CVMI LAB - 201921725 안성현

Duck-farm-604 Project

<2022/02/05>

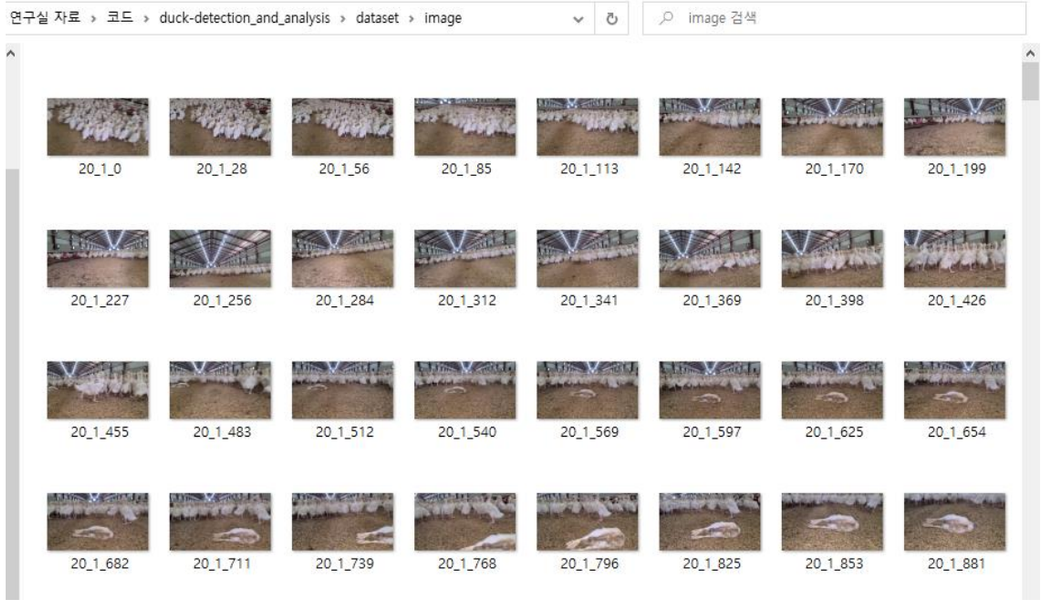
목 차

작업 소개

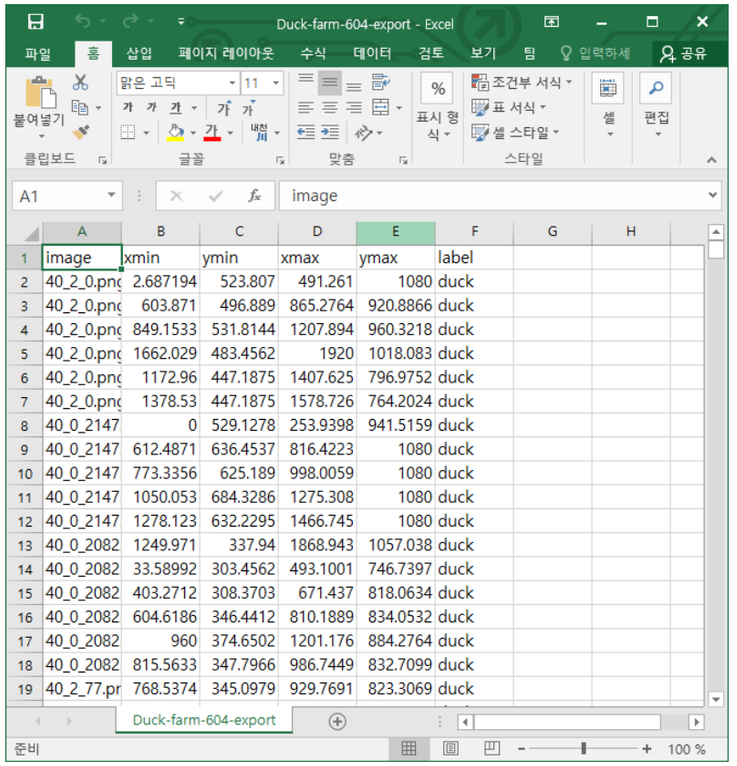
- Detectron을 이용한 오리 객체 검출
  - Detectron을 이용해서 (정상 오리, 죽은 오리, 누워있는 오리)를 검출한다
  - 농장에서의 실사용을 위해 '실시간 객체 검출'을 목표로 한다

작업 소개

- Duck-farm-604 Dataset
  - image: '오리령수_동영상 번호_동영상 내 프레임 번호'로 이루어진 png 이미지 604개
  - annotation: 각 이미지들을 라벨링해서 얻은 (image|xmin|ymin|xmax|ymax|label)로 이루어진 CSV 파일



dataset/image/*.*png*



	A	B	C	D	E	F
1	image	xmin	ymin	xmax	ymax	label
2	40_2_0.png	2.687194	523.807	491.261	1080	duck
3	40_2_0.png	603.871	496.889	865.2764	920.8866	duck
4	40_2_0.png	849.1533	531.8144	1207.894	960.3218	duck
5	40_2_0.png	1662.029	483.4562	1920	1018.083	duck
6	40_2_0.png	1172.96	447.1875	1407.625	796.9752	duck
7	40_2_0.png	1378.53	447.1875	1578.726	764.2024	duck
8	40_0_2147	0	529.1278	253.9398	941.5159	duck
9	40_0_2147	612.4871	636.4537	816.4223	1080	duck
10	40_0_2147	773.3356	625.189	998.0059	1080	duck
11	40_0_2147	1050.053	684.3286	1275.308	1080	duck
12	40_0_2147	1278.123	632.2295	1466.745	1080	duck
13	40_0_2082	1249.971	337.94	1868.943	1057.038	duck
14	40_0_2082	33.58992	303.4562	493.1001	746.7397	duck
15	40_0_2082	403.2712	308.3703	671.437	818.0634	duck
16	40_0_2082	604.6186	346.4412	810.1889	834.0532	duck
17	40_0_2082	960	374.6502	1201.176	884.2764	duck
18	40_0_2082	815.5633	347.7966	986.7449	832.7099	duck
19	40_2_77.pr	768.5374	345.0979	929.7691	823.3069	duck

dataset/Duck-farm-604-export.csv

작업 소개

- Detectron
  - pytorch 기반 object detection과 sementic segemanation을 위한 training/inference 플랫폼
  - FAIR(Facebook Artificial Intelligence Research)에서 개발

Model zoo는 FAIR에서 학습한 모델들이 관리되는 폴더

(->)

Pre-trained된 모델을 이용해서 전이학습이 간편함

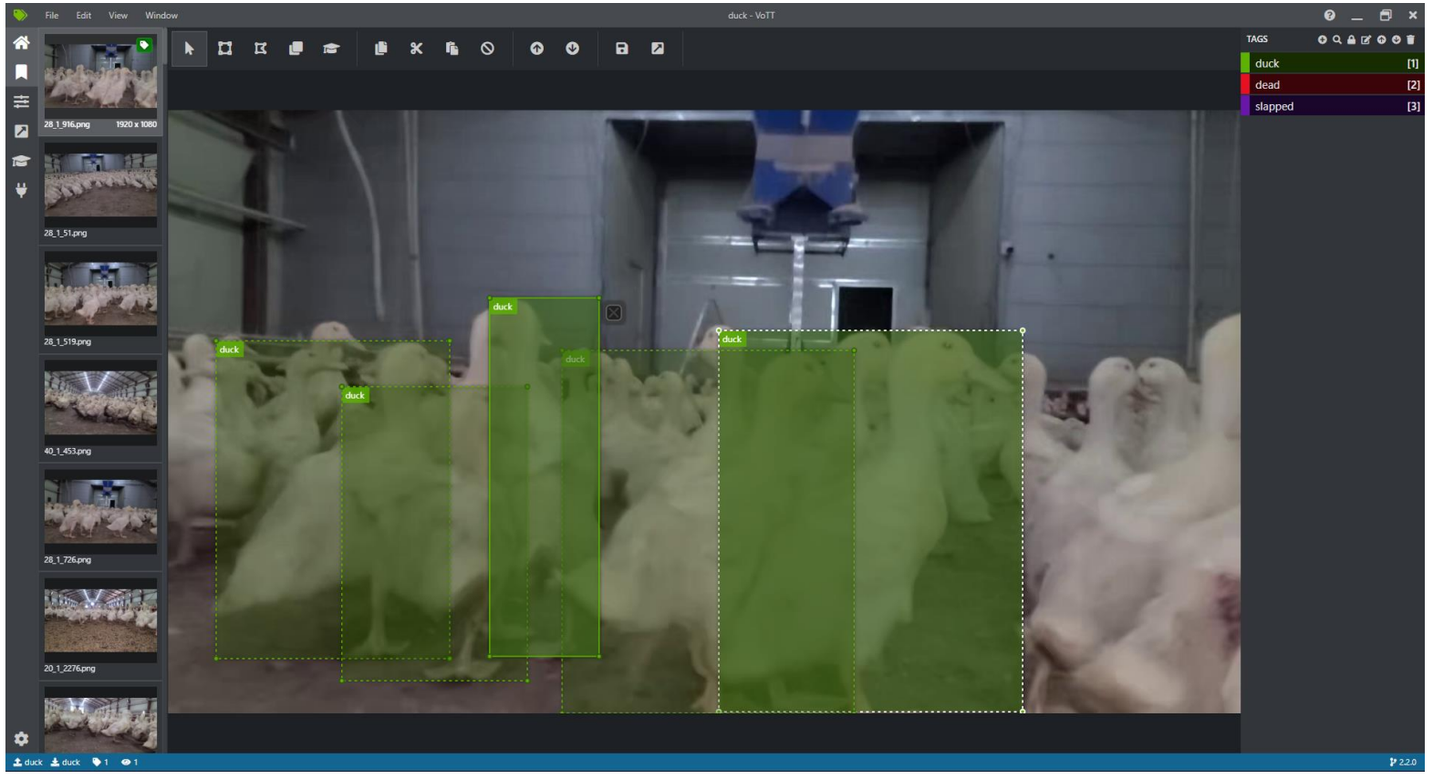
Faster R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

