2020년 1학기 “데이터베이스시스템”

팀 프로젝트 최종 보고서

팀 1조

데이터사이언스학과 19512079 박민규

데이터사이언스학과 20510077 백종민

기계설계로봇공학과 19520048 김원준

Image database management using Oracle Blob

1. 주제 선정 배경 및 목표

Oracle에 관하여 숫자나 텍스트 정보가 테이블 형태로 저장되고, SQL을 통해 데이터를 조작할 수 있는 것을 넘어서서 이미지파일을 테이블 형태로 저장하고 관리할 수 있는 방법을 탐구하고자 한다. 추가적인 분석으로 이미지 안의 텍스트 인식을 통해 텍스트를 포함한 이미지와 포함하지 않은 이미지를 분류하려 한다. 구글에 특정 단어에 대한 이미지를 찾으면 이미지 안에 워터마크와 같은 이미지과 관련 없는 텍스트가 포함되어 있는 이미지들을 찾을 수 있다. 그런데 많은 이미지를 다루게 된다면 사용자가 직접 눈으로 텍스트가 들어있지 않은 이미지를 분류해야 한다. 딥러닝을 통해 이미지를 분류할 수 있다면 더 효과적이고 빠르게 이미지를 찾고 관리할 수 있다.

본 프로젝트의 목표는 첫 번째로 이미지 데이터를 Oracle Blob을 이용해 테이블 형태로 저장하고 SQL query로 조작하는 것을 구현하는 것이다. 두 번째로 OpenCV 기술을 활용해 이미지 속 텍스트를 detect하여 첫 번째 과제에 접목시키는 것이다.

2. 연구 내용

2.1 System environment

Os : Window 10, 64bit

Python : 3.7, using pycharm, using cx_oracle

Oracle : Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

Selenium : 3.141.0 version

BeautifulSoup : 4.9.1 version

2.2 Framework

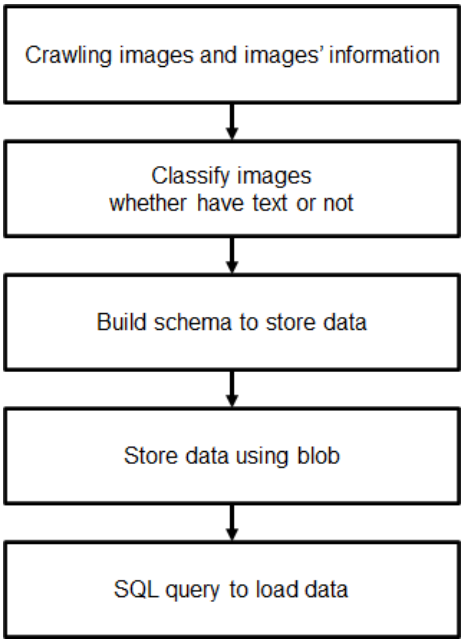

Figure 1. Framework

프로젝트의 Framework는 Figure 1과 같다. 구글 이미지는 검색어에 대한 방대한 이미지를 보여준다. 예를 들어 'Ocean"이란 단어를 검색하면 Ocean과 가장 관련이 높은 이미지를 보여준다. 우린 이 이미지들을 Python crawler를 통해 가져올 것이고 여기에 사용되는 library는 Beautiful soup과 Selenium이 될 것이다. 우리는 검색어보다 상위 주제을 설정하고 이미지를 수집한다. 예를 들어 Nature이란 상위 주제가 있고 그 밑에 Ocean, Tree, Sky 등이 있다. 이미지는 Search_term, title, size_width, size_height, url(source), text_contain의 entity를 가지며 Search_term에 따른 Schema로 구성된다.

우리는 이미지 분류를 실시한다. 이미지는 가끔 불필요한 텍스트를 가지고 있거나 워터마크를 포함하고 있어 이미지를 실제로 사용해야 하는 경우 문제가 된다. 이를 위해 텍스트를 자동으로 감지하고 텍스트가 없는 이미지를 따로 분류하여 저장 할 필요가 있다. 이를 위해 본 프로젝트는 이미지 opening, contour 모델을 활용해 이미지 속 텍스트를 탐지하여 이미지를 분류한다.

보통 각 entity마다 들어가는 data의 type은 string이나 number와 같다. Image는 다른 데이터와 다르게 크기가 매우 커 Oracle Blob을 사용해야 테이블 형태로 저장 할 수 있다. 저장된 데이터는 사용자의 목적에 따라 SQL query를 통해 불러올 수 있게 된다.

2.3 Crawling

크롤링(crawling), 스크레이핑(scraping)이라고도 부르며 웹 페이지를 그대로 가져와서 데이터를 추출하는 행위이다. 이러한 행위를 하는 소프트웨어를 크롤러(crawler)라고 부른다.
전세계적으로 많은 사람들이 인터넷 서칭(searching)을 하기 위해 사용하는 프로그램 검색 엔진 안에도 크롤러가 존재한다. 유명한 구글에도 사용자가 검색을하면 크롤링을 하는 크롤러가 존재한다. 이러한 검색 엔진과 같은 사이트에서는 최신 상태를 유지하기 위해 웹 크롤링을 한다.

Figure 2. google crawler

✓ 크롤링한 정보를 상업적으로 사용하려면 정보를 제공하는 측의 허가 필요합니다.

파이썬에서는 크롤링을 도와주는 다양한 라이브러리들이 존재한다. Beautifulsoup, Selenium, Scrapy에 대해 장단점을 살펴보고 프로젝트 필요한 라이브러리를 사용한다.

Beautifulsoup
- HTML, XML 파일의 정보를 추출해내는 라이브러리
- python 내장 모듈인 requests 혹은 urllib을 이용해 HTML를 다운로드 받고, beautifulsoup로 데이터를 추출하는 방식

Selenium
- 웹 자동화 테스트(버튼 클릭, 스크롤 조작 등)에 사용되는 프레임워크
- 인터넷브라우저를 통해 크롤링하는 개념, 실제 보여지는 웹페이지의 전부를 가져올 수 있다.

Scrap
- 크롤링을 위해 개발된 프레임워크
- 미들웨어, 파이프라인 등등 다양한 기능들과 플러그인을 할 수 있습니다.
- 유료입니다.

Code 1. 웹사이트 **url**

```
baseUrl = "https://www.google.com/search"
webDriver = "./chromedriver.exe"
params = {
      "q": searchItem,
       "tbm": "isch",
       "sa": "1",
       "source": "lnms&tbm=isch"
   }
url = baseUrl+"?"+urllib.parse.urlencode(params)
```

본 프로젝트에서는 크롬을 이용하여 이미지를 검색하고, 이 브라우저를 제어를 하기 위해 webdriver라는 API(chromedriver.exe)를 사용한다. 이는 브라우저를 직접 동작시킴으로써, 비동기적으로 load되는 컨텐츠들을 가져오기 위함이다. 'params' 에서의 "q" 부분에 우리가 찾고자 하는 이미지 검색어('searchItem')을 입력하면 검색된 이미지 url을 얻어올 수 있다.

