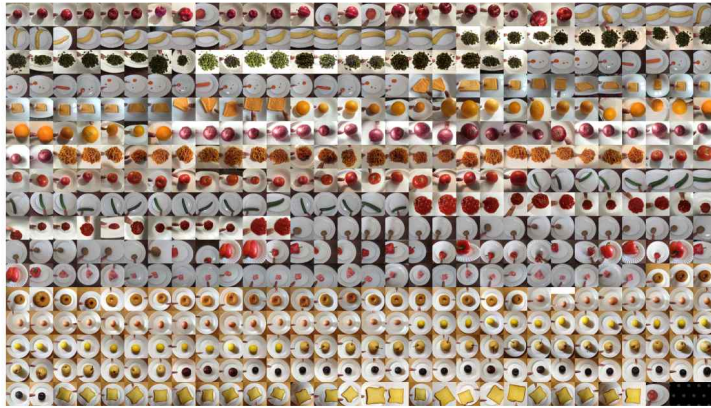
프로그램 전체 구성도는 아래의 그림과 같다. 칼로리를 계산하고 싶은 이미지를 업로드하면 이미지를 전처리하고 머신러닝, 딥러닝 모델에 적용하여 음식의 종류를 예측하고 부피와 질량을 추정해 칼로리와 영양성분 계산하고 웹에 출력한다.



[그림 11] 프로그램 전체 설계도

3.2 트레이닝셋 구축

총 20개의 음식(사과, 바나나, 콩, 당근, 치즈, 오렌지, 양파, 파스타, 토마토, 오이, 소스, 키위, 피망, 수박, 도넛, 계란, 레몬, 배, 자두, 빵)에 대하여 각각 25장씩 총 500장의 이미지 파일을 트레이닝셋으로 구축하였다.



[그림 12] 트레이닝셋 모음

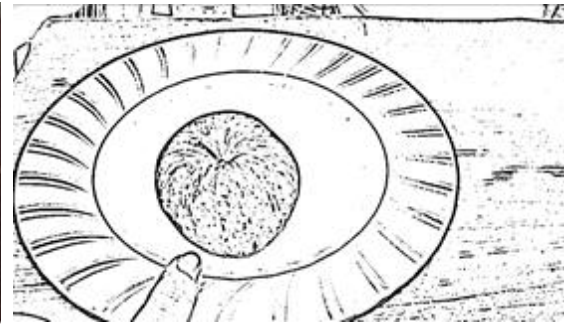
3.3. 음식 이미지 전처리

이미지에서 음식과 접시, 사람의 신체부분을 분할해서 음식의 영역을 계산한다. 이미지를 임계값 기준으로 흰색과 검은색으로 이진화 하기 위해서 openCV cv2.adaptiveThreshold 함수를 이용하고 이진화 이미지에서 윤곽선을 검출하기 위해 cv2.findContours 함수를 사용하며, 검출된 윤곽선을 그리기 위해 cv2.drawContours 함수를 사용한다.

```
def getAreaOfFood(img1):  
    #BGR에서 그레이 스케일 이미지로 변환  
    img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)  
    #미디언 블러링으로 노이즈 제거  
    img_filt = cv2.medianBlur( img, 5)  
    #이미지 이진화  
    img_th = cv2.adaptiveThreshold(img_filt, 255,  
                                  cv2.ADAPTIVE_THRESH_GAUSSIAN_C,  
                                  cv2.THRESH_BINARY, 11, 2)  
    contours, hierarchy = cv2.findContours(img_th, cv2.RETR_LIST,  
                                           cv2.CHAIN_APPROX_SIMPLE)
```



[그림 13] 원본이미지 img1



[그림 14] 이진화된 이미지 img_th

```
# 접시와 음식에 해당하는 가장 큰 윤곽선을 찾는다.
mask = np.zeros(img.shape, np.uint8)
largest_areas = sorted(contours, key=cv2.contourArea)
#검은색 배경에 가장 큰 컨투어만 그림
cv2.drawContours(mask, [largest_areas[-1]], 0, (255,255,255), -1)
#입력이미지에서 가장 큰 컨투어만 추출
img_bigcontour = cv2.bitwise_and(img1,img1,mask = mask)
```

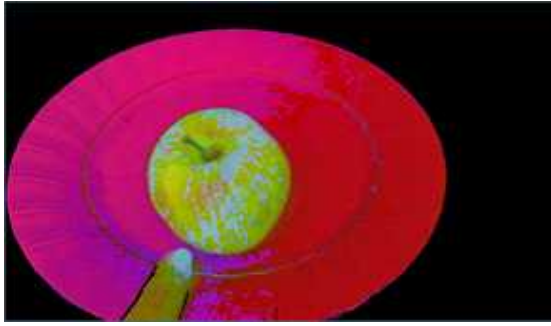


[그림 15] mask

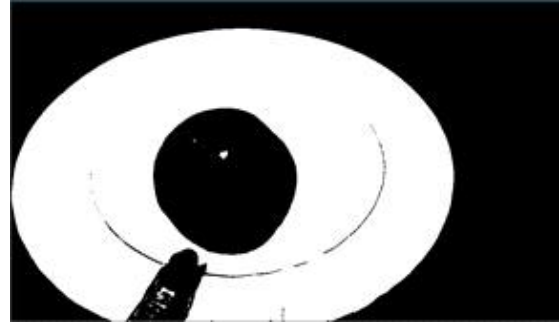


[그림 16] img_bigcontour

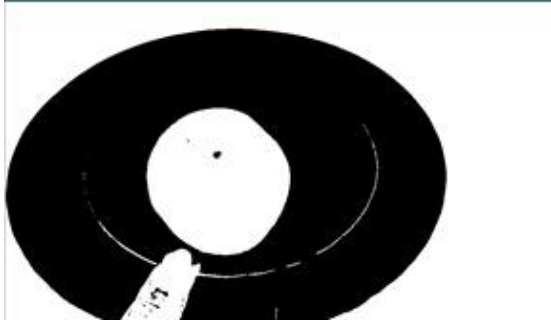
```
#hsv채널로 변경
hsv_img = cv2.cvtColor(img_bigcontour, cv2.COLOR_BGR2HSV)
h,s,v = cv2.split(hsv_img)
#inRange함수로 plate부분의 영역을 설정
mask_plate = cv2.inRange(hsv_img, np.array([0,0,100]),
                           np.array([255,90,255]))
mask_not_plate = cv2.bitwise_not(mask_plate)
#가장 큰 컨투어(plate)에서 plate가 아닌 부분만(음식) 추출
fruit_skin = cv2.bitwise_and(img_bigcontour,img_bigcontour,mask
                             = mask_not_plate)
```



[그림 17] hsv_image



[그림 18] mask_plate



[그림 19] mask_not_plate

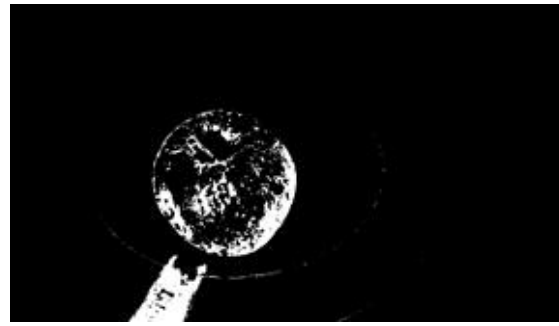


[그림 20] fruit_skin

```
#피부를 제외하기 위해 fruit_skin을 hsv채널로 변환
hsv_img = cv2.cvtColor(fruit_skin, cv2.COLOR_BGR2HSV)
#피부 영역을 설정
skin = cv2.inRange(hsv_img, np.array([0,10,60]), np.array([10,160,255]))
#Scalar(0, 10, 60), Scalar(20, 150, 255)
not_skin = cv2.bitwise_not(skin)
#피부가 아닌 부분만 절정, 음식 픽셀만 획득
fruit = cv2.bitwise_and(fruit_skin, fruit_skin, mask = not_skin)
```



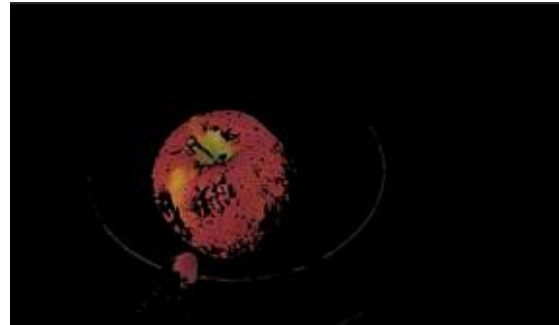

[그림 21] hsv_img



[그림 22] skin

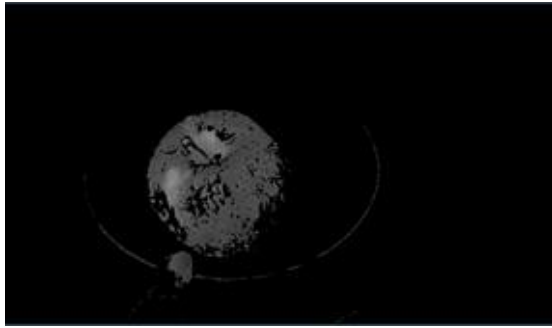


[그림 23] Not_skin

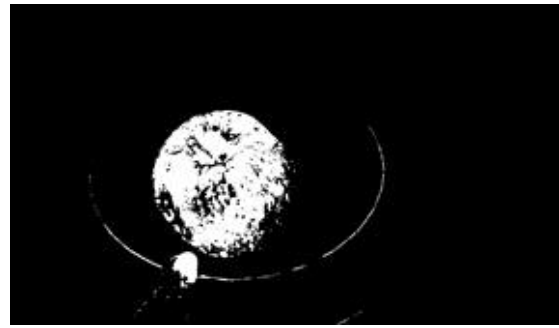


[그림 24] fruit

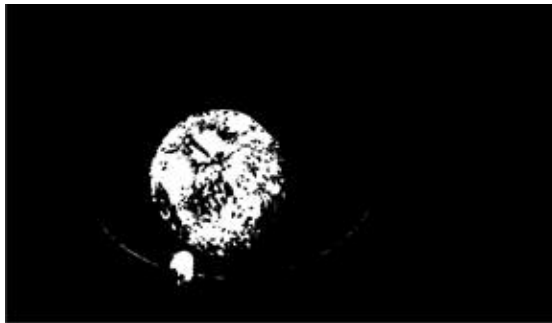
```
#그레이 이미지로 변환
fruit_bw = cv2.cvtColor(fruit, cv2.COLOR_BGR2GRAY)
#흑백 바이너리 이미지
fruit_bin = cv2.inRange(fruit_bw, 10, 255) #binary of fruit
#윤곽선을 찾기전 3x3 타원 커널을 사용한 erosion으로 어두운 영역을 늘림
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
erode_fruit = cv2.erode(fruit_bin,kernel,iterations = 1)
```