진행 과정

- 라벨링
  - VOTT를 이용해서 604개의 오리 이미지에 대해 라벨링을 진행함
  - 오리의 몸 80% 이상이 보이도록 바운딩 박스를 지정함



1. 머리 모두 드러나게
2. 꼬리 보이게
3. 머리,꼬리,발 끝에 맞게
4. 꼬리 아래, 발 부분은 살짝 가려져도 라벨링 진행

진행 과정

- COCO Dataset 제작
  - dataset/image/*.*png*, *dataset/Duck-farm-604-export.csv* -> COCO Json

행마다 [Image File 이름, Bounding Box의 xmin,ymin,xmax,ymax,label]이 저장됨

이름	날짜	유형	크기
20_1_0	2022-01-13 오후 9:42	PNG 파일	1,752KB
20_1_28	2022-01-13 오후 9:45	PNG 파일	2,943KB
20_1_56	2022-01-13 오후 9:45	PNG 파일	2,787KB
20_1_85	2022-01-13 오후 9:43	PNG 파일	1,814KB
20_1_113	2022-01-13 오후 9:45	PNG 파일	2,876KB
20_1_142	2022-01-13 오후 9:45	PNG 파일	3,015KB
20_1_170	2022-01-13 오후 9:43	PNG 파일	2,014KB
20_1_199	2022-01-13 오후 9:45	PNG 파일	2,848KB
20_1_227	2022-01-13 오후 9:43	PNG 파일	2,038KB
20_1_256	2022-01-13 오후 9:45	PNG 파일	2,451KB
20_1_284	2022-01-13 오후 9:44	PNG 파일	2,147KB
20_1_312	2022-01-13 오후 9:45	PNG 파일	2,314KB
20_1_341	2022-01-13 오후 9:45	PNG 파일	2,534KB
20_1_369	2022-01-13 오후 9:45	PNG 파일	2,434KB
20_1_398	2022-01-13 오후 9:43	PNG 파일	1,994KB

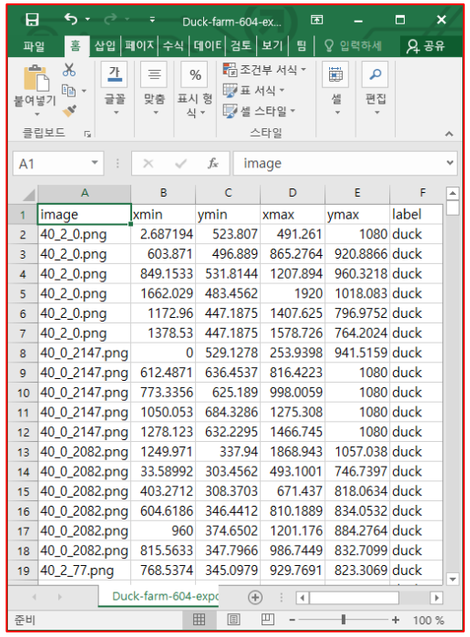


	image	xmin	ymin	xmax	ymax	label
2	40_2_0.png	2.687194	523.807	491.261	1080	duck
3	40_2_0.png	603.871	496.889	865.2764	920.8866	duck
4	40_2_0.png	849.1533	531.8144	1207.894	960.3218	duck
5	40_2_0.png	1662.029	483.4562	1920	1018.083	duck
6	40_2_0.png	1172.96	447.1875	1407.625	796.9752	duck
7	40_2_0.png	1378.53	447.1875	1578.726	764.2024	duck
8	40_0_2147.png	0	529.1278	253.9398	941.5159	duck
9	40_0_2147.png	612.4871	636.4537	816.4223	1080	duck
10	40_0_2147.png	773.3356	625.189	998.0059	1080	duck
11	40_0_2147.png	1050.053	684.3286	1275.308	1080	duck
12	40_0_2147.png	1278.123	632.2295	1466.745	1080	duck
13	40_0_2082.png	1249.971	337.94	1868.943	1057.038	duck
14	40_0_2082.png	33.58992	303.4562	493.1001	746.7397	duck
15	40_0_2082.png	403.2712	308.3703	671.437	818.0634	duck
16	40_0_2082.png	604.6186	346.4412	810.1889	834.0532	duck
17	40_0_2082.png	960	374.6502	1201.176	884.2764	duck
18	40_0_2082.png	815.5633	347.7966	986.7449	832.7099	duck
19	40_2_77.png	768.5374	345.0979	929.7691	823.3069	duck

(->)

labels_test	2022-01-14 오전 9:07	JSON 원본 파일	10KB
labels_train	2022-01-14 오전 9:06	JSON 원본 파일	756KB

Detectron2에서 제공하는 Pre-trained모델을 사용하려면 Dataset을 **COCO Json**으로 만들어야 함
COCO Json을 만들고 등록해서 사용

진행 과정

- COCO Dataset 제작 함수 (1)
  - *dataset/Duck-farm-604-export.csv* -> 파싱 (id, image_id, category_id, bbox)

```
# csv행을 읽은 개수를 나타내는 변수
global num1
num1=0
# parse_label 호출 횟수
global num2
num2=0

# coco json으로 만들기 위한 label 파싱 (id,image_id,category_id,bbox)
def parse_label(image_path, id, _annot_id):
    global num1,num2
    annot_id = _annot_id
    list_annot = []

    while(num1 < len(x)):
        item=x[num1]
        # annotation file(csv) 'image'와 name(image_path)가 같은지 확인
        if(item[0] != name(image_path)):
            break

        class_=0
        if(item[5]=='dead'):
            class_=1
        elif(item[5]=='slapped'):
            class_=2
```

```
        xmin = float(item[1])
        ymin = float(item[2])
        xmax = float(item[3])
        ymax = float(item[4])
        width = float(xmax - xmin)
        height = float(ymax - ymin)

        annotation = dict()
        annotation["id"] = int(annot_id) # 각 오브젝트에 대한 아이디
        annotation["image_id"] = int(id) # 영상 아이디
        annotation["category_id"] = int(class_)
        annotation["bbox"] = [xmin, ymin, width, height]
        annotation["iscrowd"]=1 # 여러 개이거나 뭉쳐 있는 경우 1, 아니면 0
        annotation["area"]=100.0
        list_annot.append(annotation)
        annot_id += 1
        num1+=1

    # 파싱한 것(딕셔너리), 개수 반환
    return list_annot, annot_id
```

진행 과정

- COCO Dataset 제작 함수 (2)
  - dataset/image/*.jpg* -> 파싱 (id, width, height, file_name)

```python
# coco json으로 만들기 위한 영상 파싱 (id,width,height,file_name)
def parse_image(image_path, id):
    _image = Image.open(image_path)
    width, height = _image.size
    file_name = os.path.basename(image_path)

    image = dict()
    image["id"] = id
    image["width"] = int(width)
    image["height"] = int(height)
    image["file_name"] = file_name
    _image.close()

    # 파싱한 것(딕셔너리) 반환
    return image
```

진행 과정

- Category 정보 생성
  - COCO Json을 만들기 위해서는 카테고리 정보가 담긴 리스트가 필요함

```
# 카테고리 정보 생성
categories = [
    {
        "id" : 0,
        "name" : "duck"
    },
    {
        "id" : 1,
        "name" : "dead"
    },
    {
        "id" : 2,
        "name" : "slapped"
    }
]

print(categories)
```

```
[{'id': 0, 'name': 'duck'}, {'id': 1, 'name': 'dead'}, {'id': 2, 'name': 'slapped'}]
```

