[PROJECT]

엣지팜 알고리즘 고도화를 위한 Bio data analysis & object detection

No. 32 Hello, Stranger

박민성, 임현진, 김영락, 정현명

목차

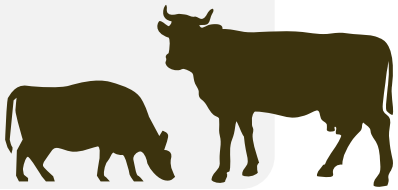
프로젝트 배경

- 가축 사육 방식의 변화로 인한 가축 간 사고(승가, 싸움 등) 발생률 증가
- 사고 발생으로 인한 가축 품질 저하
- 가축 관리 비용 증가

프로젝트 목적

가축 관리 시스템 자동화를 통한

- 동물복지 실현
- 인력 감축, 관리 비용 감소
- 가축 품질 및 생산성 향상



개발환경

운영체제

사용언어

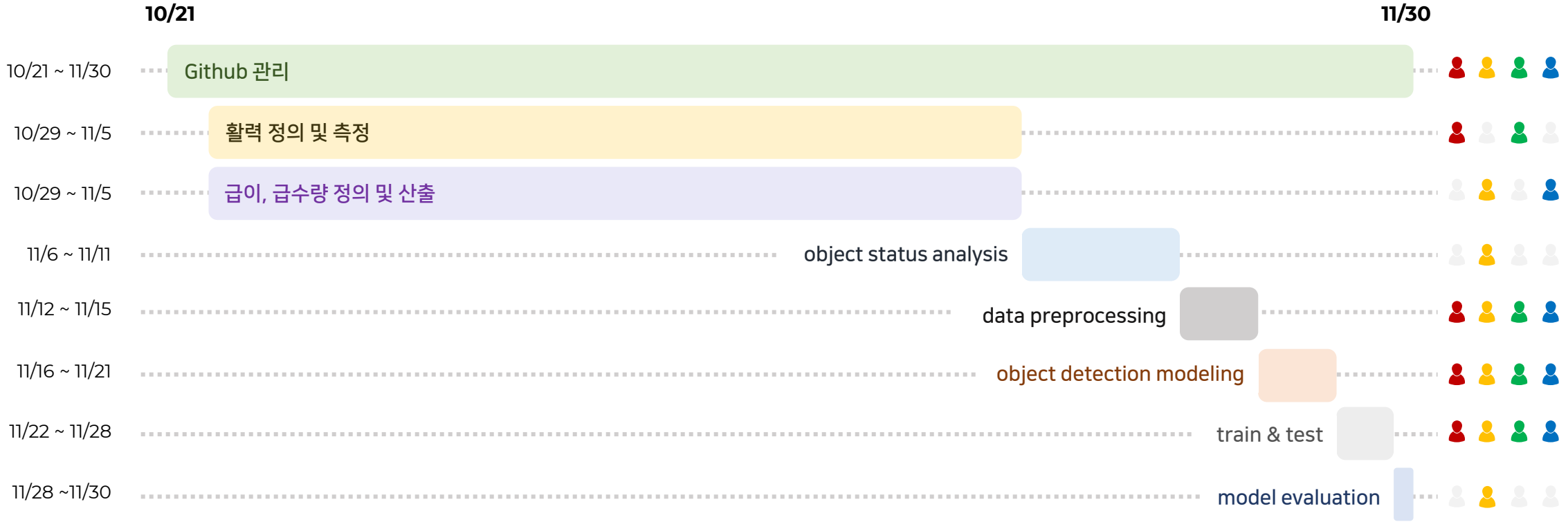
IDE

버전관리

프레임워크

환경설정

팀원 역할 및 프로젝트 프로세스

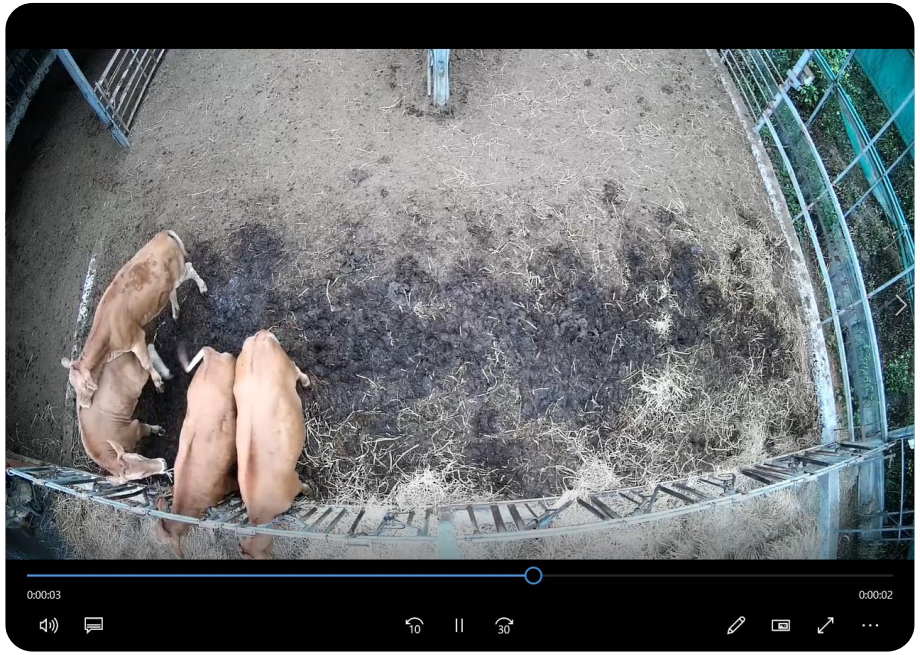


Hello, stranger

김영락 정현명 임현진 박민성

프로젝트 단계별 내용(코드, 이미지)