[그림 25] fruit_bw



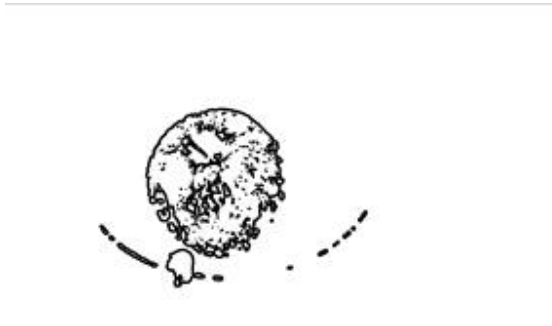
[그림 26] fruit_bin



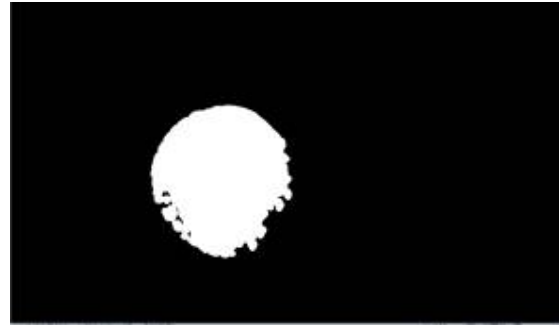
[그림 27] erode_fruit

```
#음식이 될 가장 큰 윤곽선을 찾는다
#erosion된 음식부분의 이미지를 이진화
img_th = cv2.adaptiveThreshold(erode_fruit,255,
                              cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
contours, hierarchy = cv2.findContours(img_th, cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)

mask_fruit = np.zeros(fruit_bin.shape, np.uint8)
largest_areas = sorted(contours, key=cv2.contourArea)
#음식만 추출하기위해 이진화된 이미지의 가장 큰 윤곽선을 마스크로 설정
cv2.drawContours(mask_fruit, [largest_areas[-2]], 0, (255,255,255), -1)
#13x13 타원 커널을 사용한 dilation으로 마스크의 밝은 영역을 늘림
kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(13,13))
mask_fruit2 = cv2.dilate(mask_fruit,kernel2,iterations = 1)
#fruit_bin에서 음식부분만 추출
res = cv2.bitwise_and(fruit_bin,fruit_bin,mask = mask_fruit2)
#img1에서 음식부분만 추출
fruit_final = cv2.bitwise_and(img1,img1,mask = mask_fruit2)
```



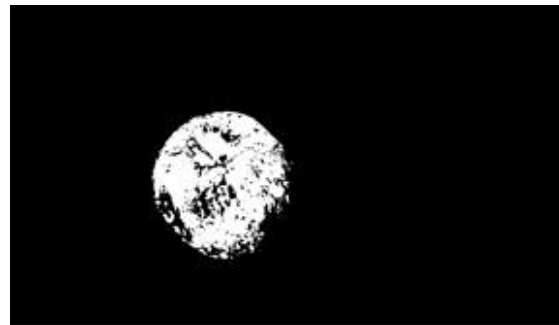
[그림 28] img_th



[그림 29] mask_fruit



[그림 30] mask_fruit2



[그림 31] res



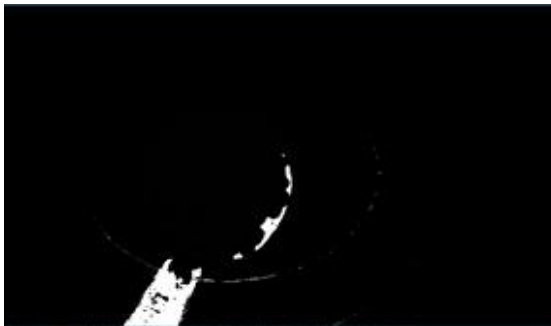
[그림 32] fruit_final

```
#최종 음식 마스크 area 면적계산
img_th = cv2.adaptiveThreshold(mask_fruit2,255,
                               cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
contours, hierarchy = cv2.findContours(img_th, cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)
largest_areas = sorted(contours, key=cv2.contourArea)
fruit_contour = largest_areas[-2]
fruit_area = cv2.contourArea(fruit_contour)
```

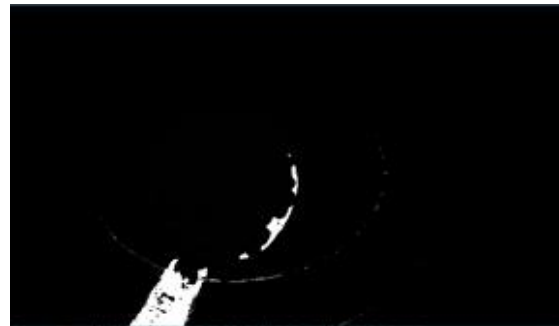


[그림 33] img_th

```
#finding the area of skin. find area of biggest contour
#신체의 피부 영역과 가장 큰 윤곽선의 area를 계산
skin2 = skin - mask_fruit2
#피부 영역의 윤곽선을 찾기전 erosion
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
skin_e = cv2.erode(skin2,kernel,iterations = 1)
#피부영역의 마스크 계산
img_th = cv2.adaptiveThreshold(skin_e,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                              cv2.THRESH_BINARY,11,2)
contours, hierarchy = cv2.findContours(img_th, cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)
mask_skin = np.zeros(skin.shape, np.uint8)
largest_areas = sorted(contours, key=cv2.contourArea)
cv2.drawContours(mask_skin, [largest_areas[-2]], 0, (255,255,255), -1)
```



[그림 34] skin2



[그림 35] skin_e



[그림 36] Mask_skin

```

#피부의 가장큰 윤곽선에 외접하는 가장작은 직사각형을 계산
skin_rect = cv2.minAreaRect(largest_areas[-2])
#직사각형의 각 꼭지점 반환
box = cv2.boxPoints(skin_rect)
print(box)
box = np.int0(box)
mask_skin2 = np.zeros(skin.shape, np.uint8)
cv2.drawContours(mask_skin2,[box],0,(255,255,255), -1)
#경계사각형의 height 계산
pix_height = max(skin_rect[1])
# cm단위로 변환시키는 변수
pix_to_cm_multiplier = 5.0/pix_height
skin_area = cv2.contourArea(box)

return fruit_area, mask_fruit2, fruit_final, skin_area,
       fruit_contour, pix_to_cm_multiplier

```



[그림 37] Mask_skin2

이미지 전처리 과정을 마치면 음식 이미지로 추정한 윤곽선의 면적, 음식 이미지의 경계선, 음식 영역, 손가락으로 추정한 영역의 면적, 음식의 윤곽선, 이미지내의 손가락 세로부분의 픽셀 단위를 cm 단위로 바꾼 값을 반환한다.

3.4. 음식 이미지 특징 추출

이미지에 대해서 색상, 질감, 모양에 대한 특징을 추출하여 전체적인 특징 벡터를 생성한다.

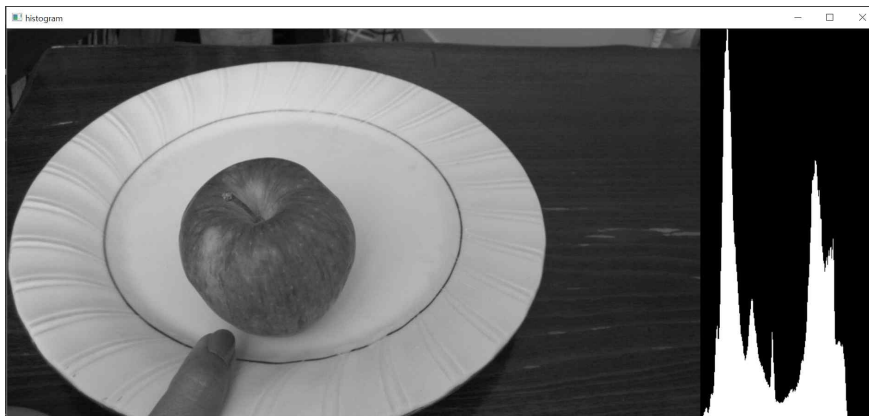
3.4.1. 음식 색상 추출

이미지 히스토그램을 이용해서 색상에 대한 특징을 추출한다. 히스토그램에서 픽셀값이 1인 픽셀의 수, 2인 픽셀의 수 이런 방식으로 범위만큼 계산하여 그림 4 와 같이 이미지에 속한 픽셀들의 색상이나 명암의 분포를 파악할 수 있다. 히스토그램을 계산하기 위해 openCV cv2.calcHist 함수를 이용한다.

```
hist = cv2.calcHist([img_hsv], [0, 1, 2], None, [6, 2, 2], [0, 180, 0, 256, 0, 256])
```

cs

채널은 3d, 범위는 [6, 2, 2] 를 사용하였고 각 픽셀의 범위는 H 는 0~180, S 는 0~255, V 는 0~255



[그림 38] 이미지 히스토그램 적용

3.4.2. 음식 질감 추출

특정 물체에 대한 질감 특징을 얻어내기 위해 Gabor Filter를 사용하였다. openCV에서 Gabor Filter를 사용하기 위한 cv2.getGaborKernel 함수를 이용한다.