진행 과정

- COCO Dataset 제작 함수 사용
  - 파싱한 결과물 -> train/test 분할 -> COCO Dataset을 각각 만듬

```
# 학습 데이터에 대한 coco json 만들기

# 파싱한 것 가져오기
images = []
annotations = []
annot_id = 0
for id, image_path in tqdm(enumerate(image_list[:-10])): # tqdm: 진행률 프로그레스 바
    # 이미지 파싱한 것 저장
    images.append(parse_image(image_path, id))
    # 라벨 파싱한 것 저장
    parsed_annotations, annot_id = parse_label(image_path, id, annot_id)
    for annotation in parsed_annotations:
        annotations.append(annotation)

# labels_train.json 생성
coco_format = dict()
coco_format["info"] = "duck train dataset"
coco_format["images"] = images
coco_format["annotations"] = annotations
coco_format["licenses"] = "..."
coco_format["categories"] = categories
with open("dataset/labels_train.json", "w") as f:
    json.dump(coco_format, f)
```

진행 과정

- 학습 (Training)
  - config객체로 학습에 대한 모든 정보를 생성하고 학습을 진행함 (checkpoint를 저장하면서 학습이 진행됨)

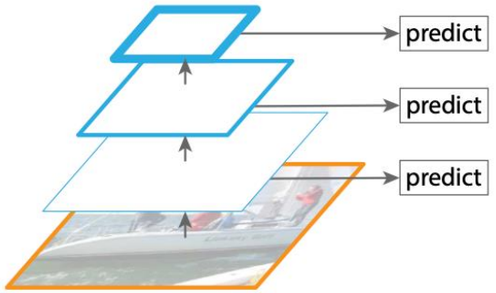
```
# config 객체를 선언하고 불러옵니다.
cfg = get_cfg()
# Detectron2에서 사전 정의한 faster_rcnn config 파일을 불러옵니다.
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_1x.yaml"))
# 학습 데이터셋
cfg.DATASETS.TRAIN = ("duck_train",)
cfg.DATASETS.TEST = ("duck_test",)
# 데이터를 불러오는 loader의 수를 정합니다. 일반적으로 'cpu 코어의 수 / 2'만큼 설정합니다. 큰 영향은 끼치지 않는 것 같습니다.
cfg.DATALOADER.NUM_WORKERS = 4
# pretrained checkpoint를 불러옵니다.
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_1x.yaml")
# batch size입니다.
cfg.SOLVER.IMS_PER_BATCH = 8
# 학습 수행 시의 iteration 횟수입니다.
cfg.SOLVER.MAX_ITER = 2000
# checkpoint를 저장하는 iteration 간격을 설정합니다.
cfg.SOLVER.CHECKPOINT_PERIOD = 50
# base learning rate를 설정합니다.
cfg.SOLVER.BASE_LR = 0.001
# WarmupMultiStepLR의 step을 설정합니다. 적용하지 않기를 원하기 때문에 빈 list를 넣었습니다.
cfg.SOLVER.STEPS = []  # do not decay learning rate
# retinanet config이 추론할 class의 수를 설정합니다.
cfg.MODEL.RETINANET.NUM_CLASSES = 3
```

resnet 50을 이용하는 FPN 기반 RetinaNet

```
#학습된 모델의 가중치 파일을 저장할 폴더 생성
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
# config 파일을 토대로 trainer를 불러옵니다.
trainer = DefaultTrainer(cfg)
# 가장 최근의 checkpoint를 불러오거나(resume=True인 경우) config 파일의 weights 링크로부터 weights를 불러옵니다.
trainer.resume_or_load(cfg)
# 학습을 진행합니다.
trainer.train()
```

관련 지식

- FPN (Feature Pyramid Network)
  - 연산량이 적으면서 성능이 좋은 multi-scale prediction 방식이다.
  - 기존 방식(c)은 resolution을 줄여가면서 feature를 추출할 때마다 prediction을 하는 방식이다.
    high resolution의 feature는 low resolution의 feature를 반영하지 못하기 때문에 각 prediction의 성능 차이가 발생한다.
  - Bottom-up 과정에서 resolution을 줄여가며 feature를 얻는다.
    Top-down 과정에서 up-sampling을 하고 같은 사이즈의 bottom-up 레이어와 합쳐서 손실된 지역적 정보를 보충한다.
    결국 high-level의 정보를 가지면서 위치 정보도 복원된 상태로 prediction을 진행할 수 있다.



(c) Pyramidal feature hierarchy

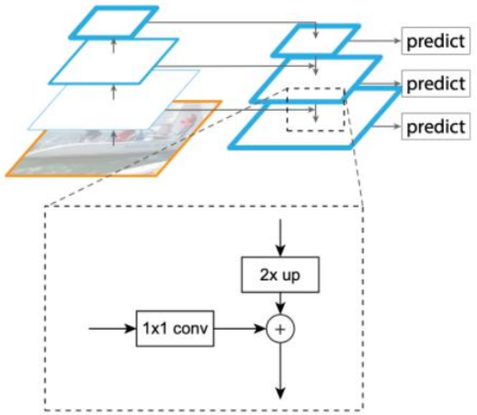
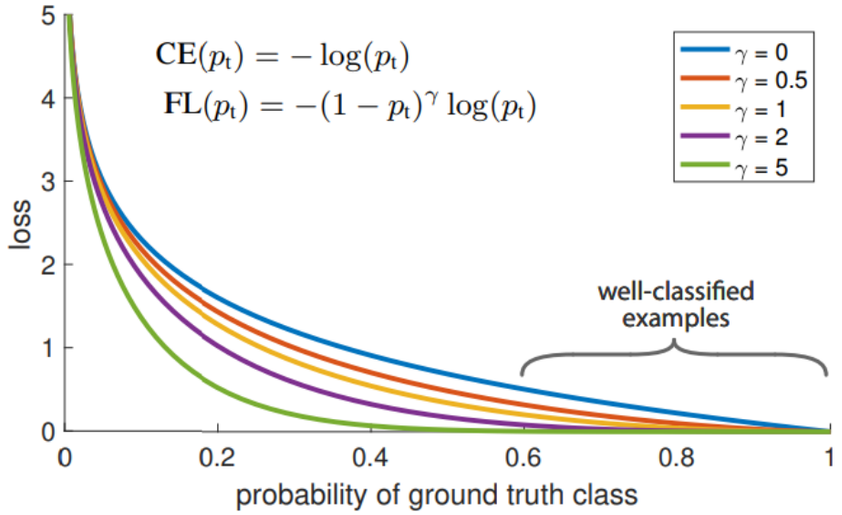


Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

관련 지식

- Focal Loss
  - one-stage detector는 foreground-background class imbalance가 극명하게 발생한다. (object보다 background가 훨씬 많음)
  - Focal Loss는 분류하기 쉬운 문제(Easy Negative Examples:background)보다 분류하기 어려운 문제(Hard Positive Examples:object)에 더 많은 가중치를 적용함으로써 객체 검출에 더욱 집중하여 학습을 진행한다.
  - p_t가 작은 경우(Hard Positive) moderating factor은 1에 가까워지고, p_t가 큰 경우(Easy Negative) 0에 가까워 진다.
  - CE랑 비교 시, p_t가 작은 경우(Hard Positive) Loss를 낮게 줄였고, p_t가 큰 경우(Easy Negative) Loss를 크게 줄였다. (CE: r=0)
    즉 찾기 쉬운 background에 대한 Loss가 현저히 낮으므로 backgound에 대해 많이 학습할 일이 없다.



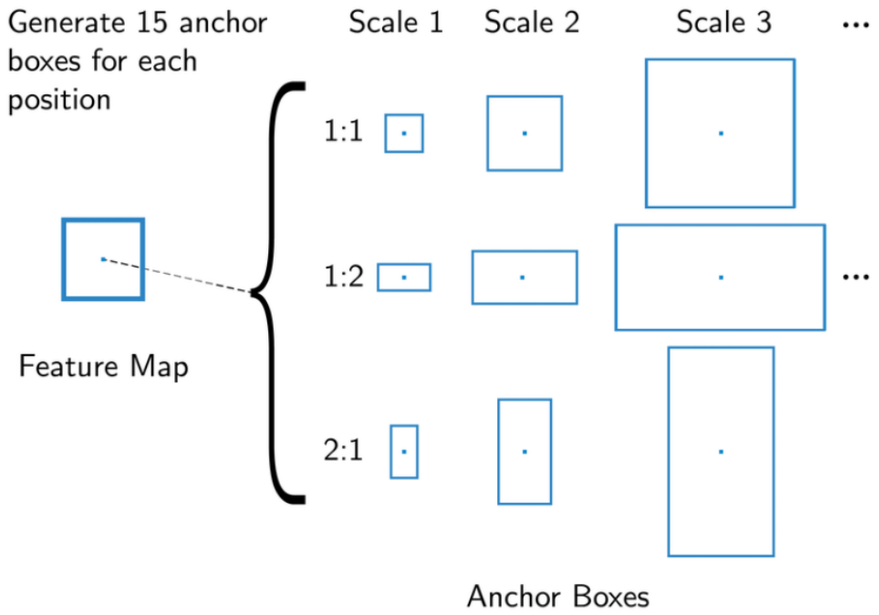
$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t).$$