Input Data



Video

mounting_001_det.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
frame,xc,yc,width,height,theta,no_x,no_y,ne_x,ne_y,ta_x,ta_y
0,335.51562,716.8231,153.29169,397.55286,-0.02791861,295.9156,902.618,315.31647,853.4316,314.7534,541.5794
0,477.74097,711.7771,155.67563,411.68088,0.0007339842,452.79477,908.36743,455.64124,885.6317,462.81076,529.3294
0,302.3953,437.8245,189.59404,344.30585,0.71013266,352.7359,525.24603,174.37184,496.10495,163.13446,290.417
0,223.08228,705.19324,132.59593,411.3255,-0.021971922,192.122,894.6599,217.08551,840.83356,242.37367,527.88446
1,335.44568,716.82056,153.40027,397.5014,-0.027696174,295.91135,902.6294,315.3343,853.4981,314.6226,541.5655
1,477.71783,711.781,155.68893,411.6654,0.0008630368,452.7664,908.35156,455.5898,885.64307,462.7869,529.3523
1,302.38037,437.8048,189.60982,344.30472,0.71008784,352.70966,525.25885,174.3612,496.11237,163.1535,290.41714
1,222.9475,705.01917,132.46005,410.97983,-0.022645364,191.89873,894.2006,216.7485,840.3116,240.99646,527.925
2,335.46262,716.81885,153.2973,397.55597,-0.027601443,295.9366,902.6083,315.3095,853.31134,314.62955,541.5394
2,477.7013,711.7721,155.67953,411.6785,0.0008408121,452.95486,908.35077,455.6235,885.6617,462.6032,529.3262
2,302.3932,437.78528,189.6165,344.27417,0.7100465,352.7224,525.2271,174.35703,496.09296,163.14577,290.4136
2,222.97003,705.1464,132.53925,411.14075,-0.022007558,192.06483,894.4418,216.82758,840.5987,241.74709,527.9209
3,335.401,716.8661,153.31717,397.58868,-0.02906389,296.07864,902.7992,315.4135,852.779,313.82993,541.51544
3,477.55304,711.7533,155.50912,411.67023,0.0010878746,452.9069,908.42883,455.69247,885.2154,462.55734,529.3189
3,302.4397,437.75684,189.55066,344.28696,0.7103008,352.61908,525.0641,174.50435,496.08255,163.16212,290.4373
3,222.9165,706.0897,131.42145,409.96576,-0.03048264,192.65627,896.3135,219.64229,840.9943,242.8255,528.31445
4,335.27838,716.7676,153.03024,397.65002,-0.02848671,296.68387,902.8348,315.27478,853.0141,312.6892,541.2657
4,477.42902,711.7077,155.31241,411.75546,0.0011661121,453.56134,908.41345,455.78632,885.2992,461.92828,529.2771
4,302.5284,437.8457,189.52965,344.36316,0.7105659,352.7337,525.13306,174.45912,496.18665,163.08432,290.47006
4,222.67328,706.012,131.0907,410.05258,-0.029028853,193.22885,896.55536,219.85374,841.309,242.10124,528.1
5,335.53296,716.81506,153.3938,397.6067,-0.028195163,295.81476,902.6363,315.34283,853.2412,314.85077,541.58466
5,477.77493,711.7853,155.7066,411.659,0.000703148,452.57175,908.3725,455.59277,885.5331,463.0378,529.3363
5,302.37274,437.7945,189.58548,344.3494,0.71022326,352.75494,525.2198,174.3881,496.10254,163.11006,290.44363
5,223.00574,705.2422,132.57076,411.27158,-0.022230767,192.05568,894.71246,216.80376,840.88837,241.75221,527.98615
6,335.52814,716.8114,153.3919,397.6104,-0.028147997,295.82993,902.62744,315.33646,853.251,314.82465,541.5785
6,477.76913,711.78613,155.69977,411.66644,0.0007056169,452.60516,908.37714,455.5907,885.54803,462.99728,529.3323
6,302.3761,437.7994,189.58365,344.34247,0.7102774,352.75638,525.217,174.38647,496.10336,163.10774,290.44266
6,223.001,705.2241,132.5601,411.2636,-0.022228569,192.06534,894.68506,216.80621,840.85864,241.73923,527.9673
7,335.52814,716.8114,153.3919,397.6104,-0.028147997,295.82993,902.62744,315.33646,853.251,314.82465,541.5785
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

Text

프로젝트 단계별 내용(코드, 이미지)

1. Object Status Analysis

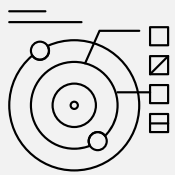
a. 활력 정의 및 산출 방식

b. 급수, 급이량 정의 및 산출 방식

c. 객체 rotate 박스 생성

d. drink, eat section 생성

e. 결과 video, image



2. Object Detection

a. Model inference

b. Data preprocessing

c. Model train, predict, evaluate

1 - a) 활력 정의 및 산출 방식

1) 객체의 이동거리 산출

각 프레임마다 객체의 중심 좌표간 거리를 math.dist 라이브러리를 이용해 합산하여, 누적 이동거리(pixel) 산출

2) 객체의 기초대사량(bmr) 측정

항온 동물인 소는 표면적과 기초 대사량이 비례하기 때문에 소를 원기둥으로 가정하여 소의 기초대사량을 측정

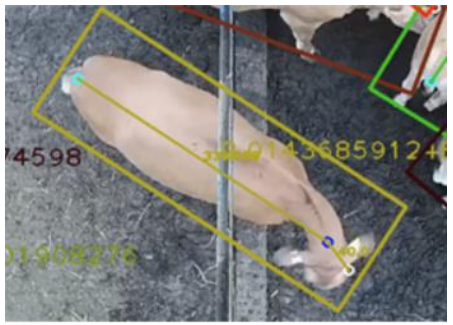
동물		생쥐	흰쥐	개	말
체중(kg)		0.02	0.20	15	441
기초대사량	㎉/일	3.82	20	773	4,983
	㎉/kg	212.0	100.0	51.5	11.3
	㎉/㎡ (체표면적)	1,185	1,160	1,039	1,048