Code 2. 단어 검색

```
browser = webdriver.Chrome(webDriver)    # 구동 브라우저 = 크롬 (옵션 사용 안할 경우)
browser.implicitly_wait(1)
browser.get(url)                                  # 브라우저 접속
html = browser.page_source                 # 검색된 페이지의 html 불러오기
browser.implicitly_wait(1)
```

'browser'라는 webdriver.Chrome(webDriver)에 대한 객체를 생성한 후, Code 1에서 지정한 검색어를 검색할 수 있도록 브라우저를 구동한다. 그 후 구동 된 브라우저의 load 된 페이지의 전체 html을 불러온다. 검색어(searchItem)를 'tree'로 하여 작동 여부를 확인한다.(Figure 3)

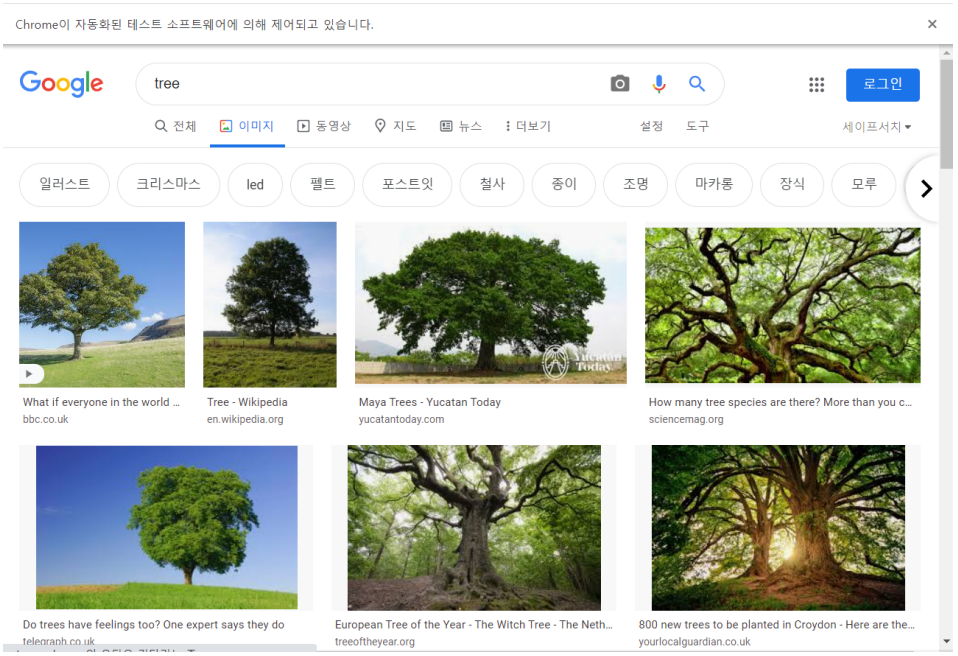

Figure 3. Search ‘tree’ image

Code 3. 이미지 개수 확인

```
# 검색된 페이지에 대한 이미지 갯수 가져오기
soup_temp = BeautifulSoup(html, 'html.parser')
img4page = len(soup_temp.findAll("img"))          # 한번의 parsing 에 이미지가 얼마나 있는지 이미지 갯수 파악
print("Img Num: ", img4page)

# 페이지 다운 (이미지를 계속 찾기 위함)
element = browser.find_element_by_tag_name("body")

imgCnt =0
while imgCnt < size * img4page:
    element.send_keys(Keys.PAGE_DOWN)  # Page down key event
    rnd = random.random()
    print(imgCnt)
    imgCnt+=img4page
    try:
        browser.find_element_by_class_name("mye4qd").click()
    except:
        time.sleep(rnd)
```

Code 3의 이미지 개수 확인은 대략적으로 이미지 개수를 파악하기 위함이다. 한 페이지에 로드 되는 이미지의 갯수를 파악한 후 스크롤을 내리도록 한다.(이는 편의상 개수를 확인하는 것이지 실제 이미지와는 상이할 수 있다.) 스크롤을 내리기 위해 PAGE_DOWN의 키 이벤트를 사용하였다.

Code 4. 이미지 크롤링

```
# 스크롤 다 내린 후 html 재파싱
html = browser.page_source
img_soup = BeautifulSoup(html, 'html.parser')
small_img = img_soup.find_all('img', attrs={'class': 'rg_i Q4LuWd'})

small_img_urls = []  # small image urls
for img in small_img:
    try:
        small_img_urls.append(img['src'])
    except:
        small_img_urls.append(img['data-src'])

browser.close()
```

스크롤을 내리면서 로드 되지 않았던 이미지들까지 포함된 html을 다시 저장한 후 파싱한다. html에서 이미지 파일은 'src'와 'data-src'에 주소표기 되어있다. 따라서, 파싱 후 html의 class : 'rg_i Q4LuWd' 을 찾은 후 그 클래스의 'src' 혹은 'data-src' 부분의 링크를 small_img_url의 리스트에 추가한다.

Code 5. 이미지 저장

```
## 저장 폴더 생성 및 이미지 저장
saveDir = folder + searchItem   # 이미지 저장 디렉토리
try:
    if not(os.path.isdir(saveDir)):
        os.makedirs(os.path.join(saveDir))
except OSError as e:
    if e.errno != e.errno.EEXIST:
        print("Failed to create directory!!!!!")
        raise

# 이미지 저장
print("\r\n")
print(" ==================Progress (Small image save)
========================")

for i, src in zip(tqdm(range(len(small_img_urls))), small_img_urls):
    try:
        urllib.request.urlretrieve(src, saveDir + "/" + str(i) + ".jpg")
```

```
        except:
            continue

    print('Job is done')

    return img_soup, url, searchItem, len(small_img_urls)
```

이미지의 텍스트를 분석하기 위해 로컬에 이미지를 저장한다. 이미지를 저장하기 위한 폴더는 try문을 통해 생성한다. 폴더명은 searchItem의 이름으로 생성된다. 기존의 폴더가 생성 되어 있을 경우 그곳에 저장하고, 폴더가 없을 경우에는 폴더를 생성한 후 이미지를 저장한다.