focusing parameter: $\gamma \geq 0$

moderating factor: $(1-p_t)^{\gamma}$

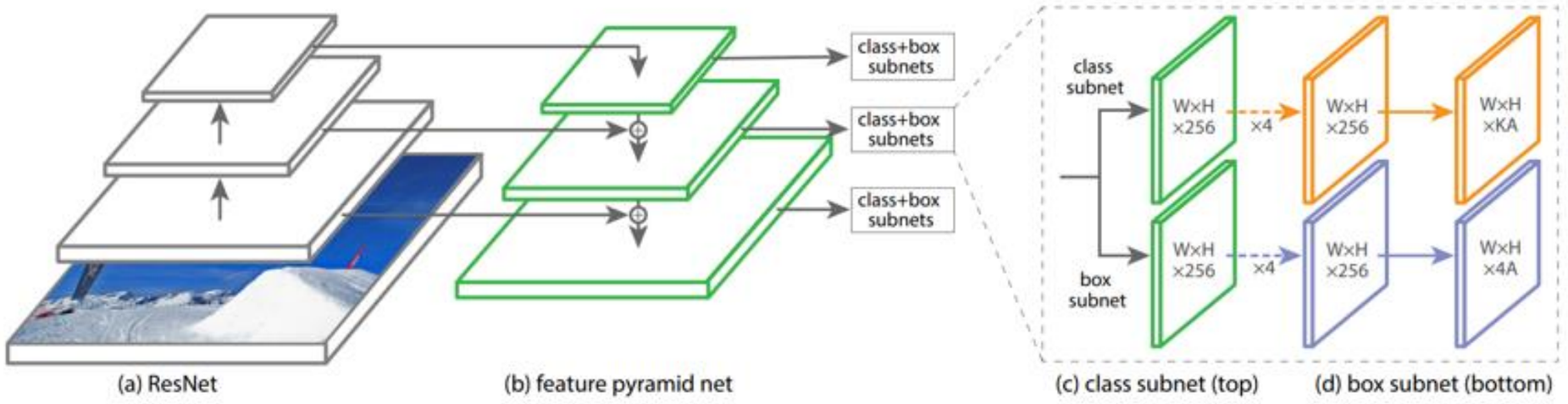
관련 지식

- Anchor Box
  - One-stage detector은 region proposal 대신 Feature Map자체에서 localization을 진행한다.
    이를 위해 anchor box(사전에 미리 정한 박스)를 사용하며, anchor box에 scales(길이 비율), aspect ratio(종횡비)를 적용한다.
  - RetinaNet은 {1:1, 1:2, 2:1} 기본 비율에 {2^0, 2^(1/3), 2^(2/3)} scale size를 적용한다.
    각 pyramid level 마다 9개의 anchor를 사용하며 scale의 범위는 32~813 pixel이다.
  - Anchor Box의 IoU 임계값은 0.5를 사용하며, IoU가 0~0.4면 background라고 정의한다.



관련 지식

- RetinaNet
  - unified network이며 backbone network와 두 개의 task-specific subnetworks로 구성되어 있다.
  - FPN을 이용해서 multi-scale prediction 성능을 향상시켰다.
  - class subnet은 각 위치에서 A개의 anchor들의 K개의 object class들의 존재 확률을 예측한다.
    box subnet은 각 anchor box의 offset 4개(x_center, y_center, width, height)를 Ground Truth Box와 유사하게 Regression한다.



작업 결과

- 인퍼런스 (Inference)
  - Defaultpredictor을 이용해서 테스트 데이터에 대해 인퍼런스를 진행함 (threshold=0.55)

전신이 다 나오는 오리일수록 정확도가 높은 것을 확인할 수 있음

작업 결과

- 평가 (Evaulation)
  - COCOEvaluator을 이용해서 여러 Task에 대해 AP(Average Precision)을 측정할 수 있음