- 체중이 무거울수록 기초대사량은 많으나, 단위 체중당 기초대사량은 작은 동물일수록 크고, 단위 체표면적당 기초대사량은 동물의 종류에 관계없이 거의 같다.
- 영양가 : 탄수화물=4㎉/g, 단백질=4㎉/g, 지방=9㎉/g
- 열의 일당량 : W=JQ, J=4.186J/㎈

3) 객체 체표면적 산출

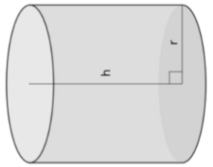
- 소의 몸통 두께는 가장 얇게 잡혔을 경우가 적합한 수치이므로 이상치 값을 제외한 Q1값 사용

- 소의 길이는 소의 목부터 엉덩이까지의 거리로 소가 움직이면서 직선거리가 바뀜으로 가장 길 때가 소의 전체 길이이므로 이상치 값을 제외한 Q3값을 사용



옳은 예시

잘못된 예시



r = 소의 몸통 두께(width) / 2

h = 소의 길이(height)

1 - a) 활력 정의 및 산출 방식

🔎 이동량을 kcal로 변환하기

1) 소의 1일 칼로리 소모(100%) : 기초대사(60%) + 소화(15%) + 활동(25%)

2) 소의 겉넓이를 이용한 기초대사량(bmr) 평균치 계산

3) 기초대사량 평균치(bmr)를 통한 소가 활동에 사용하는 평균 칼로리 계산
: 1일 활동 칼로리 소모량(kcal) = (1일 기초대사량(kcal) / 60) * 25

4) 환산계수(kcal / px) = 1일 활동 칼로리 소모량(kcal) / 전체 소의 1일 이동거리 누적치의 평균(px)

5) 각 소가 이동에 소모하는 칼로리(kcal) =각 소의 픽셀 이동 누적치(px) * 환산계수(kcal / px)

```
object_dict[int(i[11])][3] += object_dict[int(i[11])][2] * move_length_to_kcal # 실시간 이동량 * 환산계수

cow_surface = math.pi * float(width_Q1[int(i[11])]) * ((float(width_Q1[int(i[11])]) /2) + (float(height_Q3[int(i[11])])))
cow_bmr = float(cow_surface) * kg_per_squaremeter / (24*60*60*30)
object_dict[int(i[11])][4] += cow_bmr



img = cv2.putText(img, str(int(i[11])), (int(i[0]),int(i[1])), cv2.FONT_HERSHEY_PLAIN, 2, (255,255,255), thickness=2)
```

1 - b) 급수, 급이량 정의 및 산출방식

1. 급수량, 급이량 정의:

1) 1일 음수 정보:

- 음수 횟수 : 10~15회
- 음수 시간: 약 30분
- 음수량 : 회당 약10L, 30~50갤런(110~190L)

2) 1일 식사 정보:

- 식사 횟수 : 9~14회
- 식사 시간: 3~5시간
- 식사량 : 몸무게의 2~2.5%, 습식기준 약 50kg,
  건식기준 약 25kg -> 2.3g~3.8g /sec, 138.9g~231.5g/min

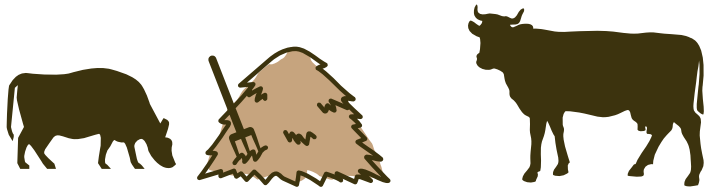
2. 산출 방식:

1) 급수량

- 1 frame = 1/30 초
- 분당 음수량 = 1일 음수량(ml)/ 1일 음수 시간(분) = 3700~6300ml / min
- frame 당 음수량 = 분당 음수량(ml) / 60*30 = 2~3.5ml / frame
- 변수 정의: ml_per_frame = 3

2) 급이량

- 1 frame = 1/30 초
- 분당 식사량 = 1일 식사량(g) / 1일 식사시간(시간)*60
  = 138.9~231.5g / min
- frame 당 식사량 = 분당 식사량(g) / 60*30 = 0.08~0.13g / frame
- 변수 정의: gram_per_frame = 0.1



1 - b) 급수, 급이량 정의 및 산출방식

```python
# object(box)와 section이 겹치는 비율(iou) 산출
def iou_box (section, box):

    section_polygon = Polygon(section)
    box_polygon = Polygon([tuple(box[0]), tuple(box[1]), tuple(box[2]), tuple(box[3])])

    intersection = section_polygon.intersection(box_polygon).area
    union = section_polygon.union(box_polygon).area
    bound = section_polygon.intersection(box_polygon).boundary

    return intersection/union, bound


# object(circle)와 section이 겹치는 비율(iou) 산출
def iou_circle (section, circle):

    section_polygon = Polygon(section)
    circle_polygon = Point(circle[0]).buffer(circle[1])

    intersection = section_polygon.intersection(circle_polygon).area
    union = section_polygon.union(circle_polygon).area
    bound = section_polygon.intersection(circle_polygon).boundary

    return intersection/union, bound
```