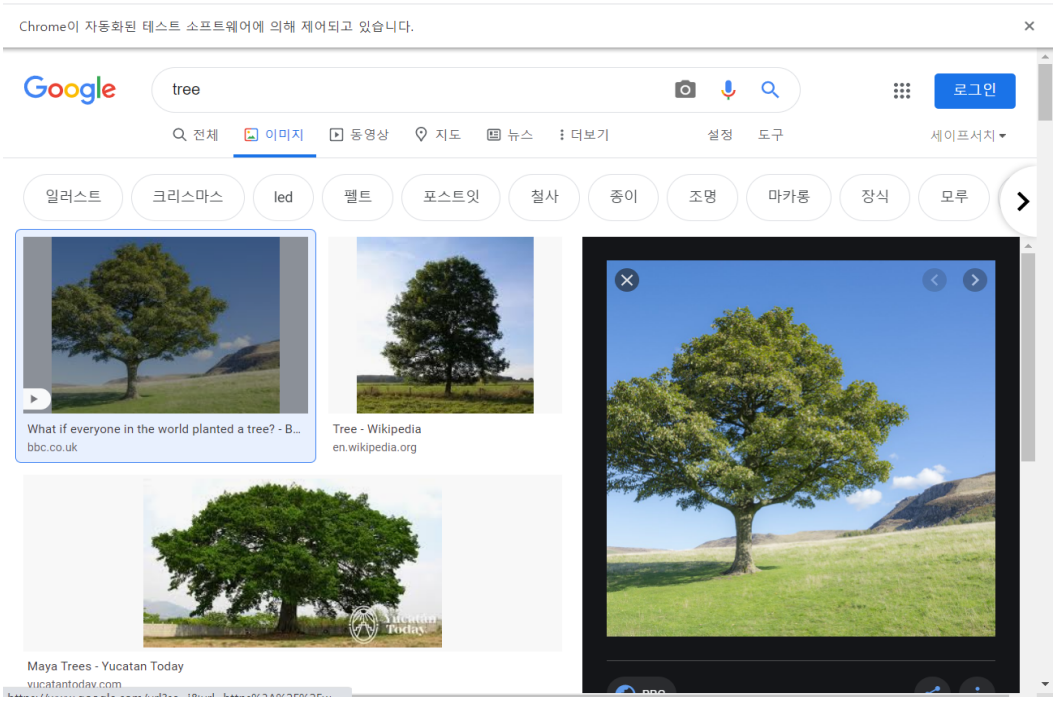

Figure 4. Crawl big size image

Figure 3와 Figure 4를 비교해보면 오른쪽 검은색 부분이 다르다는 것을 알 수 있다. 그 이유는 Figure 3를 통해 이미지를 크롤링 해본 결과, 이미지의 해상도가 좋지 않다는 것을 확인할 수 있었다. 사용자는 보통 고해상도의 이미지를 원하기 때문에 우리가 필요한 이미지는 그림을 클릭해서 보여지는 검은색 부분의 이미지이다. 하지만 고해상도 일수록 text detection 정확도가 떨어지는 것을 확인했다. 그래서 우리는 저해상도 이미지로 text 포함 유무에 대한 정보를 고해상도 이미지에 그대로 적용시켰다.

두 방법(저해상도,고해상도 크롤링)의 장단점이 있었다. 저해상도 이미지의 경우 스크롤을 내린 후, 하나의 html을 통해 빠르게 이미지를 크롤링을 할 수 있었다. 반면 고해상도 이미지를 가져오기 위해서는 스크롤을 내린 후, 얻은 html에서 이미지 하나씩 클릭하고 고해상도 이미지가 로딩되는 시간을 기다리고 다시 html 소스를 받아야 했다.

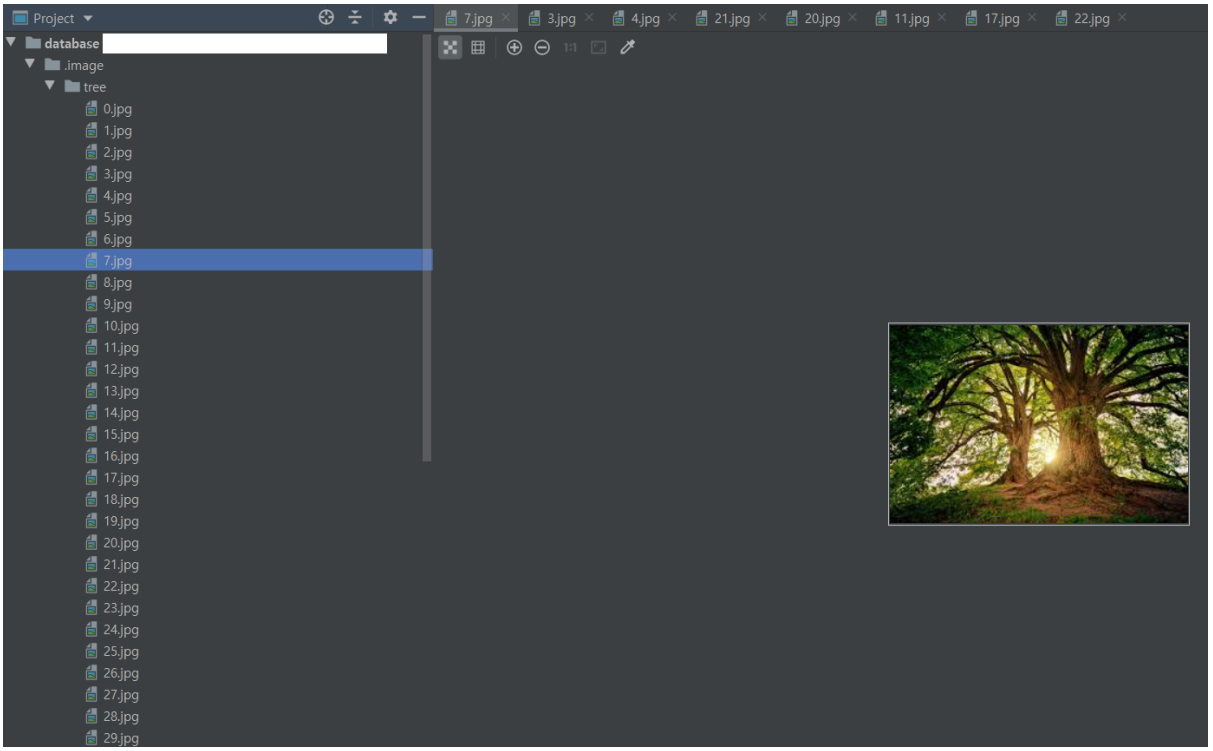

Figure 5. Image crawling outcomes

Figure 5는 크롤링 코드를 작동 시키고 난 후, 이미지가 제대로 저장이 되어있는지 최종적으로
확인을 해보았다. database파일 안에 .image 폴더를 생성하여 검색한 단어(tree)에 맞춰 다시 tree폴더가 생성되고 이미지들이 저장 된 것을 볼 수 있다.