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.625
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.917
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.720
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.719
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.138
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.711
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.719
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.719
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
[01/14 09:21:11 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 62.461 | 91.685 | 72.047 | 71.881 |  nan  |  nan  |
[01/14 09:21:11 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[01/14 09:21:11 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP   | category   | AP   |
|:-----------|:-------|:-----------|:-----|:-----------|:-----|
| duck       | 62.461 | dead       | nan  | slapped    | nan  |
OrderedDict([('bbox', {'AP': 62.460672363300226, 'AP50': 91.68480461665422, 'AP75': 72.04720844960023, 'APs': 71.88118811881188, 'APm': nan, 'APl': nan,
'AP-duck': 62.460672363300226, 'AP-dead': nan, 'AP-slapped': nan})])
```

죽은 오리와 누워 있는 오리에 대한 추가 학습이 필요함

데이터 분석

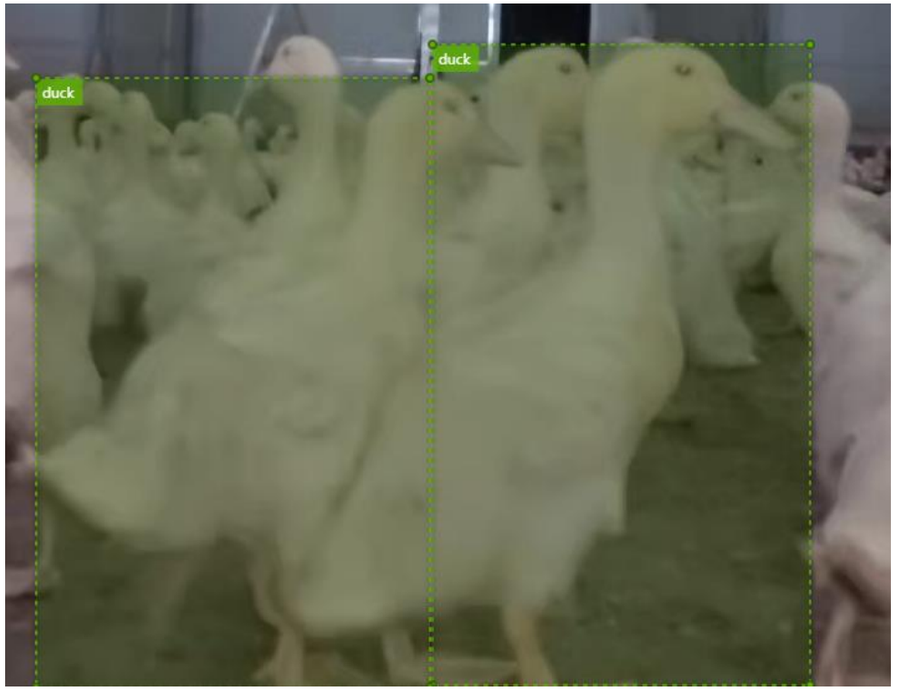
- Pre-Processing
  - 대표 이미지 선택 및 resizing

0: xmin=192, xmax=409, ymin=415, ymax=838
1: xmin=445, xmax=641, ymin=407, ymax=811
2: xmin=720, xmax=921, ymin=403, ymax=832
3: xmin=1038, xmax=1224, ymin=377, ymax=864
4: xmin=1277, xmax=1514, ymin=424, ymax=767
5: xmin=1521, xmax=1743, ymin=418, ymax=806
6: xmin=1696, xmax=1915, ymin=419, ymax=906

대표 이미지('32_2_270.png')를 기반으로 오리를 추출함 <랜덤 선택>
오리 이미지의 크기는 (300x500)으로 리사이징 함 <왜곡이 덜 되는 영상 크기로 리사이징>
추출한 오리 이미지들은 xmin을 기준으로 오름차순 정렬하였음

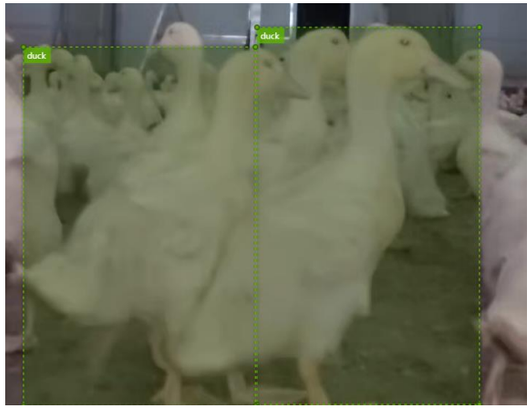
데이터 분석

- Question
  - 라벨링 중 겹치면서 오리의 몸이 전부 나오는 것이 좋은가? or 최대한 겹치지 않게 하는 것이 좋은가? 에 대한 의문 발생



데이터 분석

- Assume
  - 겹치지 않도록 라벨링을 하면 일반적인 오리의 Identity(머리,꼬리 등)마저 잃어 버리므로 성능이 안 좋을 것이다.

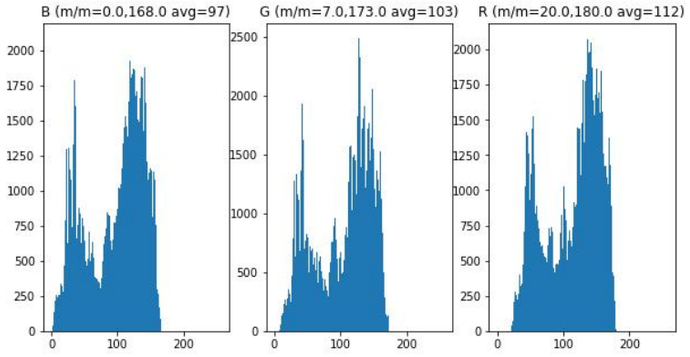


  - 따라서 겹치면서 오리의 Identity(머리,꼬리 등)을 잃지 않도록 하는 것이 성능에 도움이 될 것이다.

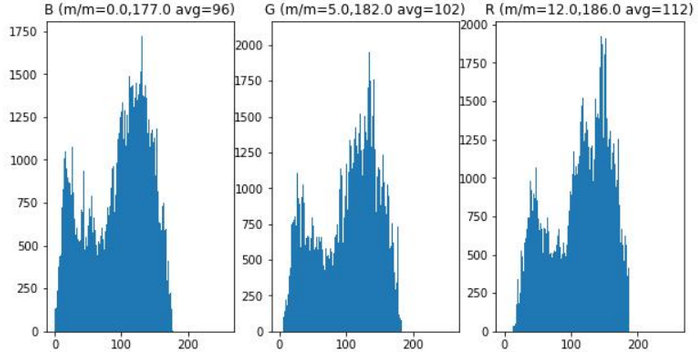
데이터 분석

- Test
  - 실제로 컴퓨터가 오리의 Identity를 머리, 꼬리라고 인식하는가? (->) 오리의 특징점 검출로 파악
  - 특징점을 검출하기 전 오리 이미지의 히스토그램을 파악

0: xmin=192, xmax=409, ymin=415, ymax=838



6: xmin=1696, xmax=1915, ymin=419, ymax=906

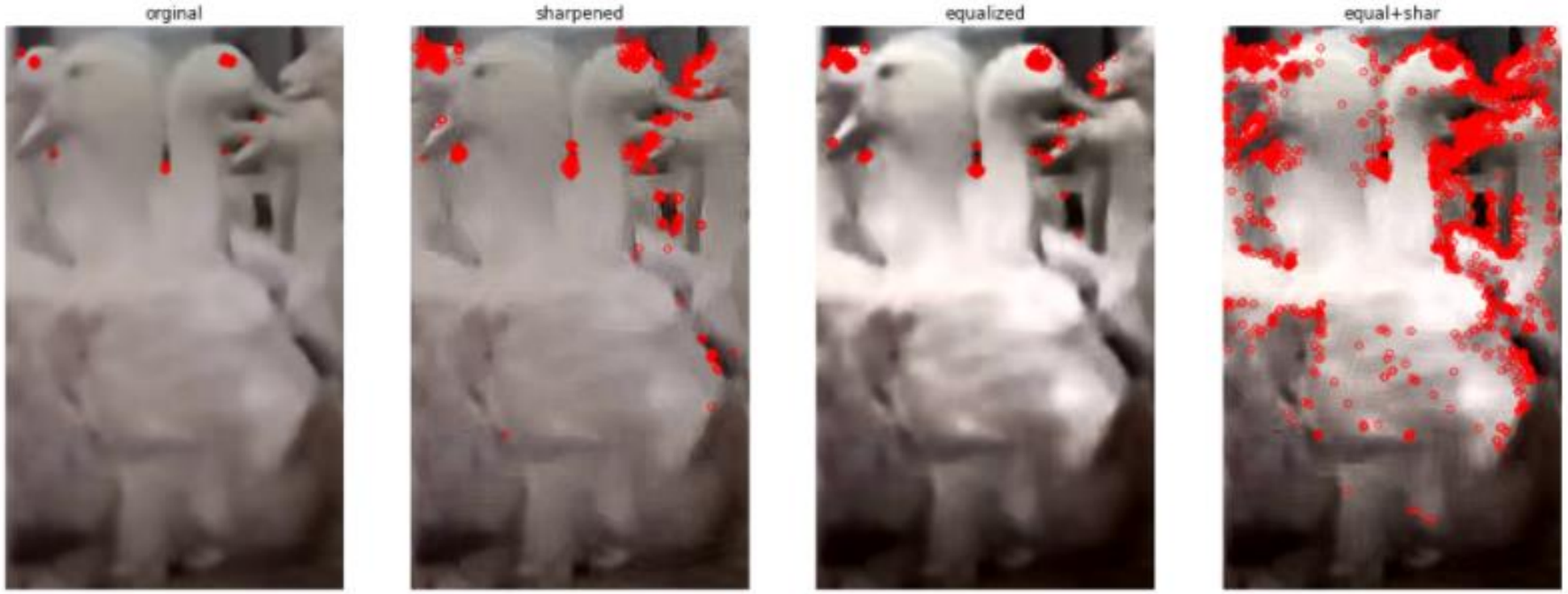


특징점을 확실히 파악하기 위해
영상처리 작업(평활화, 샤프닝 등)
이 필요하다고 생각함

데이터 분석

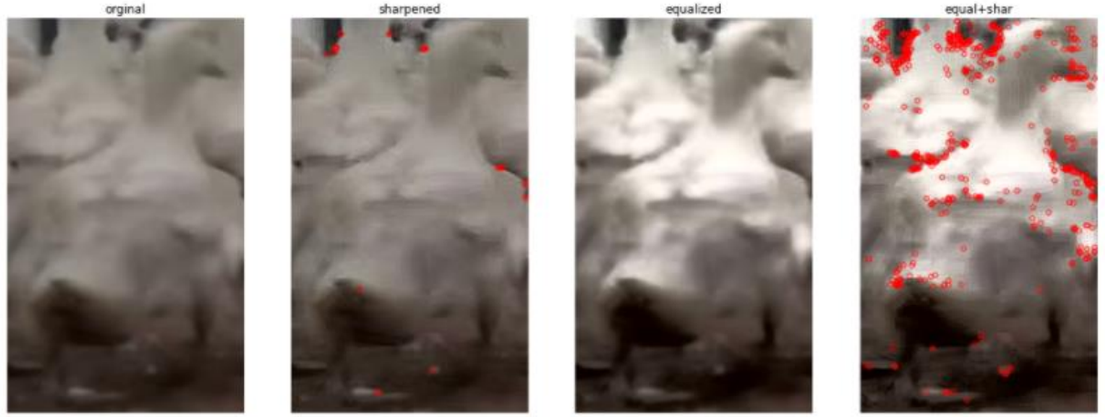
- Image Processing & Analysis
  - [원본 이미지|샤프닝한 이미지|평활화한 이미지|샤프닝+평활화한 이미지]에 대해 코너를 검출함
  - FAST 코너 검출 방법을 이용함