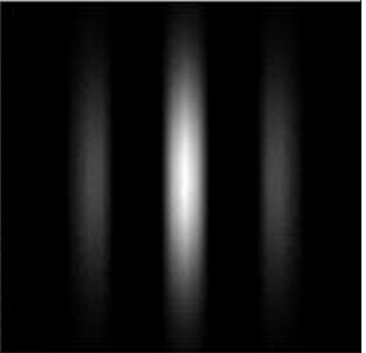

```
def build_filters():
    """
    가버필터 커널 생성 함수
    """
    filters = []
    ksize = 31
    for theta in np.arange(0, np.pi, np.pi / 8):
        for wav in [8.0, 13.0]:
            for ar in [0.8, 2.0]:
                kern = cv2.getGaborKernel((ksize, ksize), 5.0, theta, wav, ar,
                                           0, ktype=cv2.CV_32F)
                filters.append(kern)
    return filters
```

커널 사이즈를 31*31로 정하고 32개의 Gabor Filter를 위해서 theta 는 8등분, lambda 는 8.0과 13.0 (정확도 분석 후 지정), gamma는 0.8과 12.0 지정(lambda와 동일)하여 8*2*2 개의 Gabor Filter Kernel 생성 후 필터에 추가한다.

```
def process(img, filters):
    """
    주어진 이미지와 생성한 가버 커널 함수를 이용하여
    가버 필터링을 적용하는 함수
    """
    feature = []
    accum = np.zeros_like(img)
    for kern in filters:
        fimg = cv2.filter2D(img, cv2.CV_8UC3, kern)
        a, b = EnergySum(fimg)
        feature.append(a)
        feature.append(b)
        np.maximum(accum, fimg, accum)

    M = max(feature)
    m = min(feature)
    feature = list(map(lambda x: x * 2, feature))
    feature = (feature - M - m)/(M - m);
    mean=np.mean(feature)
    dev=np.std(feature)
    feature = (feature - mean)/dev;
    return feature
```

실제 입력 이미지를 Gabor Filter Kernel에 필터링 시킨 후 나온 이미지 값에 대해 정규화를 진행해 질감특징을 추출한다.

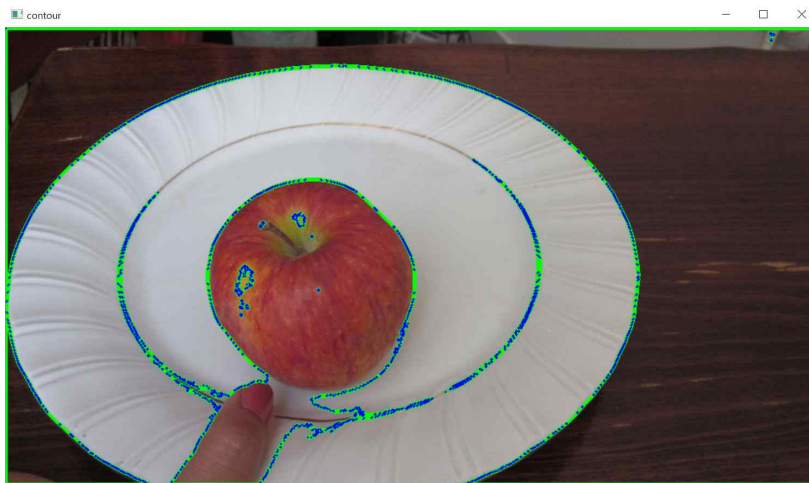
<원본 이미지> (예시 이미지)	통과시킬 Gabor Filter (Gabor Filter 중 하나)	통과 후 이미지
		

[그림 39] Gabor Filter 적용

위와 같이 Gabor Filter를 이용하여 질감 특징을 추출한다.

3.4.3. 음식 모양 추출

cv2.drawContours 함수를 이용해서 그림 6 과 같이 이미지에 대해 contour(이미지 경계선)를 나타낼 수 있다.



[그림 40] 이미지 Contour 출력

모멘트 계산을 위해 openCV cv2.moments() 함수를 사용한다.

```
moments = cv2.moments(contours[0])
hu = cv2.HuMoments(moments)
```

contour[0] : 모멘트계산대상 컨투어좌표, moments : 결과 모멘트
Hu-Moments은 회전 및 크기에 영향을 받지 않는 moments,
cv2.HuMoments() 함수를 이용해 구한다.

```
def createFeature(img):  
    '''  
    이미지의 세 가지 특징(색상, 질감, 모양)을 사용하여 이미지의  
    특징 벡터 추출 함수 생성  
    '''  
    feature = []  
    areaFruit, binaryImg, colourImg, areaSkin, fruitContour,  
    pix_to_cm_multiplier = getAreaOfFood(img)  
    #이미지 세그멘테이션에 이미지 값을 넣은 리턴 값을 각각 변수에 할당  
    color = getColorFeature(colourImg)  
    texture = getTextureFeature(colourImg)  
    shape = getShapeFeatures(binaryImg)  
    #세 가지 특징 추출 함수 입력 파라미터에 필요한 이미지 값을 넣은 후  
    리턴 값을 각각 변수에 할당
```

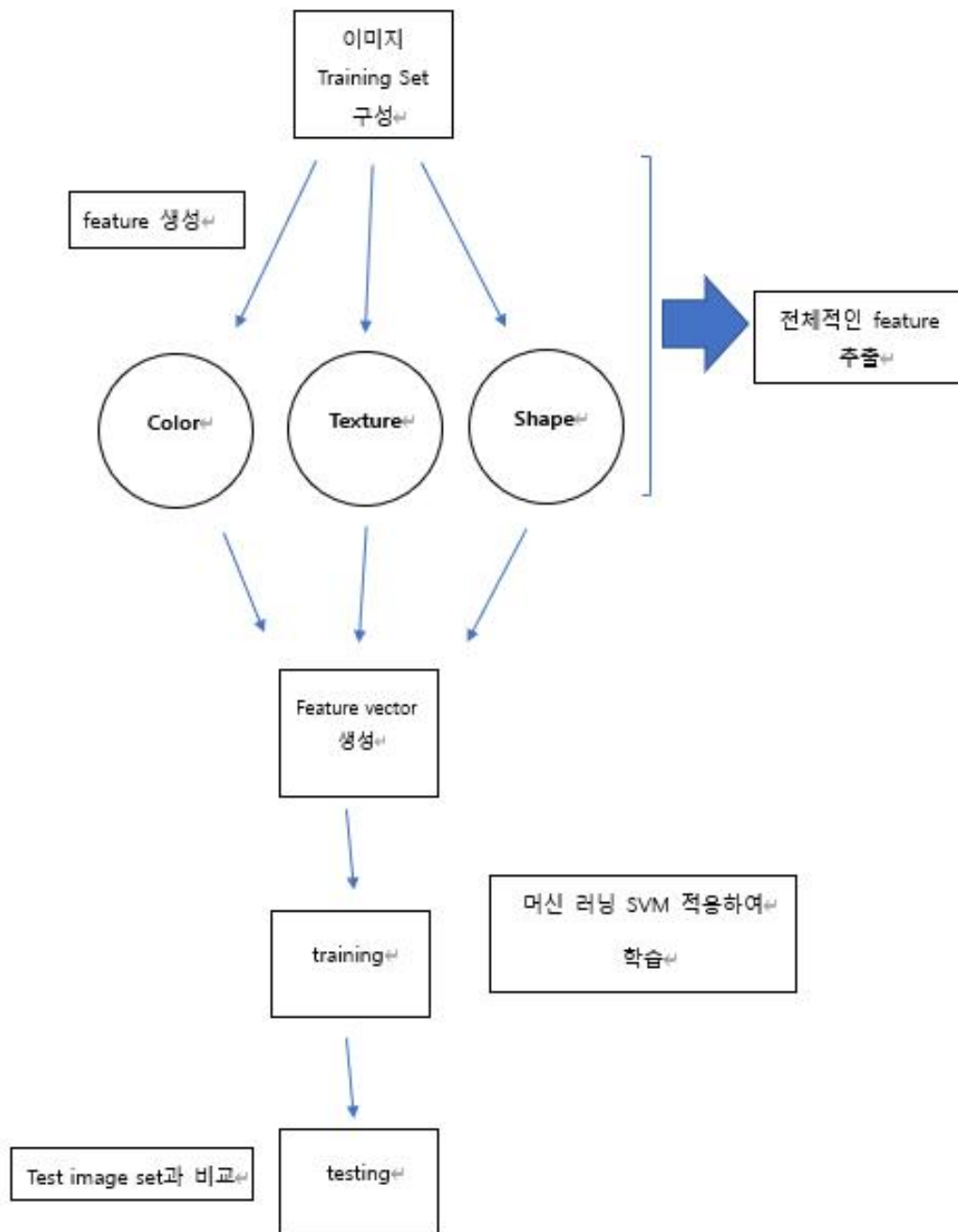
세 가지 특징 추출 함수를 통한 색상, 질감, 모양 특징을 조합하여 특징 벡터를 추출하고 추출한 색상, 질감, 모양을 특징에 추가 한 뒤 특징에 대해서 정규화한 뒤 데이터를 반환 한다.

3.5. 특징 데이터 학습

3.5.1. 머신러닝(SVM) 모델 학습

아래 그림은 머신러닝 svm 모델을 만들기 위한 전체적인 과정이다. 이미지셋을 구성하고 각각 이미지에 대해 색상, 질감, 형상 특징들을 통해 특징 벡터를 만들어내고 트레이닝을 통해 특징벡터를 학습을 시키고 테스트하여 머신러닝 svm 모델을 생성한다.

생성한 svm 모델을 이용하여 사용자가 이미지를 입력하면 svm 모델에 적용하여 음식의 종류를 예측한다.



[그림 41] 머신러닝 학습 과정

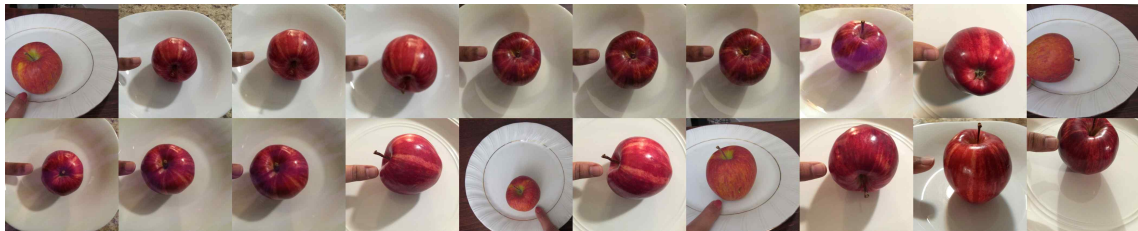
(1) Training

SVM 모델을 활용하여 특정 음식의 Feature Vector 데이터를 구분한다.