2.4 Classification

텍스트를 포함한 '책상' 이미지는 CNN으로 바로 학습시키면 책상으로 분류하고 이미지 속에 있는 텍스트는 무시하게 된다. 그래서 이미지를 학습시키기 전에 텍스트의 공간을 떼어내는 작업이 우선적으로 진행되어야 한다. 보통 텍스트는 경계선을 뚜렷하게 가지고 있기에 영상처리 기법인 Contour를 사용하게 된다. OpenCV는 Contour를 찾아주는 findContours()를 제공하는데 이러한 방법으로 Text Contour를 추출할 수 있다(Figure 6)[5].

Figure 6. Contour Example

이미지를 흑백으로 이진화 하여 Contour를 추출해도 이미지에는 잡영이 생각보다 많기 때문에 여러 잡영 제거 과정을 거쳐야 한다(Figure 7).

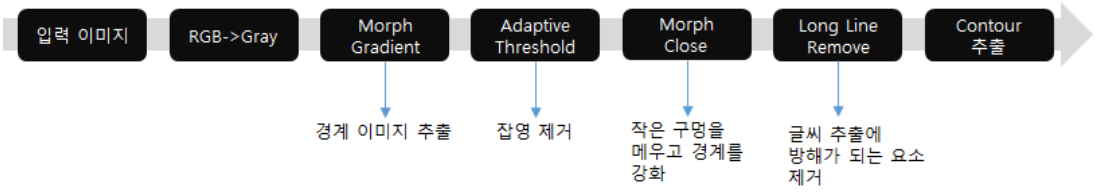

Figure 7. Contour Procedures

이렇게 이미지에 대한 전처리가 끝난 후 이미 주어진 CV2 모델에 따라 이미지의 텍스트 contour를 감지하여 이미지에 텍스트가 포함돼 있으면 1, 포함하지 않으면 0으로 구분하여 튜플을 생성한다. 감지된 이미지의 텍스트는 아래와 같다(Figure 8).

Figure 8. Text Detection Outcomes(Search_term = ‘lion’)

이미지 크기가 다를 때 (size_width, size_height) 이미지 처리에 대한 문제점을 아직 겪어보지 않았지만 이 문제를 해결해야 Contour 처리가 원활하게 진행될 것으로 예상된다. 또한 이미지 안의 텍스트가 불필요하지 않은 경우에 대한 분류기준은 이 프로젝트의 수준을 한참 벗어나는 수준이기에 한계점으로 남지 않을까 예상된다.

Code 4. text detection - morphology

```
def text_detect(image_path, east_path):
 # load the input image and grab the image dimensions
 image = cv2.imread(image_path)
 orig = image.copy()

 # opening(removing noise) : erosion followed by dilation
 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
 image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

 (H, W) = image.shape[:2]

 # set the new width and height and then determine the ratio in change
 # for both the width and height
 (newW, newH) = (320, 320)
 rW = W / float(newW)
 rH = H / float(newH)

 # resize the image and grab the new image dimensions
 image = cv2.resize(image, (newW, newH))

 (H, W) = image.shape[:2]
```

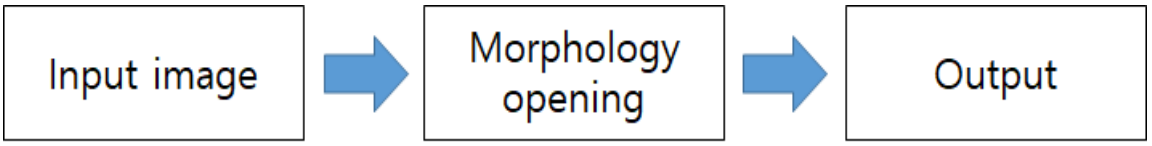

Figure 9. Image preprocessing using project

Code 6.의 단계에서 이미지의 텍스트를 감지한다. 크롤링 후 검색된 이미지들은 로컬에 임시 저장하게 되는데 이 이미지들의 텍스트를 탐색하기 위해 이미지를 로드한다. (원본 이미지는 변수 'orig'에 백업한다.) 텍스트를 분별하기 위해 전처리 기법으로 모폴로지 알고리즘을 사용한다. 마스크는 사각형의 2x2 커널을 사용하였으며, opening 기법을 사용하여, 주위의 노이즈(positive noise)를 제거한다. 검색된 이미지의 크기는 각각 다르므로 이를 균일화 시켜줄 필요가 있다. 따라서 원본이미지의 width와 height의 값을 가져온 뒤, 320x320의 크기로 이에 대한 비율을 구한 후 이미지 크기를 resizing 해준다.

기존의 Figure 7. Contour Procedures의 결과값이 생각보다 좋지 않게 나와서 원본이미지에 positive noise만 제거하는 간단한 전처리(Figure 9)만 한다.

Code 5. text detection - RNN

```
# define the two output layer names for the EAST detector model that
# we are interested -- the first is the output probabilities and the
# second can be used to derive the bounding box coordinates of text
layerNames = [
    "feature_fusion/Conv_7/Sigmoid",
    "feature_fusion/concat_3"]

# load the pre-trained EAST text detector
# print("[INFO] loading EAST text detector...")
net = cv2.dnn.readNet(east_path + '/frozen_east_text_detection.pb')

# construct a blob from the image and then perform a forward pass of
# the model to obtain the two output layer sets
blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
                             (123.68, 116.78, 103.94), swapRB=True, crop=False)
start = time.time()
net.setInput(blob)
(scores, geometry) = net.forward(layerNames)
end = time.time()

# show timing information on text prediction
print("[INFO] text detection took {:.6f} seconds".format(end - start))

# grab the number of rows and columns from the scores volume, then
# initialize our set of bounding box rectangles and corresponding
# confidence scores
(numRows, numCols) = scores.shape[2:4]
rects = []
confidences = []

# loop over the number of rows
for y in range(0, numRows):
    # extract the scores (probabilities), followed by the geometrical
    # data used to derive potential bounding box coordinates that
    # surround text
    scoresData = scores[0, 0, y]
    xData0 = geometry[0, 0, y]
    xData1 = geometry[0, 1, y]
    xData2 = geometry[0, 2, y]
    xData3 = geometry[0, 3, y]
    anglesData = geometry[0, 4, y]

    # loop over the number of columns
```

```
    for x in range(0, numCols):
        # if our score does not have sufficient probability, ignore it
        if scoresData[x] < 0.5:
            continue

        # compute the offset factor as our resulting feature maps will
        # be 4x smaller than the input image
        (offsetX, offsetY) = (x * 4.0, y * 4.0)

        # extract the rotation angle for the prediction and then
        # compute the sin and cosine
        angle = anglesData[x]
        cos = np.cos(angle)
        sin = np.sin(angle)

        # use the geometry volume to derive the width and height of
        # the bounding box
        h = xData0[x] + xData2[x]
        w = xData1[x] + xData3[x]

        # compute both the starting and ending (x, y)-coordinates for
        # the text prediction bounding box
        endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
        endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
        startX = int(endX - w)
        startY = int(endY - h)

        # add the bounding box coordinates and probability score to
        # our respective lists
        rects.append((startX, startY, endX, endY))
        confidences.append(scoresData[x])

# apply non-maxima suppression to suppress weak, overlapping bounding
# boxes
boxes = non_max_suppression(np.array(rects), probs=confidences)

    # loop over the bounding boxes
roi_all = []
for index, (startX, startY, endX, endY) in enumerate(boxes):
    # scale the bounding box coordinates based on the respective
    # ratios
    startX = int(startX * rW)
    startY = int(startY * rH)
    endX = int(endX * rW)
    endY = int(endY * rH)
    roi = orig[startY:endY, startX:endX]

    roi_all.append(roi)

return roi_all
```