```
4: xmin=1277, xmax=1514, ymin=424, ymax=767
```

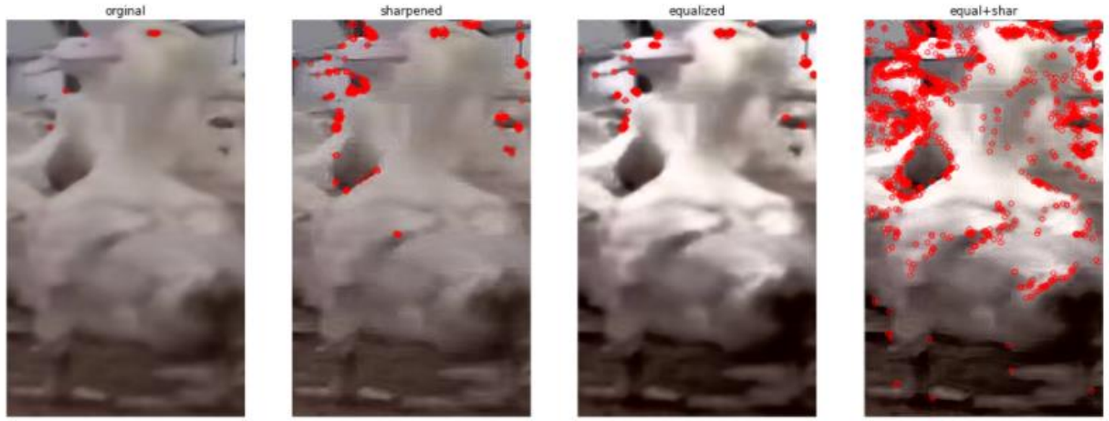


데이터 분석

0: xmin=192, xmax=409, ymin=415, ymax=838

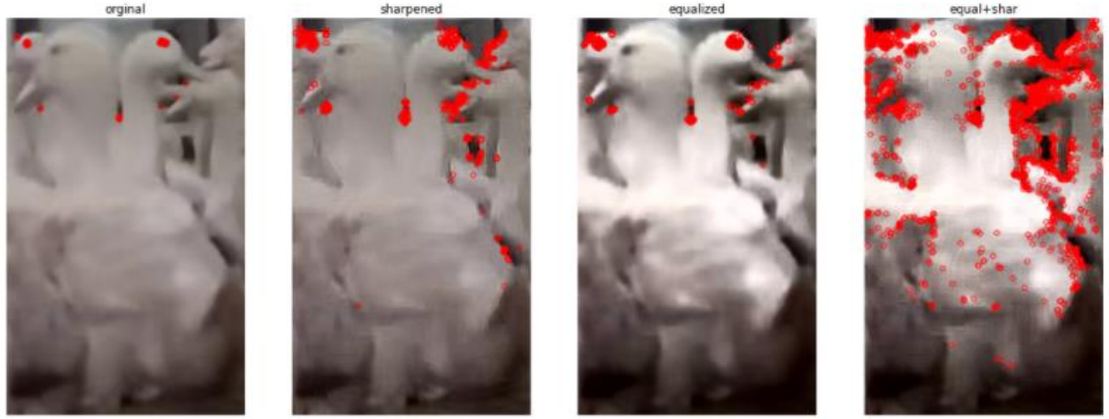


3: xmin=1038, xmax=1224, ymin=377, ymax=864



1: xmin=445, xmax=641, ymin=407, ymax=811

4: xmin=1277, xmax=1514, ymin=424, ymax=767

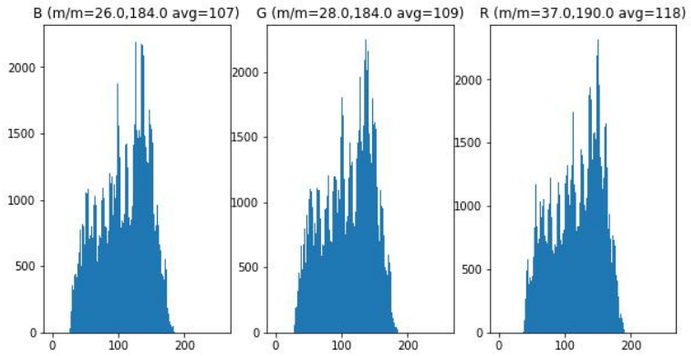


오리의 **머리 부분**에 코너가 많이 검출됨 -> 오리의 Identity가 머리인 것은 확인이 됨

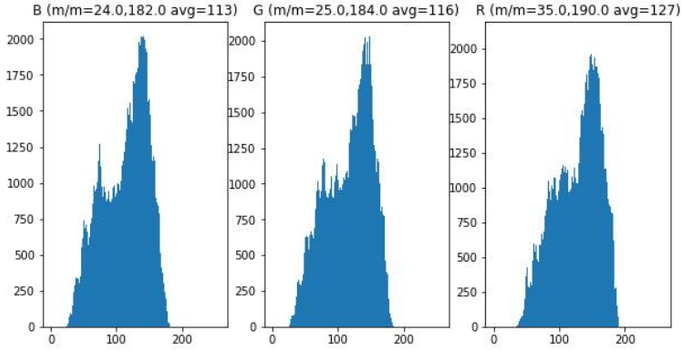
데이터 분석

- Test
  - 실제로 컴퓨터가 오리의 Identity를 머리, 꼬리라고 인식하는가? (->) 오리의 특징점 검출로 파악
  - 오리의 평균 이미지에 대해서도 조사

0: 40_2_0.png

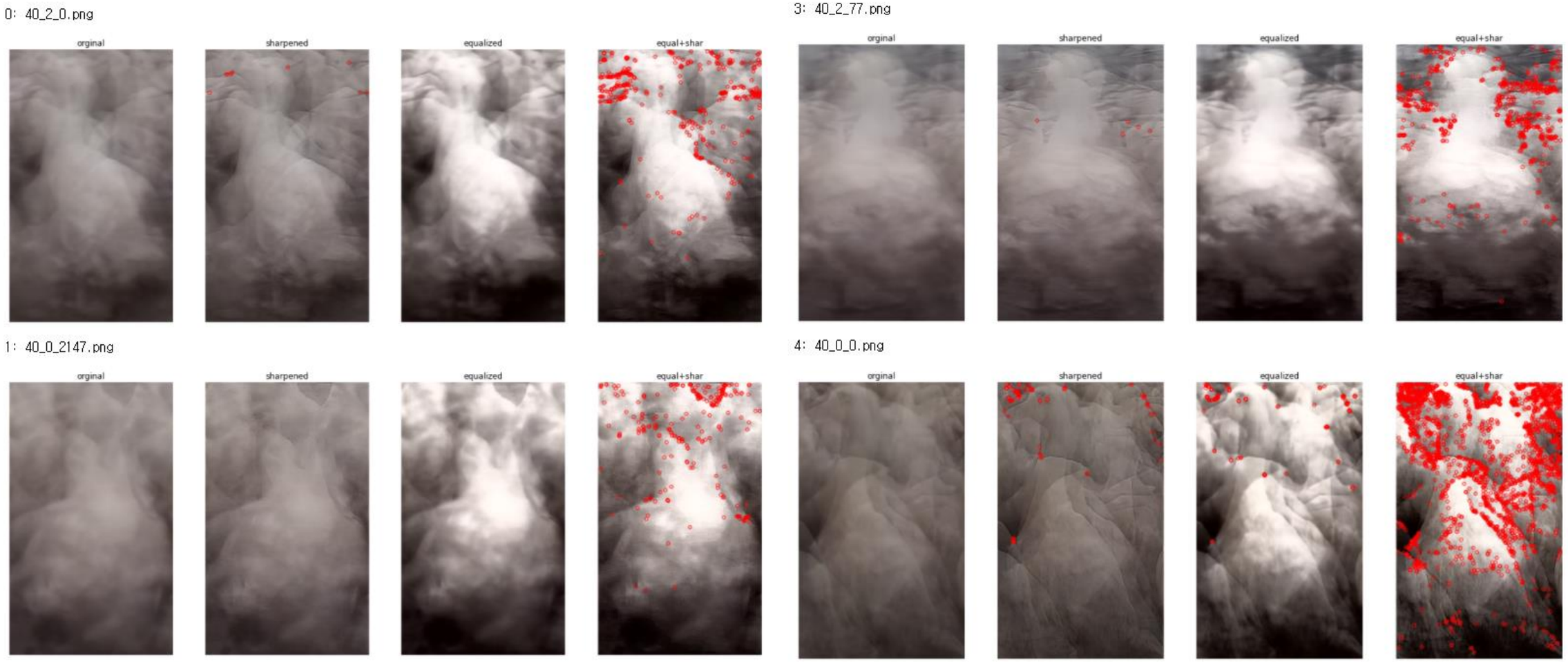


9: 40_2_233.png



누적 영상을 구하고 영상 개수로 나누는 방식으로
평균 이미지를 구함
-> 전체 오리에 대해 적용 시 형태를 완전히 잃어 실험이 불가능함
-> 랜덤으로 얻은 10개의 영상을 기준으로 평균 영상 10개를 생성함
-> 영상처리 후 코너를 검출해 봄

데이터 분석



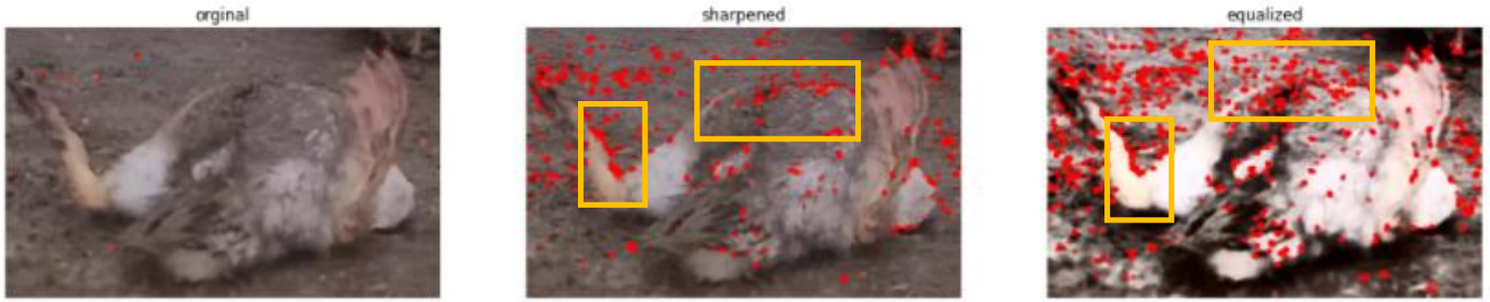
오리의 **머리로 추정되는 부분**에 코너가 많이 검출됨 -> 오리의 Identity가 머리인 것은 확인이 됨

Conclusion. 라벨링 진행 시 오리가 겹치더라도 머리 부분은 필수로 들어가야 됨

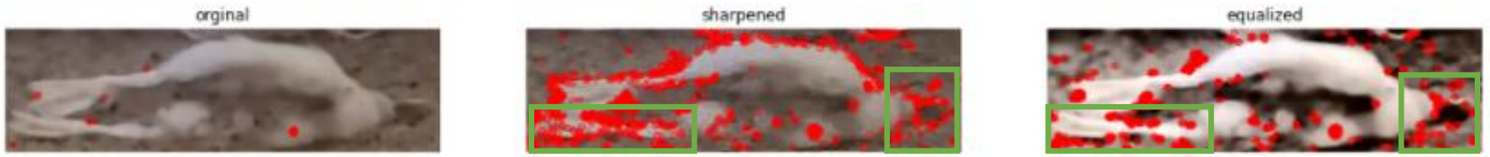
데이터 분석

- Question
  - 비정상 오리는 특징점이 어떻게 될까?

- Test
  - [죽은 오리|뒤집힌 오리]에 대해서도 FAST 코너 검출을 진행함



(->) [오리의 배, 몸통-발 사이]로 검출됨



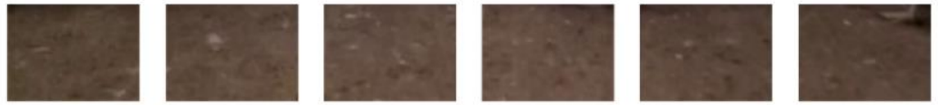