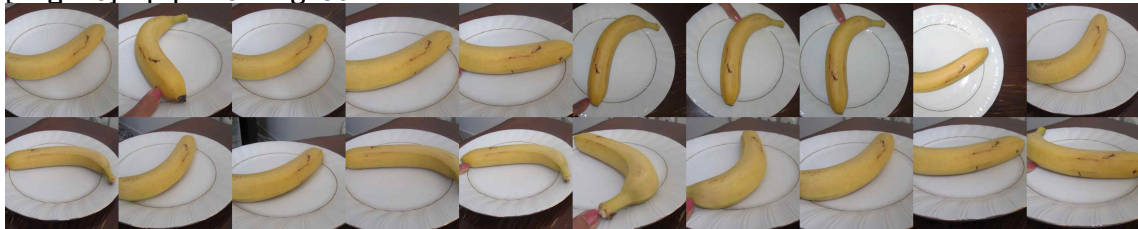
Label	음식명
1	사과
2	바나나
3	콩
...	...

[그림 42] 음식별 라벨 예시

각각의 음식에 대한 Label 을 위의 표처럼 1,2,3 과 같은 숫자 형태로 설정하였다. 각각의 Label 은 동일한 음식에 대한 25장의 이미지를 포함하며, 1~20번 이미지는 트레이닝 목적이고, 21~25 번 이미지는 테스트 목적이다. 이미지 파일명을 "Label 명_1.jpg" 로 설정하였다.
(ex. 사과 1번 이미지에 대한 파일명 -> 1_1.jpg)



[그림 43] 사과 Training Set



[그림 44] 바나나 Training Set



[그림 45] 콩 Training Set

위의 그림들과 같이 음식 한 개에 대해서 20장의 이미지를 트레이닝셋으로 수집하고 학습을 진행한다.

```
fea, farea, skinarea, fcont, pix_to_cm = readFeatureImg(  
    "./images/All_Images/" + str(j) + "_" + str(i) + ".jpg")
```

트레이닝 이미지에서 readFeatureImg 함수로 각 음식에 대한 특징벡터를 추출한다.

```
trainData = np.array(feature_mat, dtype=np.float32).reshape(-1, 94)  
responses = np.array(response, dtype=int)  
svm = cv2.ml.SVM_create()  
svm.setKernel(cv2.ml.SVM_LINEAR)  
svm.setType(cv2.ml.SVM_C_SVC)  
svm.setC(2.67)  
svm.setGamma(5.383)  
svm.train(trainData, cv2.ml.ROW_SAMPLE, responses)  
svm.save('svm_data.dat')
```

feature_mat에 각 음식의 특징벡터를 추가해서 최종적으로 95x1 차원 벡터 trainData를 가지고 SVM 모델에 실제값과 함께 학습시켜 svm 모델을 생성한다.

(2) Testing



[그림 46] 사과 Test Set



[그림 47] 바나나 Test Set



[그림 48] 콩 Test Set

생성한 머신러닝 학습모델로 임의의 데이터 셋을 트레이닝 하였고, 정확도를 측정하기 위해 위의 그림과 같이 각 음식에 대해 5 개의 이미지를 Test Set 으로 구축하였다.

```
# testing
testData = np.array(feature_mat, dtype=np.float32).reshape(-1, 94)
responses = np.array(response, dtype=int)
result = svm_model.predict(testData)[1]
mask = result == responses
```

각 테스트이미지에서 readFeatureImg 함수를 통해 추출된 특징벡터를 모아 95x1 차원벡터 testData 로 변환하고 이것을 미리 학습된 SVM 모델을 load 하여 예측한다.

```
# testing accuracy
correct = np.count_nonzero(mask)
print(correct) # 일치하는 이미지수
print(result.size) # 테스트셋 이미지수
print((correct * 100.0) / result.size) # 정확도
```

테스트 이미지셋에 대하여 테스트를 진행하면 91%의 정확도를 보인다.

	A	B	C
1	Image name	Desired response	Output label
2			
3	./images/Test_Images/1_21.jpg	1	1
4			
5	./images/Test_Images/1_22.jpg	1	1
6			
7	./images/Test_Images/1_23.jpg	1	1
8			
9	./images/Test_Images/1_24.jpg	1	1
10			
11	./images/Test_Images/1_25.jpg	1	1
12			
13	./images/Test_Images/2_21.jpg	2	2
14			
15	./images/Test_Images/2_22.jpg	2	2
16			
17	./images/Test_Images/2_23.jpg	2	2
18			
19	./images/Test_Images/2_24.jpg	2	2
20			
21	./images/Test_Images/2_25.jpg	2	2
22			
23	./images/Test_Images/3_21.jpg	3	3
24			
25	./images/Test_Images/3_22.jpg	3	3
26			
27	./images/Test_Images/3_23.jpg	3	3
28			
29	./images/Test_Images/3_24.jpg	3	3
30			
31	./images/Test_Images/3_25.jpg	3	3

테스팅 결과를 직관적으로 알 수 있게 하기 위해 엑셀 파일로 테스트 결과를 출력하였다.

- image name : 음식 파일명
- Desired response : 음식의 실제 Label
- Output label : 예측결과

Desired response 와 Output label 의 값을 비교하여 제대로 예측이 되었는지 확인가능하다.

[그림 49] SVM Testing 결과

3.5.2. 딥러닝(CNN) 모델 학습

딥러닝 CNN 모델을 생성하기 위해 머신러닝 모델 생성과정과 동일하게 트레이닝셋을 먼저 구성하고 트레이닝, 테스트 과정을 진행한다.

train 폴더에 트레이닝셋 데이터를 저장한다. 각각 음식별로 음식의 이름대로 train 폴더의 하위폴더로 생성하고 음식별 폴더안에 음식 이미지를 저장한다. 학습을 진행할때 폴더의 이름에 맞게 분류하여 학습을 진행한다.

```
batch_size = 32
train_datagen = ImageDataGenerator(rescale=1/255)
train_generator = train_datagen.flow_from_directory(
    'train',
    target_size=(200, 200),
    batch_size=batch_size,
    classes = [
        'Apple',
        'Banana',
        'Beans',
        'Carrot',
        'Cheese',
        'Orange',
        'Onion',
        'Pasta',
        'Tomato',
        'Cucumber',
        'Sauce',
        'Kiwi',
        'Capsicum',
        'Watermelon',
        'Doughnut',
        'Egg',
        'Lemon',
        'Pear',
        'Plum',
        'Bread'
    ],
    class_mode='categorical')
```

먼저 ImageDataGenerator 메소드를 사용하여 학습에 사용될 원본 이미지의 0 ~ 255 RGB 계수를 1/255로 스케일링 하여 0 ~ 1 범위로 변환 시켜준다. flow_from_directory 메소드로 현재 위치의 train 폴더의 구조를 그대로 가져와서 train generator를 생성한다. 원본 트레이닝 이미지를 200x200으로 자동으로 조절하고 배치 사이즈를 32로 지정하였다. 다중 클래스 문제이므로 class mode는 'categorical'로 지정하였다.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(200, 200, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(20, activation='softmax')
])
```

총 13개의 레이어로 모델 구성

- 컨볼루션 레이어 : 입력 이미지 크기 200 x 200, 입력이미지 채널 3개, 3x3 크기 필터 16개, 활성화 함수 'relu'
- 맥스풀링 레이어 : 풀 크기 : 2 x 2
- 컨볼루션 레이어 : 3 x 3 크기 필터 64개, 활성화 함수 'relu'
- 맥스풀링 레이어 : 풀 크기 : 2 x 2
- 컨볼루션 레이어 : 3 x 3 크기 필터 64개, 활성화 함수 'relu'
- 맥스풀링 레이어 : 풀 크기 : 2 x 2
- 컨볼루션 레이어 : 3 x 3 크기 필터 64개, 활성화 함수 'relu'
- 맥스풀링 레이어 : 풀 크기 : 2 x 2
- 컨볼루션 레이어 : 3 x 3 크기 필터 64개, 활성화 함수 'relu'
- 플래튼 레이어
- 댄스 레이어 : 출력 뉴런 수 128개, 활성화 함수 'relu'
- 댄스 레이어 : 출력 뉴런 수 3개, 활성화 함수 'softmax'


```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['acc'])
```

loss : 현재 가중치 세트를 평가하는 데 사용한 손실 함수, 다중 클래스 문제이므로 'categorical_crossentropy'으로 지정한다.

optimizer : 지수이동 평균을 이용한 최적화 알고리즘으로 경사 하강법 알고리즘 중 하나인 'RMSprop'을 사용, 학습률 lr(learning rate)은 0.001으로 설정한다.

metrics : 평가 척도를 나타내며 분류 문제에서는 일반적으로 'accuracy'으로 지정한다.