Code 7에서는 RNN을 이용하여 텍스트를 감지하도록 한다. RNN은 기본적으로 Figure 10의 구조로 되어 있으며, 흔히 음성, 문자 등의 순차적으로 나열되어 있는 데이터를 처리하는데 적합한 모델중 하나이다. 이 프로젝트에서는 RNN의 신경망 모델인 'frozen east text detection' 의 모델을 사용하여 텍스트를 찾아낸다. 따라서, 네트워크에 'frozen east text detection.pb'파일을 로드하고 입력 이미지를 준비하기 위해 opencv의 dnn 함수 중 하나인 blobFromImage를 사용한다. 여기에 들어가는 매개변수는 이미지, 각 픽셀 값의 배율(scale factor), 이미지의 크기(width, height), 각 RGB에 해당하는 mean subtraction 값, RGB swap 여부(opencv는 채널을 BGR로 사용, Tensorflow는 RGB형식을 사용하기 때문에 B와 R의 채널값을 바꿔준다)이다.

이미지의 텍스트를 예측하기 위해 forward pass를 진행한다. layer의 activation function은 sigmoid를 사용한다. 또한, 2번째 레이어의 concat_3는 텍스트의 경계 좌표를 도출하기 위해 사용한다.

이제 입력 이미지를 루핑시킨 후 scoresData 의 점수에 의해 필터링한 후 높은 확률의 값들에 대한 경계 값을 가져온다. Figure 11.은 '0'의 검색어를 지정한 후 크롤링 된 이미지를 텍스트 감지한 결과 값이다.

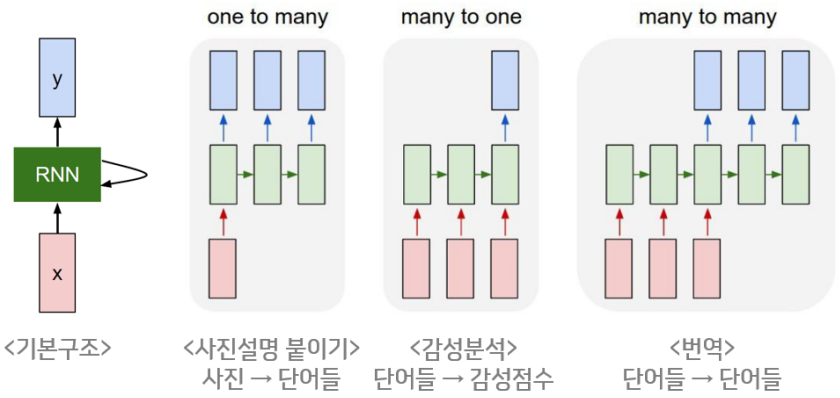

Figure 10. RNN basis structure

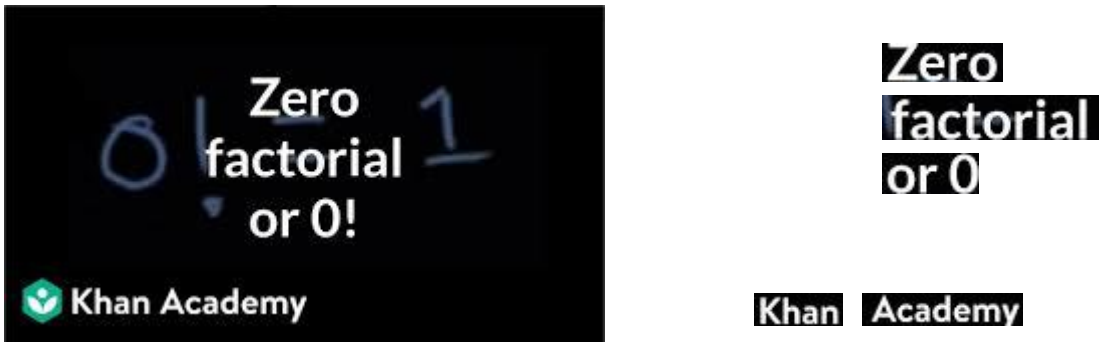

Figure 11. Text detection outcomes example

2.5 Schema

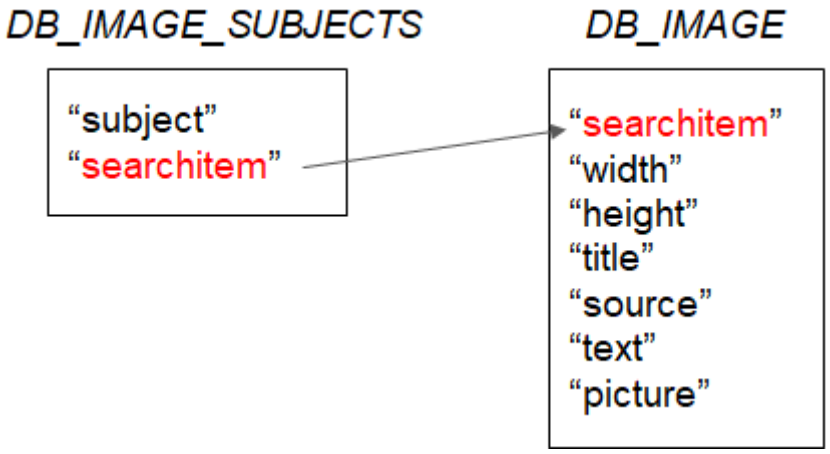

Figure 12. Schema Constraint

Schema는 Figure 15와 같이 구성된다. Subject은 이미지의 주제를 의미하는데 Nature, Animal, Flower 이고, 그에 따른 searchitem은 Nature(tree, ocean, sky, rain, sunset), Animal(sheep, lion, elephant, bird, giraffe), Flower(rose, forsythia, rose of sharon, tulip, daisy)와 같다. Name은 이미지의 이름과 같은데, 예를 들어 ocean을 검색해서 나온 하나의 이미지의 width와 height는 이미지의 픽셀 크기를 의미하며, title은 'Oceans Are Warming Faster Than…'이다. source는 이미지의 출처로 url 값을 나타낸다. tex는 분류모델에 따라 text를 포함하면 1, 포함하지 않으면 0으로 나타낸다. picture은 실제 이미지파일을 blob 형태로 가지고 있다.