1) iou_box 함수:

- 객체(box 형태)와 section이 겹치는 부분(intersection)과 객체와 section의 전체 범위(union)의 비율 산출

2) iou_circle 함수:

- 객체(circle 형태)와 section이 겹치는 부분(intersection)과 객체와 section의 전체 범위(union)의 비율 산출

```python
# 객체의 코, 목 좌표를 지름으로 하는 원 생성
img = cv2.circle(img, (int((i[5]+i[7])/2), int((i[6]+i[8])/2)),
                 int(round(math.dist([int(i[5]),int(i[6])],[int(i[7]),int(i[8])]),3)/2), (255,255,255), 2)
circle = [(int((i[5]+i[7])/2), int((i[6]+i[8])/2)),
          int(round(math.dist([int(i[5]),int(i[6])],[int(i[7]),int(i[8])]),3)/2)]

# 객체와 drink_section iou 계산
# drink_iou, drink_bound = iou_box(drink_section, box)
drink_iou, drink_bound = iou_circle(drink_section, circle)

# 객체와 eat_section iou 계산
# eat_iou, eat_bound = iou_box(eat_section, box)
eat_iou, eat_bound = iou_circle(eat_section, circle)

# config에서 지정한 iou보다 현재 프레임의 특정 개체와 drink_section간 iou가 큰 프레임을 카운트
if drink_iou > config['drink_iou'] :
    # 식사량 frame count
    object_dict[int(i[11])][5] += 1
    # intersection 색칠
    img = cv2.fillConvexPoly(img, np.int32([np.array(drink_bound)]), (100, 100, 255))

# config에서 지정한 iou보다 현재 프레임의 특정 개체와 eat_section간 iou가 큰 프레임을 카운트
if eat_iou > config['eat_iou'] :
    # 음수량 frame count
    object_dict[int(i[11])][6] += 1
    # intersection 색칠
    img = cv2.fillConvexPoly(img, np.int32([np.array(eat_bound)]), (255, 100, 0))
```

1) 객체와 drink, eat section의 iou 산출:

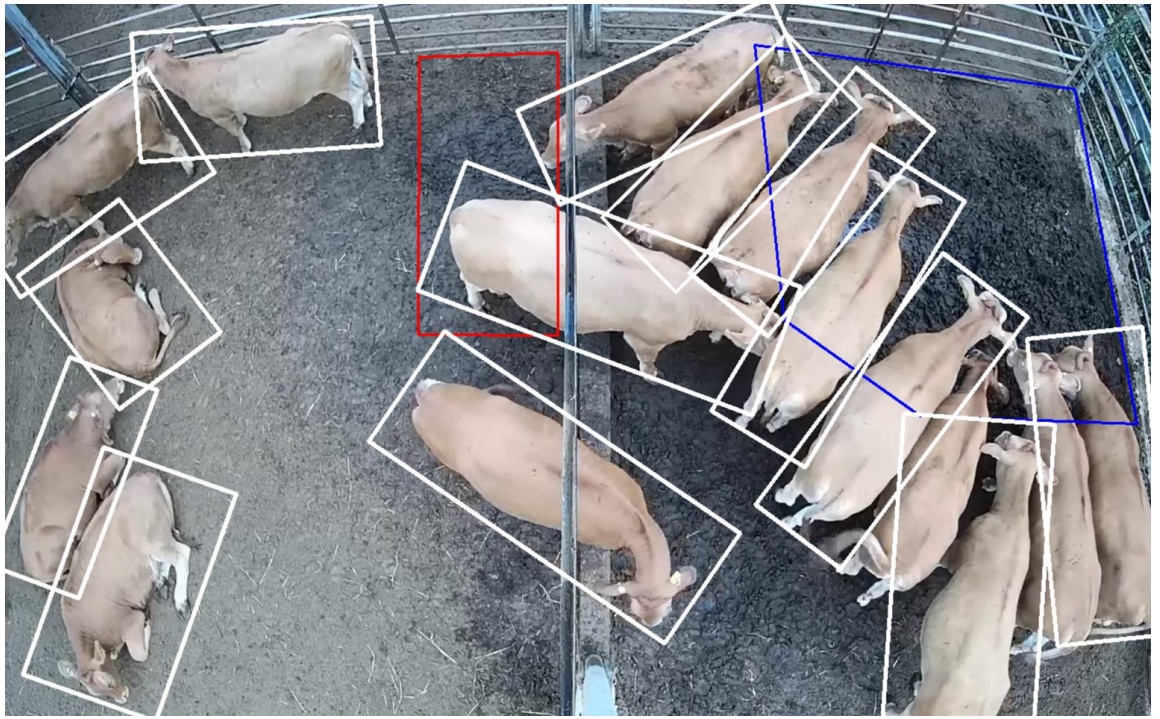
- iou_box 함수를 사용 할 경우 객체의 머리 부분이 아닌 다른 부분이 section과 겹쳤을 때도 iou 값이 높아짐 -> 식사 또는 음수 중이 아닌 상황에도 iou가 기준치를 넘으면 식사 또는 음수 중으로 판단 -> iou_circle 함수를 사용하여 iou 산출
- 이때 circle은 객체의 코와 목 좌표를 지름으로 하는 원(=머리)

2) 급수, 급이량 누적:

- config.yaml에 정의된 drink, eat iou 값보다 높을 경우 해당 frame을 counting
- ml_per_frame과 gram_per_frame을 각각 counting 된 값과 곱해 누적 급수, 급이량 산출

1 - d) drink, eat section 생성

```python
# 마우스로 다각형을 만드는 클래스
class draw_polygon :

    # 클래스 생성 시
    def __init__(self,config):

        # 리스트 형으로 drink_section과 eat_section 초기화'
        self.drink_section = []
        self.eat_section = []

        # config의 영상 첫 프레임을 가져와 drink_img, eat_img에 저장
        cap = cv2.VideoCapture(config['mp4_path'])
        self.drink_img =  cap.read()[1]
        self.eat_img = cap.read()[1]

        # drink와 eat의 각 이미지에서 다각형 점을 찍은 후 esc로 탈출, 찍은 점은 각 section 리스트에 저장
        self.select_section()

    # drink_img의 마우스 이벤트
    def drink_mouse_event(self, event, x, y, flags, param):

        # 왼쪽 클릭 시 클릭 좌표 저장 및 drink_img에 해당좌표 표시
        if event == cv2.EVENT_FLAG_LBUTTON :
            self.drink_section.append([x,y])
            cv2.circle(self.drink_img,(x,y),3,(255,0,0),2)
            cv2.imshow('drink',self.drink_img)

    # eat_img의 마우스 이벤트
    def eat_mouse_event(self, event, x, y, flags, param):

        # 왼쪽 클릭 시 클릭 좌표 저장 및 eat_img에 해당좌표 표시
        if event == cv2.EVENT_FLAG_LBUTTON :
            self.eat_section.append([x,y])
            cv2.circle(self.eat_img,(x,y),3,(0,0,255),2)
            cv2.imshow('eat',self.eat_img)
```



1) red box: drink section

2) blue box: eat section

3) white box: objects

drink_mouse_event, eat_mouse_event 함수:

- 마우스 클릭으로 drink, eat section 범위 설정

1 - d) drink, eat section 생성

1) red circle: 음수 중인 객체

2) blue circle: 식사 중인 객체

3) white text: 객체 번호, 누적 음수량, 누적 식사량, 누적 활력량

2 - a) Model inference

1) pytorch기반의 YOLOX 모델을 활용한 obeject detection

- base line : https://github.com/zhangming8/yolox-pytorch

2) 입력데이터 형식(COCO dataset format)

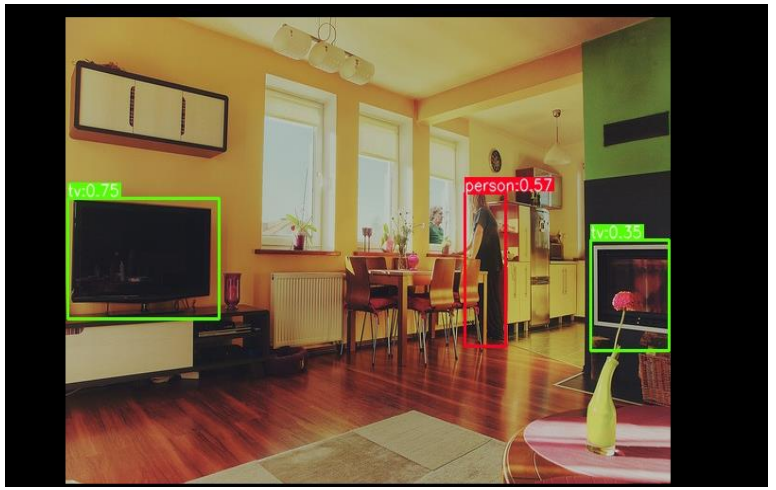
- image 파일과 이를 설명하는 annotation.json 파일로 이루어짐


```
info{
   "year"              : int,
   "version"           : str,
   "description"       : str,
   "contributor"       : str,
   "url"               : str,
   "date_created"      : datetime,
}

image{
   "id"                : int,
   "width"             : int,
   "height"            : int,
   "file_name"         : str,
   "license"           : int,
   "flickr_url"        : str,
   "coco_url"          : str,
   "date_captured"     : datetime,
}

license{
   "id"                : int,
   "name"              : str,
   "url"               : str,
}
```

```
annotation{
   "id"                : int,
   "image_id"          : int,
   "category_id"       : int,
   "segmentation"      : RLE or [polygon],
   "area"              : float,
   "bbox"              : [x,y,width,height],
   "iscrowd"           : 0 or 1,
}

categories[{
   "id"                : int,
   "name"              : str,
   "supercategory"     : str,
}]
```

[annotation.json 구조]

3) 학습 결과 예시:




```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.134
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.251
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.131
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.069
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.158
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.164
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.159
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.276
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.300
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.133
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.341
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.365
```

2 - b) Data preprocessing

```python
# 소 영상과 trk.txt를 입력받아 50 frame마다 img를 저장하고 해당 이미지에 대한 정보를 누적하여
# annotation.json 파일을 생성하는 함수
def create_img_ann (config,switch):


    # 가장 바깥의 딕셔너리
    coco_group = OrderedDict()

    # 내부 5가지 정보 딕셔너리
    info = OrderedDict()
    licenses = OrderedDict()
    categories = OrderedDict()
    images = OrderedDict()
    annotations = OrderedDict()


    # info, licenses 는 학습에 사용되지 않으므로 임의로 설정
    info['year'] = '2021'
    info["version"] = "0.1.0"
    info["description"] = "Dataset in COCO Format"
    info["contributor"] = ""
    info["url"] = ""
    info["date_created"] = "2021-11"

    licenses["id"] = 1
    licenses["url"] = ""
    licenses["name"] = "ALL RIGHTS RESERVED"

    # categories 는 모든 이미지에서 소만 탐지하면 되기 때문에 다음과 같이 설정
    categories["id"] = 1
    categories["name"] = "cow"
    categories["supercategory"] = "none"

    # info, licenses, categories를 가장 바깥 딕셔너리에 저장
    coco_group["info"] = info
    coco_group["licenses"] = [licenses]
    coco_group["categories"] = [categories]
```