(->) [오리의 부리, 발]로 검출됨

그러나 데이터가 너무 적어 평균 이미지와 비교할 수 없으므로 참고용 분석임

데이터 분석

- 번외
  - 배경의 RGB 분포

0: xmin=0, xmax=300, ymin=0, ymax=300
1: xmin=300, xmax=600, ymin=0, ymax=300
2: xmin=600, xmax=900, ymin=0, ymax=300
3: xmin=900, xmax=1200, ymin=0, ymax=300
4: xmin=1200, xmax=1500, ymin=0, ymax=300
5: xmin=1500, xmax=1800, ymin=0, ymax=300
6: xmin=0, xmax=300, ymin=850, ymax=1070
7: xmin=300, xmax=600, ymin=850, ymax=1070
8: xmin=600, xmax=900, ymin=850, ymax=1070
9: xmin=900, xmax=1200, ymin=850, ymax=1070
10: xmin=1200, xmax=1500, ymin=850, ymax=1070
11: xmin=1500, xmax=1800, ymin=850, ymax=1070

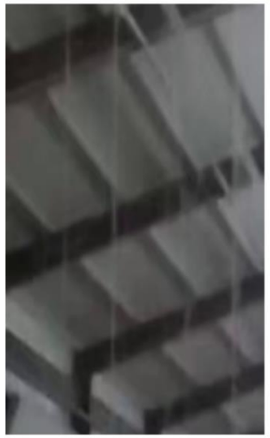


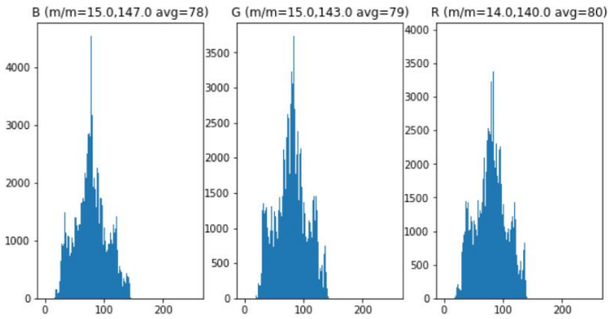
배경 이미지가 따로 없어서
(농장 천장/바닥)영상들을 만들어서 진행함

(->)

[천장 이미지 RGB 분포]

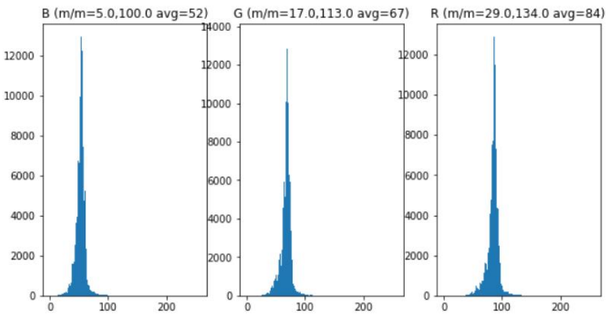
0: xmin=0, xmax=300, ymin=0, ymax=300 -> (resized: 500x300)





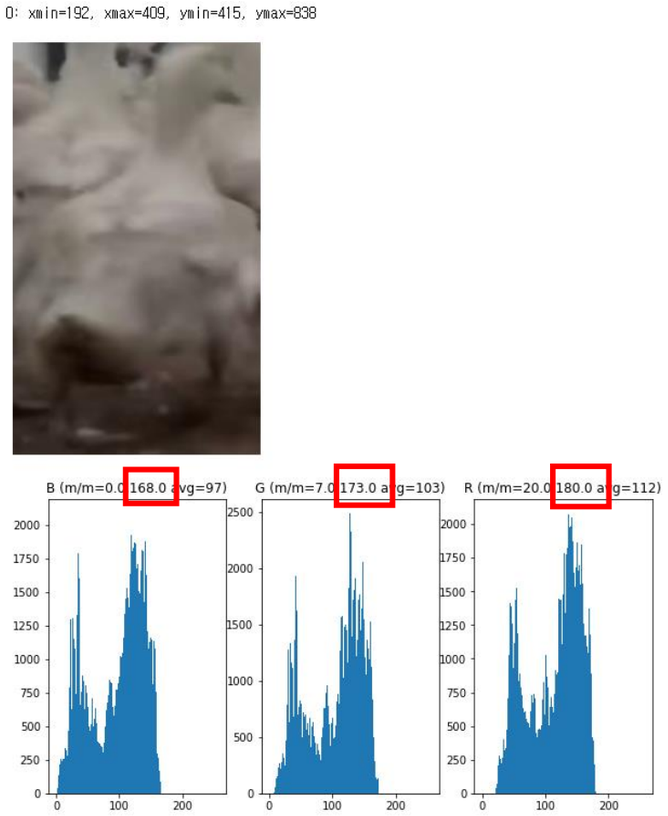
[바닥 이미지 RGB 분포]

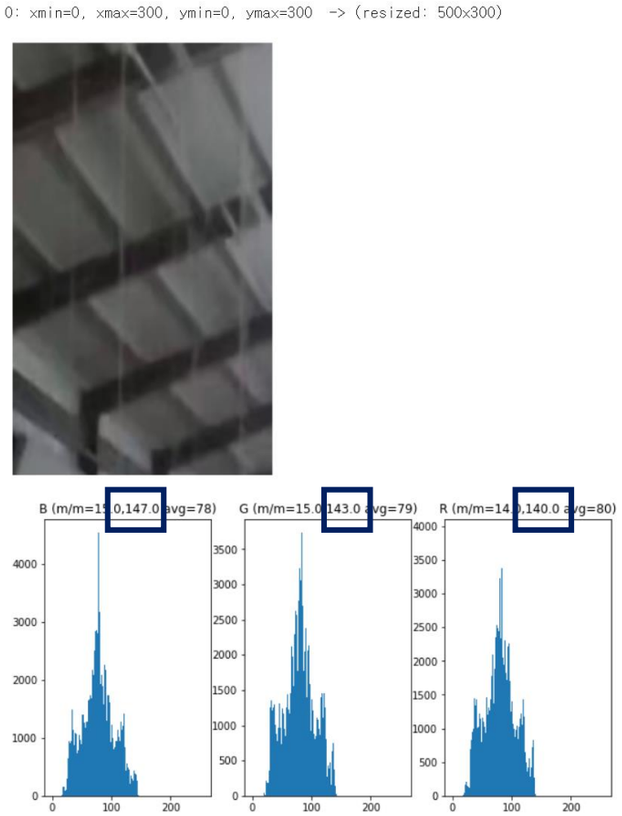
6: xmin=0, xmax=300, ymin=850, ymax=1070 -> (resized: 500x300)

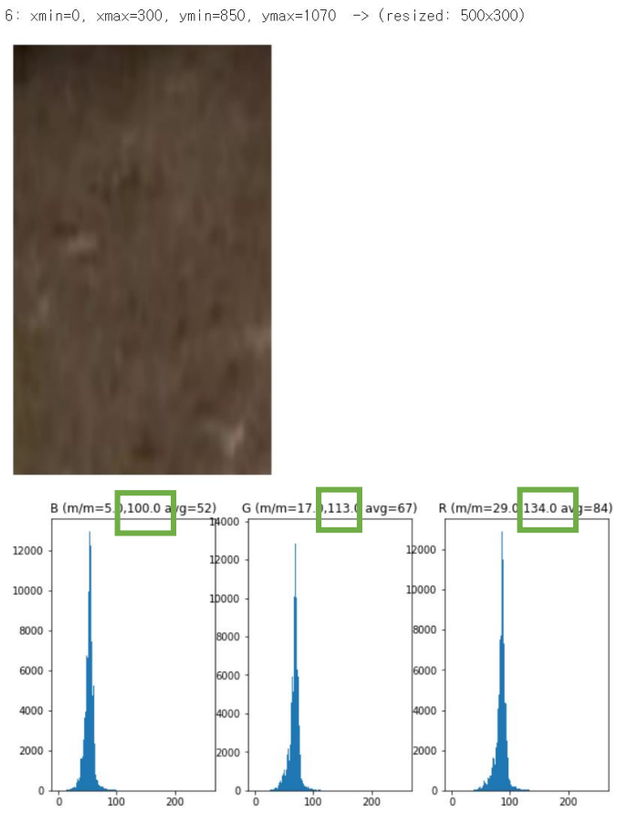


데이터 분석

- 번외
  - RGB 분포 비교





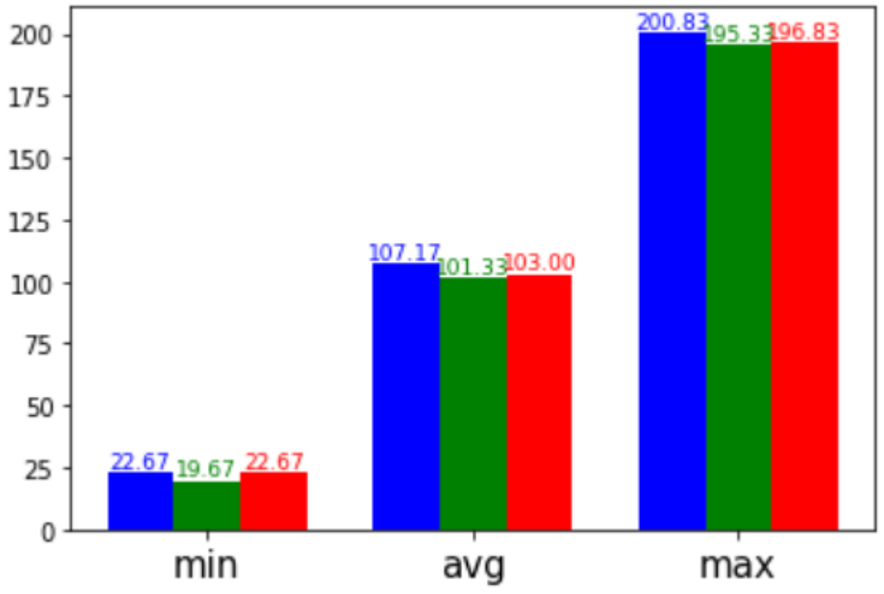


오리->천장->바닥 순으로 명암비가 낮음 / 오리 데이터의 max pixel이 가장 큼