```
n_epochs = 25
history = model.fit_generator(
    train_generator,
    steps_per_epoch=int(total_sample/batch_size),
    epochs=n_epochs,
    verbose=1)
```

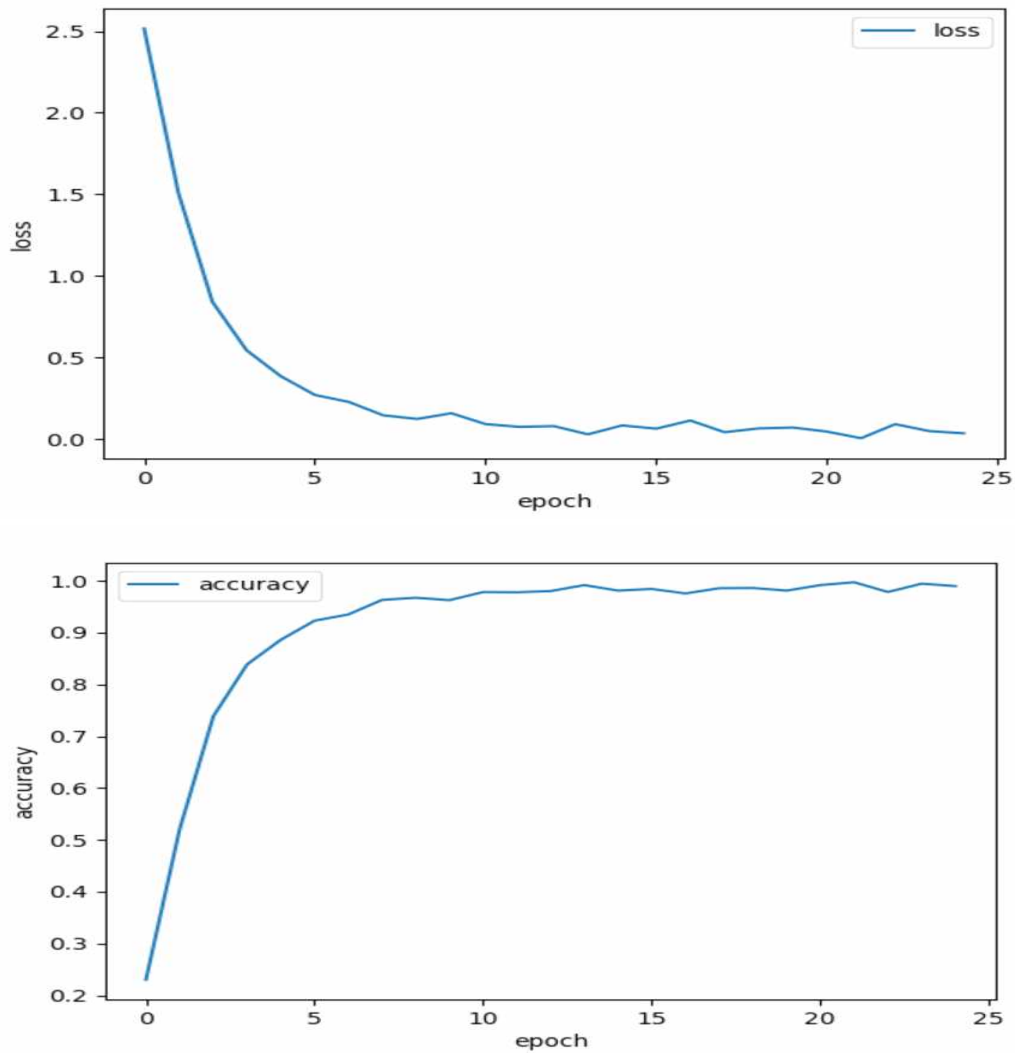
첫번째인자 : 훈련데이터셋을 제공할 제네레이터로 앞서 생성한 train_generator을 지정한다.

steps_per_epoch : 한 epoch에 사용한 스텝 수를 지정합니다. 전체 트레이닝 샘플을 배치사이즈 32으로 나눈 정수 값으로 설정한다..

epochs : 전체 훈련 데이터셋에 대해 학습 반복 횟수를 지정한다. 25번을 반복적으로 학습시킨다.

epoch 의 횟수를 10회부터 35회까지 실험해보았는데 25회를 초과 할 때부터는 오버피팅 현상이 발생하여 25회를 가장 적정 학습 횟수라고 판단하여 학습 횟수 epoch를 25로 지정하였다.

20가지의 음식 이미지에 대해 딥러닝 CNN 을 이용해 학습을 진행 하면 아래와 같은 정확도를 보인다.



[그림 50] epoch 에 따른 loss, accuracy

```
Epoch 25/25

1/15 [=>.....] - ETA: 26s - loss: 0.0012 - acc: 1.0000
2/15 [==>.....] - ETA: 22s - loss: 0.0013 - acc: 1.0000
3/15 [====>.....] - ETA: 19s - loss: 0.0010 - acc: 1.0000
4/15 [=====>.....] - ETA: 17s - loss: 0.0013 - acc: 1.0000
5/15 [=====>.....] - ETA: 15s - loss: 0.0012 - acc: 1.0000
6/15 [======>.....] - ETA: 14s - loss: 0.0014 - acc: 1.0000
7/15 [======>.....] - ETA: 12s - loss: 0.0014 - acc: 1.0000
8/15 [======>.....] - ETA: 10s - loss: 0.0017 - acc: 1.0000
9/15 [======>.....] - ETA: 9s - loss: 0.0017 - acc: 1.0000
10/15 [======>.....] - ETA: 7s - loss: 0.0017 - acc: 1.0000
11/15 [======>.....] - ETA: 6s - loss: 0.0016 - acc: 1.0000
12/15 [======>.....] - ETA: 4s - loss: 0.0016 - acc: 1.0000
13/15 [======>.....] - ETA: 3s - loss: 0.0016 - acc: 1.0000
14/15 [======>.....] - ETA: 1s - loss: 0.0015 - acc: 1.0000
15/15 [=====] - 23s 2s/step - loss: 0.0015 - acc: 1.0000
-- Evaluate --
acc: 100.00%
```

[그림 51] 최대 정확도

3.6. 칼로리 계산



[그림 52] 칼로리 계산 과정

그림과 같이 음식의 단면적을 구하고 부피를 구해 질량을 획득하고 질량을 바탕으로 칼로리를 계산하는 형태를 가진다.

(1) 음식 면적 계산

2차원 이미지 내에서 일정한 크기를 가지는 물체를 비교대상으로 정하고 이미지 내의 음식과 비교대상의 넓이 비율을 통해 3차원 실세계에서의 실제 음식의 단면적의 넓이를 구하는 방식으로 접근하였다. 이때 비교대상이 될 만한 물체는 일정한 크기를 가지고 크기가 작은 동전과 같은 형태가 가장 적절하지만 항상 사용자가 소지하고 있는 손가락을 비교대상으로 정하였다. 따라서 사용자는 음식의 칼로리 정보를 획득하기 위해서는 음식 사진에 본인의 손가락을 함께 촬영하여야 한다.

제손가락크기 실제이미지면적 = 이미지상의 손가락면적 : 이미지상의 음식면적

실제 손가락크기를 세로 5cm, 가로 2.3cm 로 설정하고 위의 수식을 계산하면 아래의 코드와 같이 실세계에서의 실제 음식의 단면적을 계산할 수 있다.

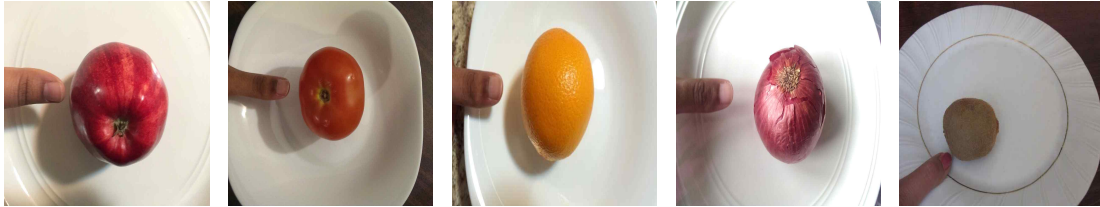
```
skin_multiplier = 5 * 2.3  
area_fruit = (area / skin_area) * skin_multiplier # area in cm^2
```

(2) 음식 부피 계산

위에서 구한 이미지 면적을 활용하여 부피를 계산한다. 음식의 종류를 예측한 상태이므로 음식의 종류에 따라 구체형, 원기둥형, 평면형, 타원형, 기타 케이스로 구분해 부피를 예측한다.

a. 구체 형태의 음식

구체 형태의 음식은 단면적이 원모양의 형태이므로 원의 반지름을 구해 구체의 부피를 구한다.

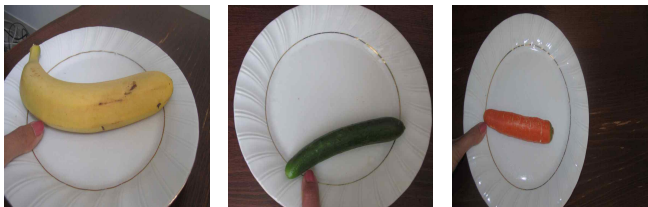


[그림 53] 구체 형태를 가지는 음식들

```
if label == 1 or label == 9 or label == 7 or label == 6 or label == 12 or  
label == 18 or label == 19:  
    radius = np.sqrt(area_fruit / np.pi)  
    volume = (4 / 3) * np.pi * radius * radius * radius
```

b. 원기둥 형태의 음식

원기둥 형태의 음식은 음식의 경계면(contour)에서 긴 변을 높이로 두고 짧은 부분의 절반이 원기둥 밑면의 반지름이다. 밑면의 반지름을 이용해 밑면의 넓이를 구하고 높이를 곱해서 원기둥의 부피를 구한다.

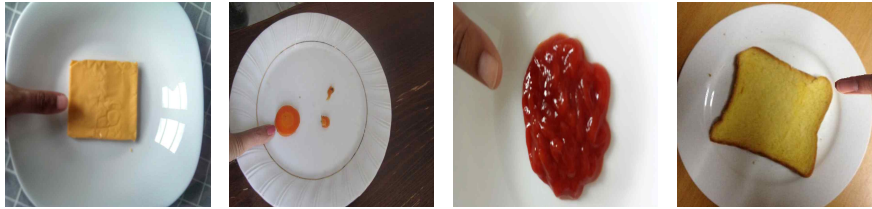


[그림 54] 원기둥 형태를 가지는 음식들

```
if label == 2 or label == 10 or (label == 4 and area_fruit > 30):  
    fruit_rect = cv2.minAreaRect(fruit_contour)  
    height = max(fruit_rect[1]) * pix_to_cm_multiplier  
    radius = area_fruit / (2.0 * height)  
    volume = np.pi * radius * radius * height
```

c. 평면 형태의 음식

평면 형태의 음식은 단면적에 높이 0.5cm 를 곱해서 음식의 부피를 구한다.

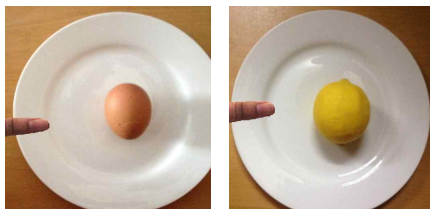


[그림 55] 평면 형태를 가지는 음식들