1) annotation.json 파일 구조 생성

- coco_group dictionary 생성
- coco_group dictionary에 annotation.json 기본 구성 요소 추가
- 기본 구성요소: info, licenses, categories, images, annotations

2 - b) Data preprocessing

```
# train 혹은 test 경로에 있는 모든 폴더에 접근하여 소 영상과 trk.txt를 통해 데이터를 생성
for t in data_list:

    # video_path ex : train_data/1/1.mp4 , text_path ex : train_data/1/1_trk.txt
    video_path, text_path  = map(lambda x : data_path+'/'+t+'/'+x ,os.listdir(data_path+'/'+t))


    cap = cv2.VideoCapture(video_path)

    df = pd.read_csv(text_path, sep=",", header=None)
    frame = 0

    if cap.isOpened():
        while True:
            # 프레임을 잘 읽어오면 ret는 True, img 는 프레임 이미지로 저장된다
            ret, img = cap.read()

            if ret:
                # frame에 맞는 객체들의 정보를 모두 가져옴
                df_frame = df.loc[df[0]==frame,1:].values.tolist()

                # 50 frame 단위로 다음과 같은 작업 실행
                if frame % 50 == 0 :

                    # img정보를 images에 저장
                    h,w,c = img.shape
                    image = {}
                    image['height'] = h
                    image['width'] = w
                    image['license'] = 1
                    image['id'] = img_cnt
                    image['file_name'] = f'{img_cnt}.jpg'

                    img_list.append(image)

                    # 현재 frame에 존재하는 각 소들의 정보를 annotation에 저장
                    for d in df_frame:

                        # trk.txt에 있는 w,h는 rotate_box 크기이고 필
                        points = rotate_box_dot(d[0], d[1], d[2], d[3], d[4])
                        bbox_w,bbox_h =circumscription(points)

                        ann = {}
                        ann['id'] = ann_id
                        ann['iscrowd'] = 0
                        ann['bbox'] = d[:2]+[bbox_w,bbox_h]  # bbox는 객체를 포함하는 직사각형 [xc, yc, w, h]
                        ann['image_id'] = img_cnt
                        ann['category_id'] =1
                        ann['area'] = float(d[2]*d[3]) # area는 segmentation의 넓이

                                                # rotate box를 segmentation으로 활용하여 정확도를 높임
                        ann['segmentation'] =[[max(0.0,i) for i in map(float,points.flatten())]]

                        ann_id += 1

                        ann_list.append(ann)
```

```
# 누적한 img 와 annotation 정보를 저장
coco_group['images'] = img_list
coco_group['annotations'] = ann_list

# train_annotation.json 혹은 test_annotation.json으로 저장
if switch =='train':
    with open ('train_annotation.json','w',encoding='utf-8') as make_file:
        json.dump(coco_group, make_file,indent='\t')

else:
    with open ('test_annotation.json','w',encoding='utf-8') as make_file:
        json.dump(coco_group, make_file,indent='\t')
```