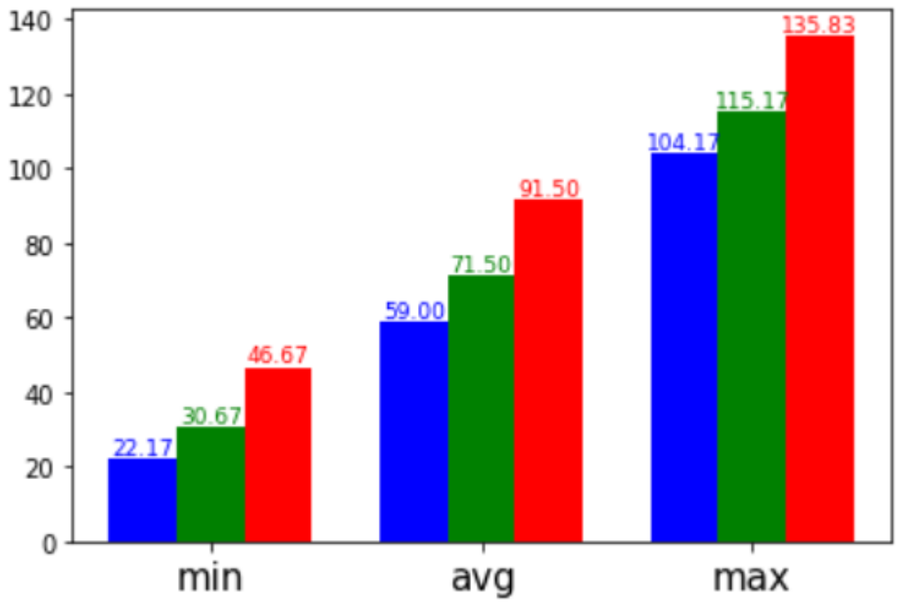
데이터 분석

- 번외
  - 배경의 RGB 평균

[천장 이미지 RGB 평균]



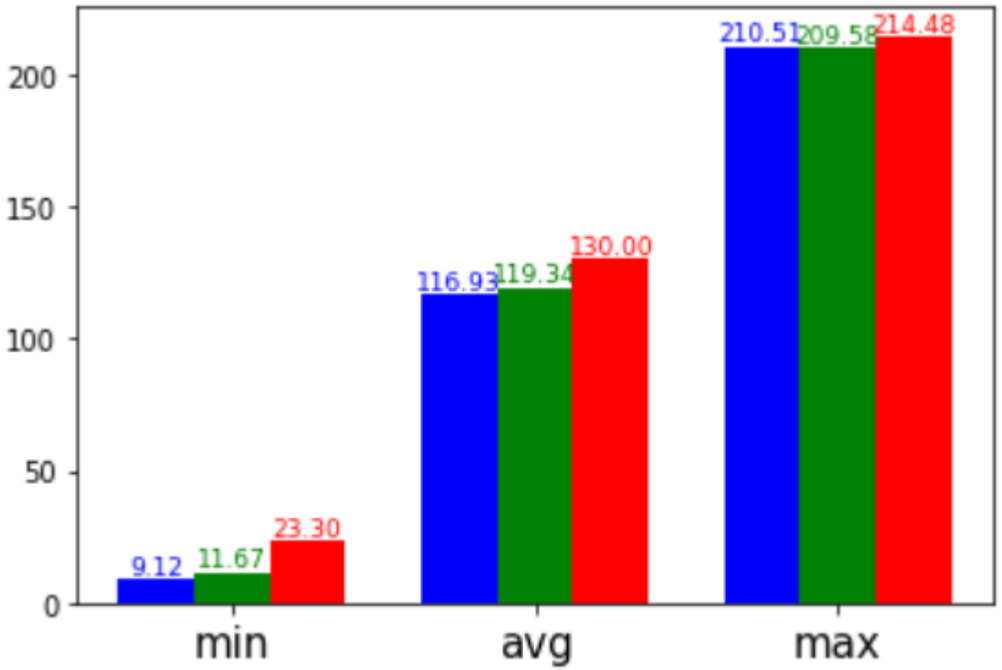
[바닥 이미지 RGB 평균]



6개의 천장/바닥 영상에 대한 RGB min 평균값, avg 평균값, max 평균값을 계산함

데이터 분석

- 번외
  - 전체 오리에 대한 RGB 평균



4509마리의 오리에 대한 RGB min 평균값, avg 평균값, max 평균값을 계산함

마무리

- 느낀점

CVMI LAB에서 처음 진행한 국가 과제였다. 학부 연구생으로 6개월 이상 활동했지만 처음 한 실습이었기에 막막함과 기대감이 동시에 들었다. 계속된 랩 미팅으로 이러한 막막함을 덜어주신 '이제영', '구정수' 선배님들께 감사하다.
데이터 분석 같은 경우 학부에서 배운 내용을 최대한 이용하려고 노력하였다. 그러나 RetinaNet과의 연관성을 따졌을 때 합리성이 부족하다는 평가를 받았다. 예를 들어 Fast 방식으로 특징점을 찾지 말고 Grad-Cam을 사용하면 좋았을 것이라는 의견이다. 앞으로 더 많은 논문을 읽고 누구에게나 납득이 갈 만한 분석을 해야겠다는 다짐이 들었다.
그래도 (라벨링,학습,분석 등) 많은 실습을 해보며 칭찬도 듣고 과제에 대한 자신감도 생긴 것 같다.
앞으로 어떤 과제가 오더라도 해낼 수 있는 사람이 되기 위해 정진하겠다.
마지막으로 학부 연구생으로 좋은 경험을 하게 해주신 '강호철 교수님'께 감사하다.

- 참고자료

- (FPN) https://velog.io/@haejoo/Feature-Pyramid-Networks-for-Object-Detection-%EB%85%BC%EB%AC%B8-%EC%A0%95%EB%A6%AC
- (Focal Loss) https://gaussian37.github.io/dl-concept-focal_loss/
- (RetinaNet) https://ropiens.tistory.com/83 https://csm-kr.tistory.com/5
- (RetinaNet) https://csm-kr.tistory.com/5
- (기타 자료) CVMI Lab Inner File

CVMI LAB - 201921725 안성현

감사합니다

<2022/02/05>