```
if (label == 4 and area_fruit < 30) or (label == 5) or (label == 11) or (label == 15) or (label == 20):
    volume = area_fruit * 0.5
```

d. 타원구 형태의 음식

타원구 형태의 음식은 음식의 형태를 구체 형태라고 생각하고 먼저 부피를 구한 뒤 구체 형태에서 0.6 을 곱하여 최종 음식의 부피를 구한다.

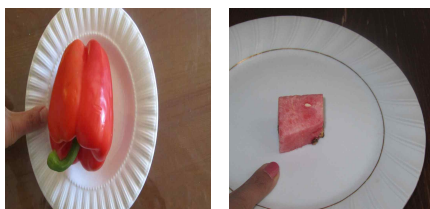


[그림 56] 타원구 형태를 가지는 음식들

```
if (label == 16) or (label == 17):
    fruit_rect = cv2.minAreaRect(fruit_contour)
    height = max(fruit_rect[1]) * pix_to_cm_multiplier
    radius = area_fruit / (2.0 * height)
    volume = np.pi * radius * radius * height * 0.6
```

e. 정육면체 형태의 음식

정육면체 형태의 음식은 단면적을 정사각형이라고 보고 정사각형의 한 변의 길이를 구해 음식의 부피를 구한다.



[그림 57] 정육면체 형태를 가지는 음식들

```
if (label == 13) or (label == 14) :
    sqaure_radius = np.sqrt(area_fruit)
    volume = sqaure_radius * sqaure_radius * sqaure_radius
```

f. 기타 여러 형태의 음식

특별한 형태가 없는 음식 같은 경우에는 높이값을 조절해 가며 부피를 테스트해보고 가장 적합한 높이값을 결정해 단면적과 곱하는 방식으로 음식의 부피를 구한다. 본 프로젝트에서는 파스타의 높이를 3cm 로 설정하였다.



[그림 58]기타 여러 가지 형태의 음식들

(3) 음식 질량 계산

음식들의 밀도를 조사하고 아래 식을 통해 음식의 질량을 구할 수 있다.

도 질량
부피

```
mass = volume * density * 1.0
```

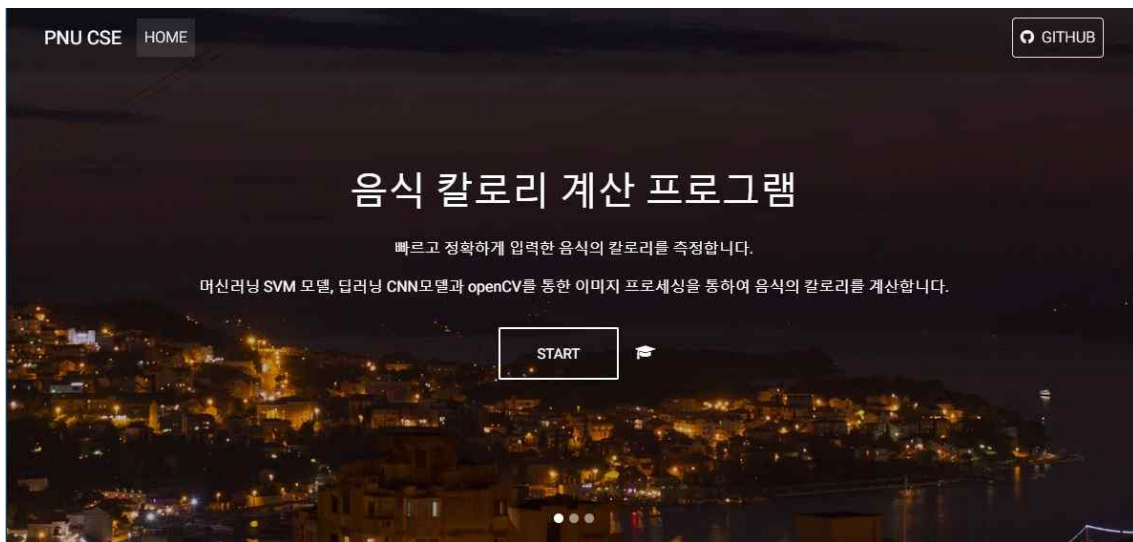
(4) 음식 칼로리 및 영양성분 계산

식약청의 100g당 음식 칼로리와 영양성분 데이터베이스를 활용하여, 위에서 획득한 질량을 통해 음식의 최종 칼로리와 영양성분을 계산한다.

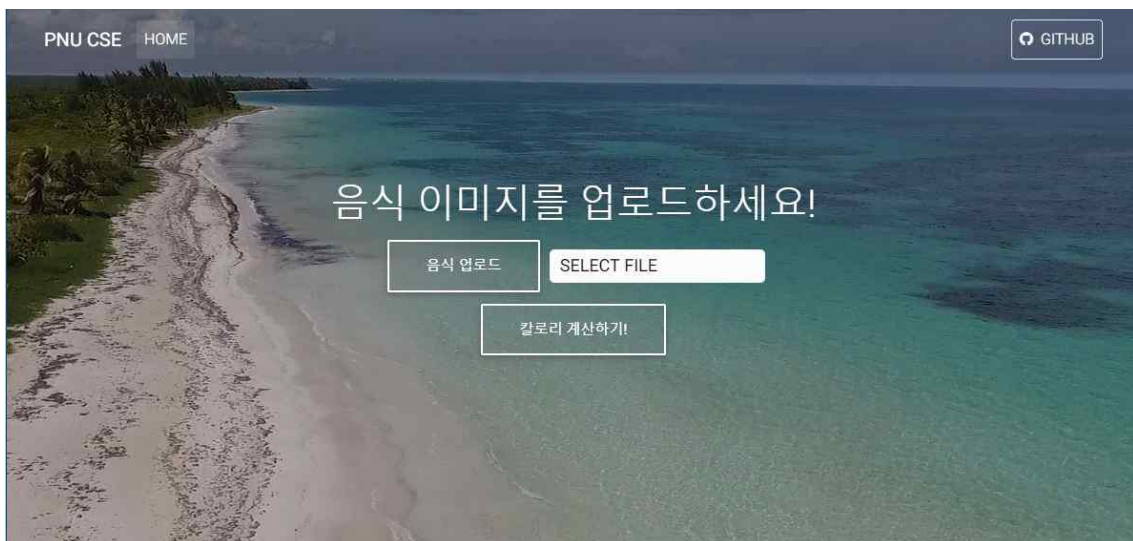
```
calorie_tot = (calorie_100 / 100.0) * mass
carbohydrate = (carbohydrate_dict[int(label)] / 100) * mass
protein = (protein_dict[int(label)] / 100) * mass
fat = (fat_dict[int(label)] / 100) * mass
cholesterol = (cholesterol_dict[int(label)] / 100) * mass
natrium = (natrium_dict[int(label)] / 100) * mass
```

3.7. 칼로리 출력

위 과정을 통해 각각 모양에 따라 최종적으로 계산된 칼로리 및 영양성분을 웹페이지 상에 출력하기 위해 파이썬 기반 마이크로 웹 프레임워크인 Flask를 이용한다. 우선 templates 폴더를 만든 후 웹 페이지의 메인 페이지를 구성하는 HTML 파일 'index.html'와 실제 사용자가 이미지를 넣는 실행 페이지를 구성하는 HTML 파일 'upload.html'를 넣는다.



[그림 59] 메인 페이지 index.html



[그림 60] 실행 페이지 upload.html

메인 페이지 index.html을 웹 상에 나타내기 위해 Jinja2 엔진에서 제공하는 render_template 함수를 이용하여 render_file()과 '호스트주소/'와 라우팅한다.

```
#메인 페이지@app.route('/')
def render_file():
    return render_template('index.html')
```

실행 페이지를 웹 상에 나타내기 위해 '호스트주소/upload'와 upload()를 라우팅 한다. GET방식으로 upload.html 파일 정보를 웹 서버에 보낸다.

```
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    if request.method == 'GET':
        return render_template('upload.html') #upload.html 리턴
```

실행 페이지에서 사용자가 칼로리 및 영양성분을 알고싶은 음식 이미지를 입력할 수 있도록 파일 업로드를 구현한다. 입력한 음식 이미지의 데이터를 POST방식으로 웹 서버에 보낸다.

```
if request.method == 'POST':
    #파일을 선택하지 않았을 시 에러 처리 if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    f = request.files['file']
    if f.filename == '':
        flash('No selected file')
        return redirect(request.url)
    #지정할 경로 + 파일명
    if f and allowed_file(f.filename):
        classes = { #15가지 과일 및 각각의 라벨 }
```

사용자가 선택한 음식 이미지의 경로를 참조하여 그에 대한 저장된 음식 데이터를 불러오기 위해 선택한 이미지를 image_name 변수에 넣은 후 os.path.join 함수를 이용, 기존 디렉터리 경로와 파일 이름을 병합하여 선택한 파일의 경로를 저장한다.

```
image_name = f.filenamebasepath = os.path.dirname(__file__)
file_path = os.path.join(
    basepath, 'static', secure_filename(f.filename))
f.save(file_path)
```


upload 함수는 upload.html 파일 및 계산된 음식의 라벨, 이미지, 질량, 칼로리, 100g당 칼로리, 탄수화물, 단백질, 지방, 콜레스테롤, 나트륨, 알람 메시지, 다운로드 경로 등의 요소들을 리턴함으로써 사용자가 입력한 음식 이미지에 따른 음식 데이터를 upload.html에 보낸다.