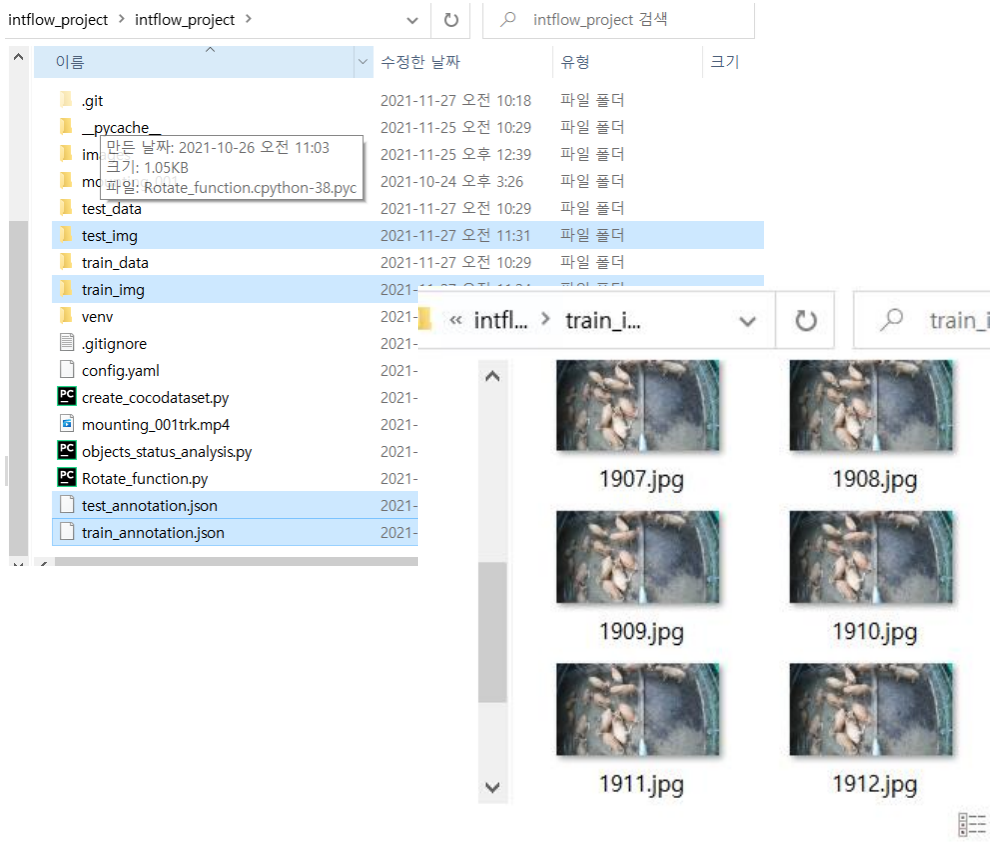
2) 입력 데이터 annotation 정보 저장

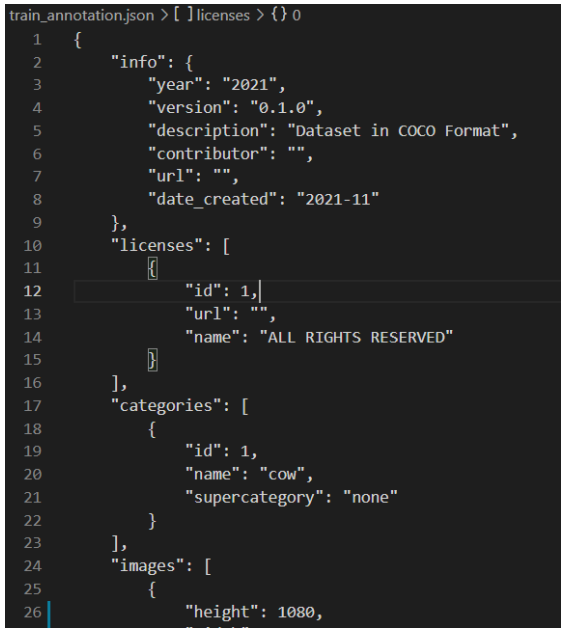
- 입력 영상에서 50 frame 단위로 이미지와 이미지 정보를 추출해 img_list에 저장
- 한 이미지에 있는 여러 객체들의 정보를 각각 ann_list에 저장
- coco_group에 img_list, ann_list 입력
- json파일로 변경 및 저장