Code 6. 외래키 설정

```
def alter_table():
    connect = cx_Oracle.connect("system", "oraclepractice", "localhost/orcl")
    cursor = connect.cursor()
    query = "ALTER TABLE DB_IMAGE \
            ADD CONSTRAINTS fk_img FOREIGN KEY(IMG_SEARCHITEM) \
            REFERENCES DB_IMAGE_SUBJECTS(IMG_SEARCHITEM) ON DELETE
CASCADE"
    cursor.execute(query)
```

두개의 테이블 DB_IMAGE와 DB_IMAGE_SUBJECTS가 데이터를 가지고 있다면 외래키를 설정함으로써 부모, 자식 관계를 표현할 수 있다. ON DELETE CASCADE를 사용하여 DB_IMAGE_SUBJECTS의 searchitem이 사라진다면 DB_IMAGE에서도 자동으로 삭제된다.

2.6 Store data using BLOB

Oracle에서는 세 가지 LOB 데이터 유형(BLOB, CLOB, NCLOB)이 있다. LOB는 TEXT, 그래픽, 이미지, 비디오, 사운드 등 구조화 되지 않은 대형 데이터를 저장하는데 사용한다. 일반적으로 테이블에 저장되는 구조화된 데이터들은 크기가 작지만, 멀티미디어 등 비구조화된 데이터들이 데이터가 큰 비중이 있다. 이러한 크기가 큰 데이터는 DB에 저장하기 힘들다.

LOB 타입 종류 구별

CLOB
- 길이가 긴 문자를 다루기 위한 타입

BLOB
- 이미지, 동영상, MP3 등, 이진 대형 개체

NCLOB
- 내셔널 문자 대형 객체(National Character Set)

BFILE
- OS에 저장되는 이진 파일의 이름과 위치를 저장

이미지 데이터를 사용하기 위해 BLOB 데이터 유형을 사용할 것이다. BLOB은 이진 데이터를 일반적으로 바이트 스트림 또는 버퍼로 표시한다. 보통 하나의 데이터는 Python의 메모리가 버텨주는 한 1GB까지 지원한다.

[1]은 실제 Python에서 Blob을 사용할 수 있는 예제를 보여준다. 간단한 코드로 이미지를 테이블 형태로 저장하고 불러올 수 있는 것을 알 수 있다. 우리는 간단한 Schema를 구성했는데 Blob을 사용함에 따라 Schema가 약간씩 변동 될 수 있다는 점을 인지하고 있다.

Code 7. 데이터베이스 접속

```
db_searchItem = searchItem
db_width = width
db_height = height
db_url = url

# 오라클 접속
# connect("사용자 ID", "Password", "server")
connect = cx_Oracle.connect("system", "oraclepractice", "localhost/orcl")
cursor = connect.cursor()

# 버전 확인 쿼리
query = "select * from PRODUCT_COMPONENT_VERSION"
cursor.execute(query)
print("Version CHECK ============================")
version_result = cursor.fetchall()                    # fetchall = read 결과 확인
print(version_result)
print("=========================================")
```

Code 8. DB_IMAGE 테이블

```
  # 테이블 존재 확인 후 없으면 테이블 생성
  if table_existence_result == 0:
    query_table = "create table DB_IMAGE(\
                          IMG_SEARCHITEM varchar2(50), \
                          IMG_WIDTH number, \
                          IMG_HEIGHT number, \
                          IMG_TITLE varchar2(500), \
                          IMG_SOURCE clob, \
                          IMG_TEXT number, \
                          IMG_PICTURE blob)"
  cursor.execute(query_table)
```

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	IMG_SEARCHITEM	VARCHAR2(50 BYTE)	Yes	(null)	1	(null)
2	IMG_WIDTH	NUMBER	Yes	(null)	2	(null)
3	IMG_HEIGHT	NUMBER	Yes	(null)	3	(null)
4	IMG_TITLE	VARCHAR2(500 BYTE)	Yes	(null)	4	(null)
5	IMG_SOURCE	CLOB	Yes	(null)	5	(null)
6	IMG_TEXT	NUMBER	Yes	(null)	6	(null)
7	IMG_PICTURE	BLOB	Yes	(null)	7	(null)

Figure 13. DB_IMAGE table

검색한 단어의 이미지를 크롤링하면서 단어도 저장되게 만들었다. 크롤링 과정에서의 이미지 속성을 Column 명으로 설정하고 DB_IMGAE의 이름을 가진 테이블을 만든다(Figure 13).

Code 9. DB_IMAGE_SUBJECTS

```
def create_table():
    connect = cx_Oracle.connect("system", "oraclepractice", "localhost/orcl")
    cursor = connect.cursor()

    # DB_IMAGE_SUBJECTS테이블 만들기
    query = "create table DB_IMAGE_SUBJECTS(\
                          SUBJECT varchar2(50),\
                          IMG_SEARCHITEM varchar2(50) PRIMARY KEY)"
    cursor.execute(query)



    # DB_IMAGE_SUBJECTS값
```

```
    subjects = ['Nature', 'Animal', 'Flower']
    search_chr = [['tree', 'ocean', 'sky', 'rain', 'sunset'],
                  ['sheep', 'lion,', 'elephant', 'bird', 'giraffe'],
                  ['rose', 'forsythia', 'rose of sharon', 'tulip', 'daisy']]

    # DB_IMAGE_SUBJECTS값 넣기
    for subj, search_chr in zip(subjects, search_chr):
        for i in search_chr:
            query2 = "insert into DB_IMAGE_SUBJECTS(SUBJECT, IMG_SEARCHITEM)
values (:subject,:search)"
            subject = subj
            search = i
            cursor.execute(query2, (subject, search))
            connect.commit()
```

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
SUBJECT	VARCHAR2(20 BYTE)	Yes	(null)	1	(null)
IMG_SEARCHITEM	VARCHAR2(20 BYTE)	No	(null)	2	(null)

	SUBJECT	IMG_SEARCHITEM
1	Nature	Tree
2	Nature	Ocean
3	Nature	Sky
4	Nature	Rain
5	Nature	Sunset
6	Animal	Sheep
7	Animal	Lion,
8	Animal	Elephant
9	Animal	Bird
10	Animal	Giraffe
11	Flower	Rose
12	Flower	Forsythia
13	Flower	Rose of Sharon
14	Flower	Tulip
15	Flower	Daisy