```
return render_template('upload.html', label=class_name, img=image_name, mass=mass,
cal=cal,cal_100=cal_100, carbo=carbo, pro=pro, fat=fat, choles=choles, nat=nat, mes=
message, download_path=download
d_path)
```

실행 페이지 내에서 사용자가 폼 제출 시에 POST방식으로 요청 된 음식 데이터를 처리한다. 파일의 이름만 전송되고 데이터는 전송되지 않는 현상을 방지하기 위해 <form>의 enctype의 형식을 'multipart/form-data'로 설정한다.

```
<form method='post' enctype=multipart/form-data>
</form>
```

upload.html 파일에서 사용자가 음식 이미지를 입력하였을 시, 선택한 음식 이미지의 경로를 참조하여 app.py의 upload 함수의 리턴값으로 받은 칼로리 및 영양성분을 웹페이지 상에 표시한다.

```
{% if img %} <div> <table class="b"> <th>음식명</th><th>질
량</th><th>칼로리</th><th>100g당칼로리</th>
<tr>
<td>{{ label }}</td><td>{{ mass }}g</td><td>{{ cal
}}kcal
</td><td>{{ cal_100 }}kcal</td>
</tr>
</table>
<p></p>
<table class="b">
<td>{{ carbo }}g</td><td>{{ pro }}g</td><td>{{ fat }}g</td><td>{{
choles }}mg</td><td>{{ nat }}mg</td>
</tr>
</table> <p></p>
{{ mes }}
</div>{% endif %}
```

3.8. 영양성분 초과 알림 (산업체 멘토링 반영)

사용자가 다이어트나 식단 조절 할 때 프로그램의 활용성을 높이기 위해 알림 설정을 구현한다. 알림 설정의 목적은 프로그램이 측정한 음식의 칼로리 및 영양성분이 일일 섭취 권장량을 초과하였을 때 웹 페이지에 나타나도록 하여 사용자에게 추가적인 정보를 제공하기 위함이다.

우선 식약처에서 정한 성인 기준 일일 섭취 권장량을 코드 상에 명시한다.

```
# 탄수화물, 단백질, 지방, 콜레스테롤, 나트륨 일일 권장섭취량, 칼로리 한끼 권장섭취량
nutrient_dict = {0: 130, 1: 100, 2: 51, 3: 300, 4: 2000, 5: 800}
```

일일 섭취 권장량에 비해 초과되는 영양소가 있을 경우 메시지를 출력한다. 예를 들어 탄수화물이 일일 섭취 권장량을 초과하였을 때 알림 메시지에 탄수화물을 추가하도록 한다. 이런 방식으로 각 영양소에 대한 알림 메시지를 구현한다.