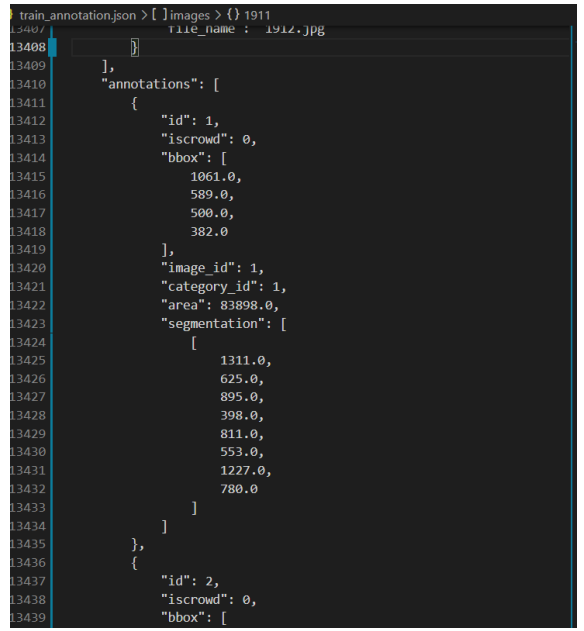
2 - b) Data preprocessing

3) 생성되는 폴더 및 파일

- train_img, test_img 폴더



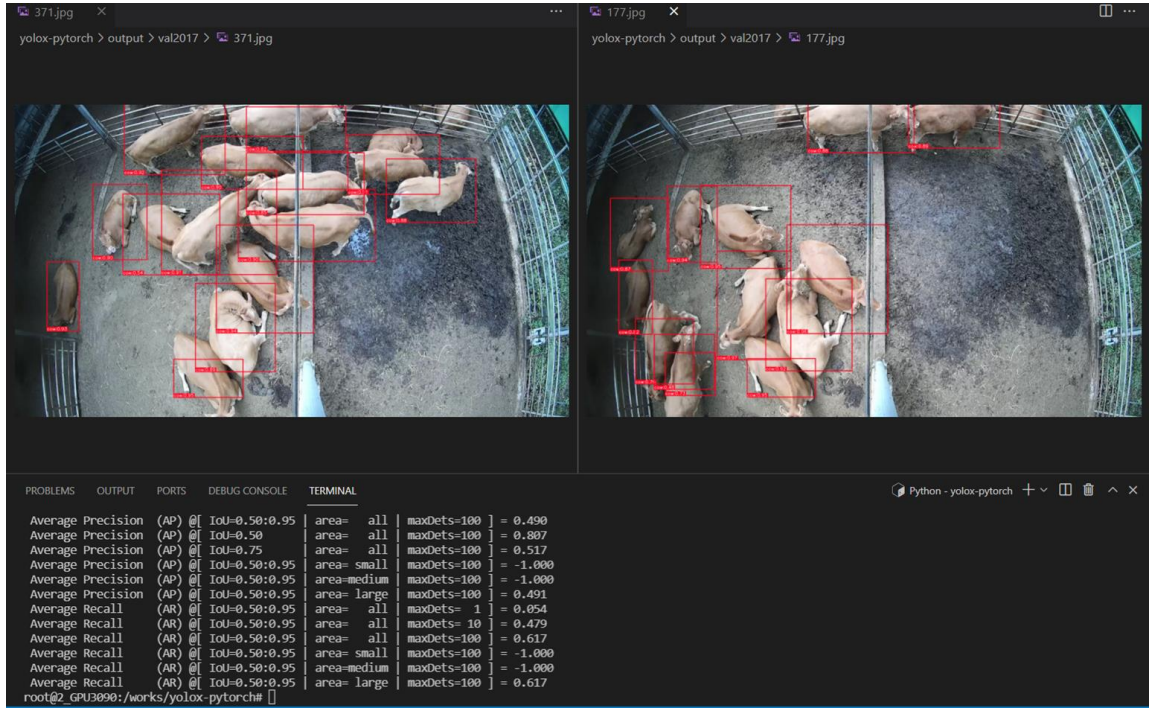




- train_annotation.json, test_annotation.json 파일
  - bbox : rotate box를 외접하는 직사각형의 정보([x_cen, y_cen, width, height])
  - segmentation : rotate box 꼭지점 좌표
  - area : rotate box의 면적

2 - c) Model train, predict, evaluate

```
opt.exp_id = "test4"  # experiment name, you can change it to any other name
opt.dataset_path = "/works/yolox-pytorch/data/dataset"  # COCO detection
# opt.dataset_path = r"D:\work\public_dataset\coco2017"  # Windows system
opt.backbone = "CSPDarknet-s"  # CSPDarknet-nano, CSPDarknet-tiny, CSPDarknet-s, CSPDarknet-m, l, x
opt.input_size = (800, 800)
opt.random_size = (14, 26)  # None; multi-size train: from 448(14*32) to 832(26*32), set None to disable it
opt.test_size = (800, 800)  # evaluate size
opt.gpus = "0,1"  # "-1" "0" "3,4,5" "0,1,2,3,4,5,6,7" # -1 for cpu
opt.batch_size = 32
opt.master_batch_size = -1  # batch size in first gpu. -1 means: master_batch_size=batch_size//len(gpus)
opt.num_epochs = 100

# coco 80 classes
opt.label_name = [
    'cow']
```



4) config 환경설정 후 실행 결과

- 완벽하지 않지만 object detect를 하는 모습

- 객체를 일부만 탐지 하거나 탐지한 영역의 여백이 많은 부분이 있음

- 약 60% 정확도를 보임

프로젝트 결과(분석)

1. Bio data analysis

자체평가

1) 잘한 점

- 소의 활동 칼로리에 대한 정보의 부재를 체표면적을 이용한 칼로리 계산 방법으로 보완
- 주석을 통한 코드 상세 설명, github 버전 관리
- 객체 지향적 코드 작성(config.yaml, class 및 함수 분류, python file 모듈화 등)

2) 아쉬운 점

- 코드의 가독성 떨어짐
- Labeling Data(det.txt, trk.txt)의 오차로 인한 측정 오류
- 단일적인 카메라의 촬영 구도와 왜곡

보완점

✓ **코드의 가독성 떨어짐 >** 자주 쓰이는 값의 변수화

✓ **Labeling Data(det.txt, trk.txt) 오차로 인한 측정 오류 >** 임계치 설정을 통한 오차 범위 최소화

✓ **단일적인 카메라의 촬영 구도와 왜곡 >** 다양한 각도의 카메라 설치를 통한 데이터 정확도 향상

프로젝트 결과(분석)

2. Object detection

1) 진행 사항: object detection 가능하지만 이상행동 판별은 미완

2) 이상행동 판별 모델링 실패 원인 분석

- 시간 및 숙련도 부족
  - yolo model 과 coco dataset의 이해 및 사용 미숙
  - 리눅스, gpu, 서버 등 새로운 작업 환경
- 데이터 부족
  - 이상행동(승가, 싸움 등)에 대한 Labeling data 부족

3) 피드백 & 보완점

- 이상행동 데이터 추가 및 모델링
- 새로운 환경 학습, 숙련도 향상

관련 논문 및 레퍼런스

- Window에서 Python 연동하기, (n.d.) , from Notion: Retrieved by https://peach-yam-219.notion.site/Window-Python-15aa7924dd4a424a9b4c850e56331e5a.
- Open CV 1. 도형 그리기 원 타원 (Python).(2019. 3. 28), from Tistory: Retrieved by https://copycoding.tistory.com/147.
- Open CV – 이미지 비디오 읽기(2018. 01. 23), from Git hub: Retrieved by https://zzsza.github.io/data/2018/01/23/opencv-1/.
- Pandas Dataframe에서 SQL 사용하기 (2020. 10. 12), from Git hub: Retrieved by https://rakjido.github.io/2020/10/12/Pandas-Dataframe-Minipulation-using-SQL.
- [파이썬 OpenCV] 그리기 함수 - line, rectangle, circle, polyines, putText (2020. 9. 25), from Tistory: Retrieved by https://deep-learning-study.tistory.com/105.
- https://github.com/argusswift/YOLOv4-pytorch
- [Python]Ubuntu에서 Python3.7 환경변수 설정하기!(2019. 9. 7), from Tistory: Retrieved by https://somjang.tistory.com/entry/PythonUbuntu%EC%97%90%EC%84%9C-Python37-%ED%99%98%EA%B2%BD%EB%B3%80%EC%88%98-%EC%84%A4%EC%A0%95%ED%95%98%EA%B8%B0bashrc%ED%8C%8C%EC%9D%BC%EC%88%98%EC%A0%95.
- Eric Hofesmann, How to work with object detection datasets in COCO format)(2019. 02), from toward data science: Retrieved by https://towardsdatascience.com/how-to-work-with-object-detection-datasets-in-coco-format-9bf4fb5848a4.

감사합니다.