Figure 14. DB_IMAGE_SUBJECTS table

Code 10. 데이터베이스 저장

```
  # # 저장 폴더에서 img 가져온 후 데이터 인코딩(바이너리)
img_file_list = os.listdir(folder + "{}/".format(db_searchItem))
paths = []
for h in img_file_list:
```

```
    paths.append(folder + "{}/".format(db_searchItem) + h)
for k, image in enumerate(paths):
    im = Image.open(image)
    buffer = BytesIO()
    if im.mode != 'RGB':
        im = im.convert('RGB')
    im.save(buffer, format='jpeg')
    img_str = base64.b64encode(buffer.getvalue())
    query_insert = "INSERT INTO DB_IMAGE(IMG_SEARCHITEM, IMG_WIDTH,
IMG_HEIGHT, IMG_TITLE, IMG_SOURCE, IMG_TEXT, IMG_PICTURE) VALUES (:item,
:width, :height, :title, :clob, :text, :blob)"
    cursor.execute(query_insert, (db_searchItem, db_width[k], db_height[k], db_title[k],
db_source[k], db_text[k], img_str))
    connect.commit()
```

	IMG_SEARCHITEM	IMG_WIDTH	IMG_HEIGHT	IMG_TITLE	IMG_SOURCE	IMG_TEXT	IMG_PICTURE
1	lion	926	616	How we arrived at a ...	https://theconversa...	0	(BLOB)
2	lion	1000	582	How we arrived at a ...	https://theconversa...	0	(BLOB)
3	lion	1024	576	BIA The Lion King 1 ...	https://insp.ngo/ko...	0	(BLOB)
4	lion	1200	900	Lion attack in Austr...	https://www.bbc.com...	1	(BLOB)
5	lion	750	1334	Lion Wikipedia	https://en.wikipedi...	0	(BLOB)
6	lion	1900	1263	Strong lion wallpape...	https://www.pintere...	0	(BLOB)
7	lion	590	390	African lion facts ...	https://www.nationa...	0	(BLOB)
8	lion	907	510	Lion Leo Panthera ...	http://www.krugerpa...	0	(BLOB)
9	lion	907	510	Lion Leo Panthera ...	http://www.krugerpa...	0	(BLOB)
10	lion	1600	900	Lions The Uniquely ...	https://www.livesci...	0	(BLOB)

Figure 15. DB_IMAGE table contents

Code 11의 경우, 첫 번째 Crawling으로 저장되어 있는 이미지 데이터를 가져오는 작업과 함께, 오라클 데이터 베이스에 이미지 속성을 추출한 값들을 저장 되도록 작성된 코드이다. FOR문을 통해 INSERT INTO 구문을 여러번 작성되는 것을 확인 가능하다.

2.7 SQL query to load data

Code 11. Text가 없는 이미지 불러오기

```
cursor.execute("select IMG_SEARCHITEM, IMG_WIDTH, IMG_HEIGHT, IMG_TITLE,
IMG_PICTURE from DB_IMAGE where IMG_TEXT = 0 and IMG_SEARCHITEM =
'{}'".format(db_serchItem))
```

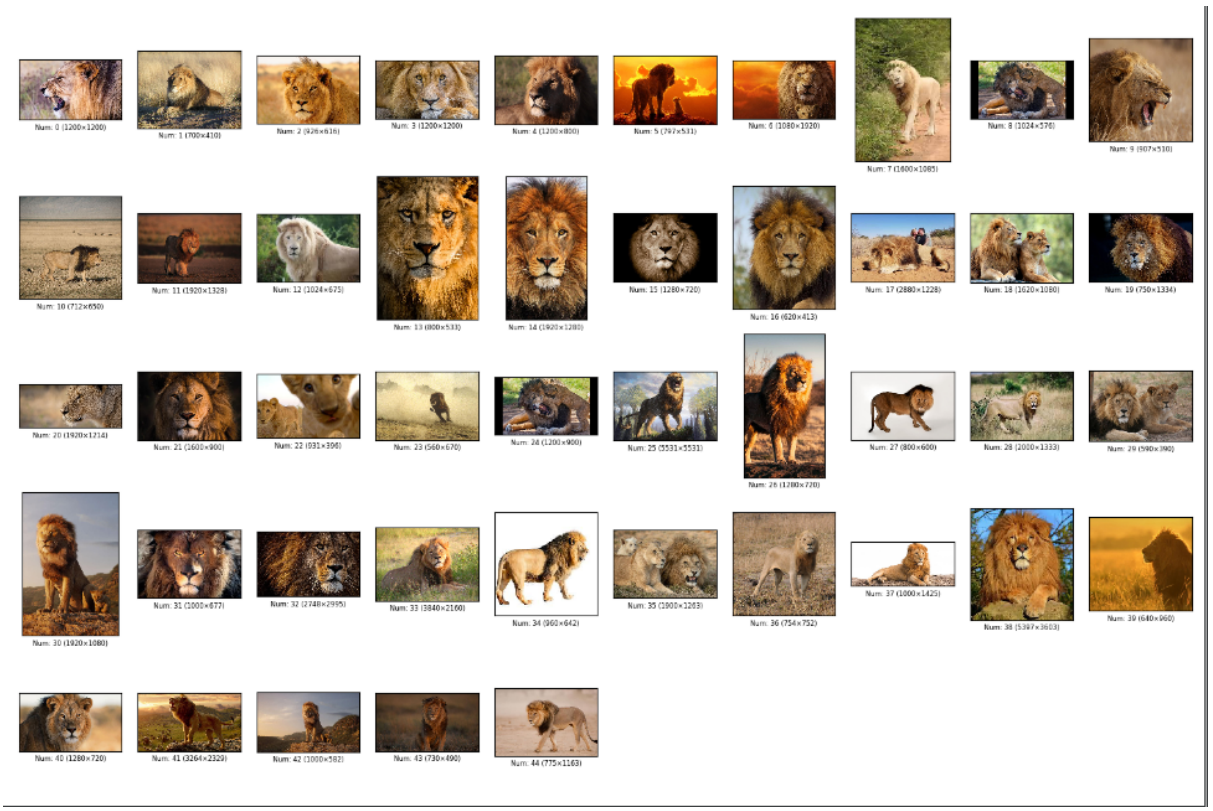

Figure 16. Image ‘lion’ outcomes without text

Figure 16을 보면 text를 포함하지 않는 이미지만 출력된 것을 확인 할 수 있다. 사용자는 용도에 따라 이미지를 따로 불러올 수 있으며, 이미지에 대한 정보를 SQL query를 통해 쉽게 얻을 수 있다.