```
if(carbo > nutrient_dict[0]):
    if(len(Alarm_message) < 3):
        Alarm_message = Alarm_message + " 탄수화물 "
    else:
        Alarm_message = Alarm_message + ", 탄수화물 "
```

실제 웹 페이지상에서 일일 섭취 권장량을 초과하는 음식을 입력하였을 때 아래 그림과 같이 알림 메시지가 나타난다.

음식 칼로리 측정 결과

음식명	질량	칼로리	100g당 칼로리
Doughnut	329.09g	1487.49kcal	452kcal

탄수화물	단백질	지방	콜레스테롤	나트륨
167.84g	16.13g	82.27g	62.53mg	1072.83mg

탄수화물, 지방, 한끼칼로리 이(가) 일일 적정섭취량을 초과하였습니다.

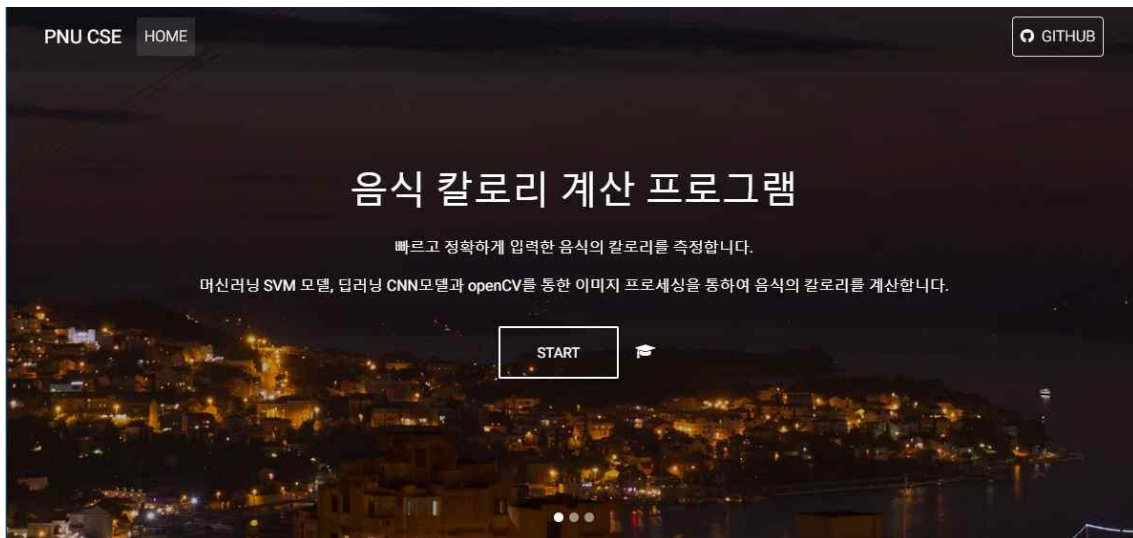
[그림 61] 영양성분 초과시 출력 화면

4. 과제 수행 결과 분석

4.1. 개발 환경

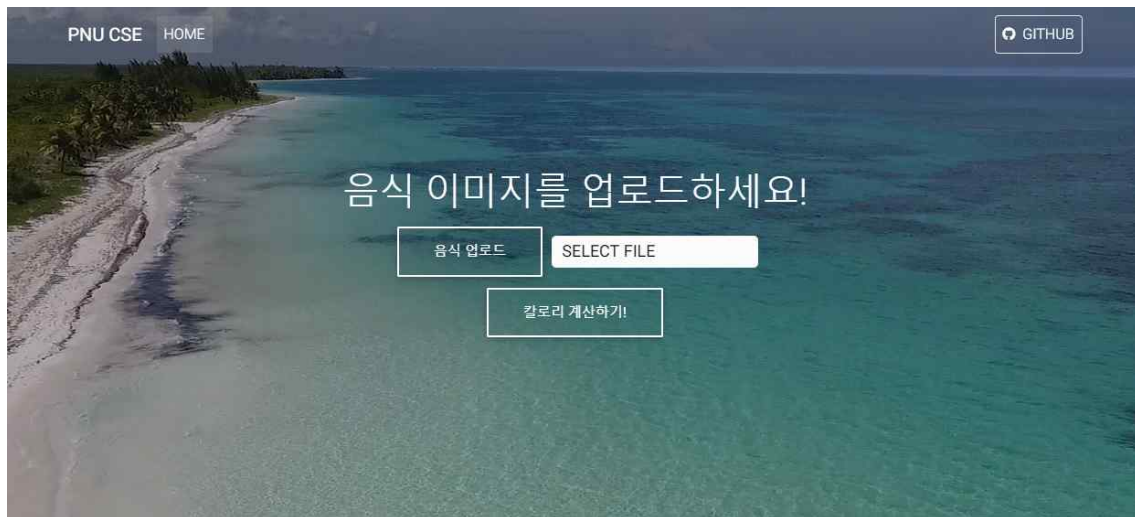
운영체제 : Windows 10 64bit 프로세서
개발 언어 : Python 3.7 64bit
개발 도구 : PyCharm
프레임워크 : Flask 1.1.2
대표 라이브러리 : Tensorflow 2.0.0, Keras 2.3.1

4.2. 개발 결과



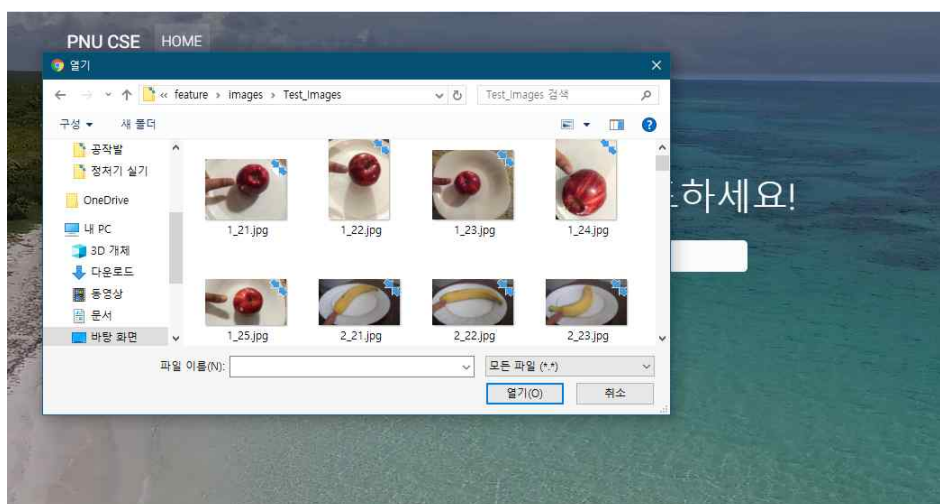
[그림 62] 메인 페이지 index.html

메인 화면 상단에 학부 홈페이지, 프로젝트의 GITHUB 사이트를 연결하는 버튼이 있다. 그리고 웹 메인화면으로 돌아올수 있는 HOME 버튼이 존재한다, 웹 사이트에 대한 간단한 소개문구가 있으며 START 버튼을 클릭하면 칼로리 계산이 필요한 음식 이미지를 업로드 하는 페이지로 전환된다.



[그림 63] 음식이미지 업로드 화면

음식 업로드 버튼을 클릭 하여 파일 선택 창에서 이미지 하나를 선택하여 업로드하고 칼로리 계산하기 버튼을 클릭하여 계산된 칼로리 및 영양성분을 출력한다.



[그림 64] 음식 업로드 버튼 클릭



음식 칼로리 측정 결과

음식명	질량	칼로리	100g당 칼로리
Apple	244.59g	127.19kcal	52kcal

탄수화물	단백질	지방	콜레스테롤	나트륨
34.24g	0.73g	0.49g	0.0mg	2.45mg

500개 이상의 데이터를 활용하여 4% 미만의 데이터 오차율을 보이는 음식 칼로리 계산 프로그램을 개발했습니다.

DOWNLOAD

[그림 65] 칼로리 계산하기 버튼 클릭시 나타나는 화면

이미지를 업로드하고 칼로리 계산하기 버튼을 클릭하면 업로드한 이미지를 좌측에 보여주고, 예측한 음식명을 보여준다. 이미지에서 음식의 부피와 질량을 예측하여 계산한 칼로리와 영양성분을 출력한다.



음식 칼로리 측정 결과

음식명	질량	칼로리	100g당 칼로리
Banana	0.0g	0.0kcal	88kcal

탄수화물	단백질	지방	콜레스테롤	나트륨
0.0g	0.0g	0.0g	0.0mg	0.0mg

500개 이상의 데이터를 활용하여 4% 미만의 데이터 오차율을 보이는 음식 칼로리 계산 프로그램을 개발했습니다.

DOWNLOAD

[그림 66] 손가락이 없는 이미지에 대한 결과 화면

이미지에서 손가락이 없을 경우에는 비교할 대상이 없기 때문에 음식 이미지에 맞는 칼로리를 계산할 수 없기 때문에 음식명과 100G 당 칼로리만 출력한다.



음식 칼로리 측정 결과

음식명	질량	칼로리	100g당 칼로리
Doughnut	329.09g	1487.49kcal	452kcal

탄수화물	단백질	지방	콜레스테롤	나트륨
167.84g	16.13g	82.27g	62.53mg	1072.83mg

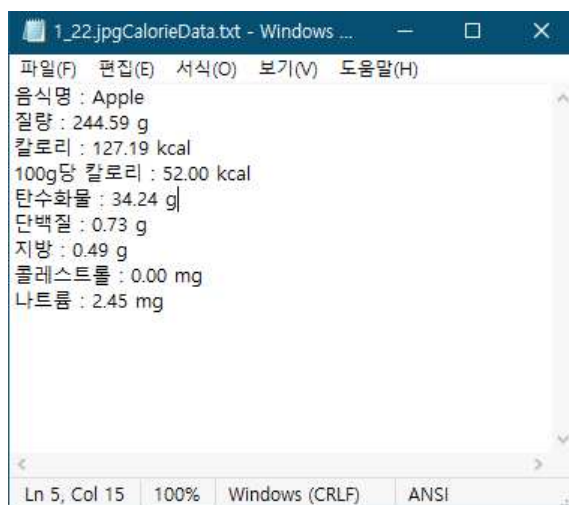
탄수화물, 지방, 한끼칼로리 이(가) 일일 적정섭취량을 초과하였습니다.

500개 이상의 데이터를 활용하여 4% 미만의 데이터 오차율을 보이는 음식 칼로리 계산 프로그램을 개발했습니다.

DOWNLOAD

[그림 67] 권장섭취량 초과 할 경우 알림 문구 추가

칼로리와 영양성분을 추정하여 출력하였는데 그 수치가 권장섭취량을 초과 할경우 그림과 같이 초과 알림 문구를 추가하여 출력한다.



[그림 68] 저장되는 텍스트 파일

DOWNLOAD 버튼을 클릭하면 업로드한 이미지에 대한 음식명과 칼로리 정보, 영양성분 정보에 대한 텍스트 파일을 다운로드 할 수 있다.

5. 결론 및 향후 개선 방향

5.1. 활용 방안

첫 번째 활용 방안은 기존에 인터넷상에서 100g 당 칼로리 및 영양성분을 제공하는 방식에서 음식 칼로리 측정 프로그램을 활용함으로써 사용자는 자신이 알고 싶은 음식의 칼로리 및 영양성분을 보다 정확하게 알 수 있게 된다. 이를 통해 사용자는 이 정보를 활용하여 효과적인 다이어트 및 균형 잡힌 식단을 짤 수 있게 된다. 예를 들어 사용자가 다이어트 중에 특정 음식을 먹었을 때 성인 일일 권장 섭취량이 초과된다면 프로그램에서 알림을 통해 사용자가 그 음식을 먹지 않게끔 유도할 것이다.

두 번째 활용 방안은 사용자마다 각각 입력한 음식의 측정한 영양성분들을 데이터베이스와 연동하여 데이터화한 뒤 오픈 API로 제공한다면 향후 식품업체나 헬스, 피트니스 산업 분야에서 이 데이터를 활용하여 많은 고객 서비스를 제공할 수 있을 것이다. 예를 들어 식품업체에선 향후 경쟁력 확보를 위해 균형 잡힌 영양식품을 만드는데 있어 통계적 자료나 분석을 위해 오픈 API를 참고할 수 있다. 헬스 산업 분야에선 오픈API를 활용하여 헬스장이나 피트니스 센터를 이용하는 고객의 다양한 식단을 짤 수 있을 것이다.

5.2. 한계점

우선 입력한 이미지 내에서 칼로리 계산을 위한 정확한 면적을 측정하기 위해선 접시와 손가락이 필수여야 한다는 점이다. 프로그램에서는 임의로 선정한 기준 물체의 면적에 대비하여 분류한 음식의 면적을 측정한다. 이 과정에서 손가락을 기준으로 삼았기에 필수적으로 이미지 내에 들어가야 한다. 또한 음식 면적을 추출하는 과정에서 가장 큰 윤곽선을 접시로 놓았고 두 번째로 큰 윤곽선을 음식으로 처리했기에 접시가 필수적으로 존재해야 한다.

두 번째 한계점으로는 여러 개의 음식이 있을 경우에는 측정한 면적이 정확하지 않음에 따라 칼로리 및 영양성분 또한 제대로 측정되지 못하는 점이 있었다. 위에 언급한대로 접시를 가장 큰 윤곽선으로 처리하고 두 번째로 큰 윤곽선을 음식의 면적으로 처리하는데 여러 개의 음식, 예를 들면 1개의 사과와

1개의 파스타 있는 경우엔 두 번째로 큰 윤곽선을 처리하는데 어려움이 있었기에 여러 개의 음식의 면적을 측정하지 못한 부분이 있었다.

마지막으로는 웹 애플리케이션으로 출력하는 부분에 있어서 접근성에 한계점이 있었다. 현실에서는 대부분 사용자들은 음식에 대한 영양 성분을 찾기 위해서 컴퓨터나 노트북을 켜는 일은 흔치 않을 것이

5.3. 향후 개선 방향

현재 20개의 음식에 대해서만 음식 예측이 가능하다. 차후에 음식 데이터셋을 많이 확보해서 학습시키면 인식 가능한 음식의 수를 늘릴 수 있다. 또한 현재에는 1개의 음식에 대해서만 구분이 가능하지만 딥러닝 부분에서 기존 CNN을 RNN과 결합하는 쪽으로 좀 더 연구해보는다면 여러 개의 음식 이미지를 구분할 수 있고 그에 따라 위 프로그램 프로세스에 추가하여 접시 안에 여러 음식이 있어도 면적을 측정할 수 있을 것으로 예상 된다.

프로그램을 안드로이드나 iOS 등 모바일 애플리케이션과 연동한다면 기존 웹 애플리케이션 대비 사용자가 프로그램을 활용하기에 더욱 용이해질 것으로 예상된다.

6. 역할 분담 및 개발 일정

6.1. 역할분담

[표 1] 프로젝트 구성원별 역할 분담

이름	내용
강나훈	· 이미지 세그멘테이션 구현 · 딥러닝 알고리즘 구현
박정민	· 칼로리 계산 및 출력 알고리즘 설계 · 머신러닝 알고리즘 구현
예병준	· 웹 애플리케이션 구현 · 음식 이미지에서 색상, 질감, 모양 추출해서 특징 벡터 생성
조원 전체	· 트레이닝셋 음식 이미지 수집 · openCV 라이브러리 분석 및 적용 · 버그수정 및 최적화 · 보고서 작성 및 발표준비

6.2. 개발 일정

[표 2] 프로젝트 개발 일정

6월					7월					8월					9월				
1주	2주	3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	5주
컴퓨터비전, opencv, 머신러닝 공부																			
		멘토의견서 제출			트레이닝셋 구축														
					특징 추출 구현														
								딥러닝 알고리즘 적용											
							중간보고서 작성												
								칼로리계산 구현											
						머신러닝 알고리즘 적용													
								웹 애플리케이션 구현											
												테스트, 보완							
															최적화 및 버그수정				
																최종보고서 작성			
																	발표 준비		

7. 참고문헌

[1]. Liang-yc, “ECUSTFD-resized”

[온라인].<https://github.com/Liang-yc/ECUSTFD-resized->

[2]. 식품의약품안전처, “한국인 영양섭취기준”

[온라인].https://www.foodsafetykorea.go.kr/foodcode/01_03.jsp?idx=12131

[3]. 황선규, “OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝”, 길벗, 2019

[4]. 주성식, 홍성민, “파이썬 웹 프로그래밍:플라스크를 이용한 쉽고 빠른 웹 개발”, 위키북스, 2016

[5]. 오타 미즈히, “(실전!) 딥러닝:텐서플로와 케라스를 이용한 딥러닝 최신 기술 활용 가이드”, 위키북스, 2019

[6]. mdbootstrap, “Full Page Video Carousel”

[온라인].<https://mdbootstrap.com/freebies/jquery/full-page-video-carousel/>

[7]. 안경잡이개발자, “다변인 선형회귀를 이용한 배추 가격 AI 예측하기-플라스크 웹 서버와 웹 디자인 구현”

[온라인].<https://ndb796.tistory.com/130>

[8]. 특허, “딥러닝 알고리즘을 이용한 사물인식을 사용하여 측정 및 소비를 위한 O2O 방식의 칼로리 관리시스템”

[온라인].<https://scienceon.kisti.re.kr/srch/selectPORSrchPatent.do?cn=KOR1020160113386&dbt=KPTN>, 2018

[9]. MathWorks, “객체 인식, 반드시 알아야 할 3가지”

[온라인].<https://scienceon.kisti.re.kr/srch/selectPORSrchPatent.do?cn=KOR1020160113386&dbt=KPTN>