3. 결론(Conclusion)

3.1 프로젝트 결과 및 토의

- 중간 점검 전

프로젝트의 전체 방향과 목표 설정이 모든 팀원들의 만장일치로 원활하게 진행 되었다. 프로젝트에 대해 의논을 하면서 발생할 수 있는 문제점을 예측할 수 있었다. Blob은 배운 수업에서 벗어나는 확장개념인 점과 텍스트 분류를 위한 딥러닝 분석 난이도가 낮지 않은 점이다. 또한 딥러닝 분석시 텍스트가 이미지에 필요하거나 불필요 하지 않은 경우 분류 기준이 모호해질 것으로 예상된다.

그러나 새로운 데이터 형식을 다룬다는 것은 데이터베이스 시스템 수업에 큰 기여를 할 수 있다는 점에서 이미지 데이터를 사용하기로 했다. 그리고 추가적인 분류 분석의 효과가 미미한 경우 이미지 데이터 관리에 더 초점을 맞춰 프로젝트를 진행 할 계획이다.

- 중간 점검 후

Blob 타입을 데이터베이스에서 활용하는 것을 중점으로 프로젝트를 진행하기에 빈약할 수 있어 이미지 워터마크 처리부분을 추가했었다. 하지만 Blob 타입은 데이터베이스에서 지원하는 간단한 연산이라는 중간 점검 feedback을 받고 어떻게 진행을 해야할지 의논을 했다. 이미지 텍스트 분류에 있어 발생할 수 있는 문제점(워터마크와 이미지 안의 텍스트 구별)을 미리 예상하고 있었지만 교수님의 feedback을 참고하여 프로젝트의 무게중심을 이미지 처리로 방향을 바꿨다. 프로젝트 진행을 이미지 처리에 많은 시간이 소요 될 것을 예상하여, 크롤링과 oracleDB 이미지 저장을 먼저 구축해 놓고 나중에 이미지 분류 작업과 합치는 방향으로 진행했다.

프로젝트 결과 이미지의 텍스트 포함여부 정확도는 95~97% 정도 된다(평균 100장 중 3~5개). 오류는 FP(False Positive)로서 text를 포함하지 않았는데 포함한 것으로 나타낸 것이다. 반대로 FN(False Negative)로서 text를 포함했는데 포함하지 않은 것으로 나타낸 오류는 찾아보기 힘들었다. 이는 사용자가 워터마크와 같은 텍스트를 포함하지 않는 이미지를 찾아서 활용하는데 유용한 결과라고 보여진다.

문제점으로는 이미지 분류에서 크롤링된 이미지의 해상도에 따라 결과가 달라진다는 점이다. 이러한 이유로 고화질 이미지를 얻기 위해서 이미지를 두번 수집하는 중복성이 발생하였다. 이 문제를 해결하기 위해 고화질 이미지도 정확히 텍스트 분류할 수 있는 모델을 만들 필요가 있다. 또한, 이미지 안에 텍스트가 불필요한 텍스트가 아닌 경우라면 필요한 이미지를 너무 많이 걸러낼 수가 있다. 예를 들어, 'supermarket'이라 검색하여 이미지를 얻고 싶은 경우에는 상호명을 모두 텍스트로 인식하여 정확도가 현저히 떨어질 수 있다는 단점을 갖는다. 이는 프로젝트의 가장 큰 한계점이라고 보여진다. 하지만 그런 주제를 제외한다면, 사용자가 직접 구글에 이미지를 검색하여 하나씩 다운로드하는 번거로운 일을 효과적으로 단순화 시킬 수 있어 실용적 측면에서 훌륭하다고 생각한다.

3.2 느낀점

박민규 : 1학기 때 빅데이터관리시스템을 들었을 때를 생각하며 2학기가 온라인으로 진행되어 참 아쉽다는 생각을 했습니다. 대면 수업을 통해 질의응답 하는 시간이 필요하다고 느꼈고, 육성으로 듣는 강의가 이해도와 집중력 측면에서 더 효과적이지 않나 생각합니다. 프로젝트를 진행하는 것에 있어서도 팀원들과 따로 약속을 잡아야 했습니다. 하지만 팀원 모두 적극적으로 문제를 해결하고자 하였고 서로의 의견을 존중하고 잘 반영하여 원래 목표하였던 바를 이룰 수 있었던 것 같습니다. 온라인 강의로 끝날 수 있는 지식을 직접 프로젝트를 통해 구현할 수 있어서 앞으로 데이터사이언스가 되는데 중요한 초석이 되지 않을까 생각합니다.

백종민 : 오프라인이 아닌 온라인 수업에서 어떻게 프로젝트를 진행 하는지, 실력이 부족해 팀에 민폐가 되지 않을까 많이 걱정되었습니다. 오히려 먼저 팀원분들의 연락을 받을 수 있었고 많은 도움을 받아 프로젝트를 완성할 수 있었다고 생각됩니다. sql을 자격증 공부를 위해 이론으로만 무작정 공부했었습니다. 하지만 수업과 프로젝트로 orcale sql을 다운받아서 실행해보고, 파이썬으로 sql 코드를 작성하면서 테이블을 만들고 자료를 넣어 DB를 구축할 수 있게 되었습니다. 특히 Blob 타입을 사용하면서 이미지와 다양한 데이터 타입도 가능하다는 것을 알게 되어 좋았습니다.

김원준 : 코로나19로 텀프로젝트를 진행하기가 여러모로 힘든 점이 많았던 것 같습니다. 하지만, 팀원들끼리의 합이 잘 맞았고, 큰 의견차 없이 텀 프로젝트를 마무리 할 수 있어서 좋은 경험이 된 것 같습니다. 무엇보다 데이터베이스를 구축하는 법을 간략하게라도 직접적인 코딩을 통해 경험했기 때문에 추후 DB구축에 많은 도움이 될 것 같습니다. 이번에 크롤링 부터 이미지의 텍스트 처리, SQL을 이용한 DB구축까지 진행한 프로그램은 향후에도 유익하게 사용할 수 있을 것 같습니다.

4. 참조**(Reference)**

4.1 Image storing

[1] https://cx-oracle.readthedocs.io/en/latest/user_guide/lob_data.html

[2] http://www.dba-oracle.com/t_storing_insert_photo_pictures_tables.htm

[3] https://docs.oracle.com/cd/B28359_01/appdev.111/b28845/ch7.htm#i1037547

[4] https://github.com/oracle/python-cx_Oracle/tree/master/samples

4.2 Classification

[5] https://d2.naver.com/helloworld/8344782

[6] https://brunch.co.kr/@kakao-it/304

[7] https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/