

불량품 판별 웹서비스



김승연

이동훈

최강현

지도교수: 박진선교수님

목 차

1. 서론	1
1.1. 연구 배경	1
1.2. 연구 목표	1
2. 연구 배경 지식	1
2.1. Pytorch	1
2.2. YOLOv5	1
2.3. Same Same but DifferNet	2
2.4. Flask	2
3. 연구 내용	4
3.1. 데이터 준비	4
3.1.1. YOLOv5 데이터 준비	4
3.1.2. DifferNet 데이터 준비	5
3.2. 모델 설계	6
3.2.1. YOLOv5 모델	6
3.3. 모델 학습	6
3.3.1. YOLOv5 모델 학습	6
3.3.2. DifferNet 모델 학습	7
3.4. 웹서비스 개발	7
3.4.1 페이지 구성	7
3.4.2 기능 및 동작 설명	8
4. 연구 결과 분석 및 평가	8
4.1. YOLOv5 Object-Detection 결과 분석 및 평가	8
4.2. DifferNet Anomaly-Detection 결과 분석 및 평가	14

5. 결론 및 향후 연구 방향	15
6. 구성원별 역할 및 개발 일정	16
6.1. 구성원별 역할	16
6.2. 개발일정	16
7. 참고 문헌	17

1. 서론

1.1. 연구 배경

현실에서 불량품을 확인하기 위해 인간이 직접 수작업으로 상태를 확인하고 분류한다. 그 수가 적게는 수십 명 많을 경우에는 수백 명의 직원이 불량 검사에 투입된다. 하지만 그렇게 많은 사람이 불량 판별을 해도 오류 없이 완벽하게 분류할 수는 없다. 심지어 물건을 구매한 소비자 또한 불량품인지 완전히 인지 못하는 경우도 존재한다. 이런 문제점들을 해결하기 위해 딥러닝의 CNN을 이용한 자동화된 불량 검출 시스템을 제안한다. 이를 사용하면 인간의 눈으로도 찾기 힘든 불량을 찾아낼 수 있다. 이미지 데이터를 수집해 학습시키고 웹이나 앱과 연동하여 많은 사람들에게 서비스를 제공할 수 있을 것으로 판단되어 주제를 이와 같이 선정하였다.

1.2. 연구 목표

본 졸업 과제는 과일 이미지를 넣었을 때 사용자가 직접 눈으로 보고 과일을 선별하지 않아도, 머신러닝을 통해 자동으로 결함의 위치와 그 종류를 웹에서 판별하여 편의를 제공함에 의의를 둔다. 크게는 상하거나 곰팡이 핀 부분, 작게는 단순한 흠집까지 판별할 수 있는 시스템을 만든다.

2. 연구 배경 지식

2.1. Pytorch

PyTorch는 Python을 위한 오픈소스 머신 러닝 라이브러리이다. Torch를 기반으로 하며, 자연어 처리와 같은 애플리케이션을 위해 사용된다. GPU 사용이 가능하기 때문에 속도가 상당히 빠르다. 아직까지는 Tensorflow의 사용자가 많지만, 직관적이지 않은 구조와 난이도 때문에, Pytorch의 사용자가 늘어나고 있는 추세이다. 이는 Facebook의 인공지능 연구팀이 개발했으며, Uber의 "Pyro"(확률론적 프로그래밍 언어) 소프트웨어가 Pytorch를 기반으로 한다

2.2. YOLOv5

YOLO는 You Only Look Once라는 이름으로, one-stage object detection 딥러닝 기법이다.

YOLO v4와 YOLO v5는 오리지널 YOLO의 저자와 다른 이가 연구 및 개발하였으며, v5는 pytorch를 사용하여 ios에 적용할 수 있다. YOLO v5는 4가지의 모델이 있는데, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x로 간단하게 small, medium, large, xlarge로 이름지

어져 있다. backbone이나 head는 모두 동일하지만, depth_multiple(model depth multiple)과 width_multiple(layer channel multiple)이 다르다. large가 1.0으로 기준이 되는 크기로 생각하면 된다.

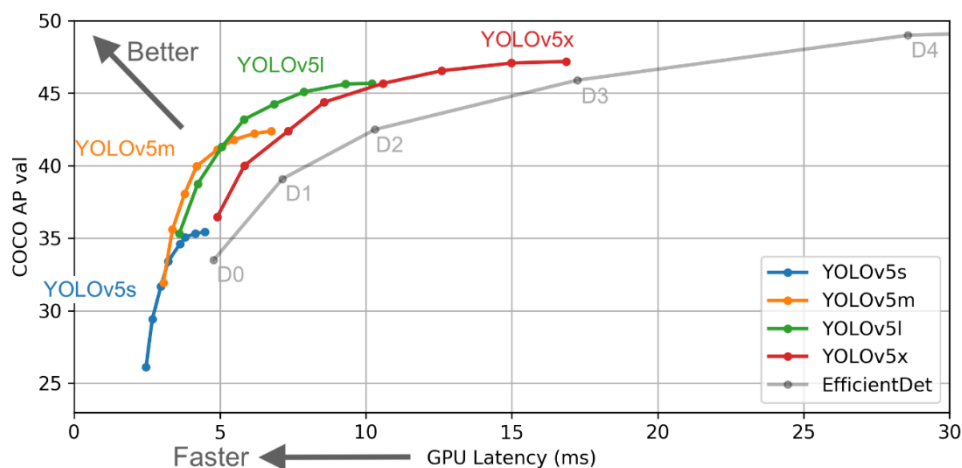


그림-1. Yolo v5 성능비교

2.3. Same Same but DifferNet

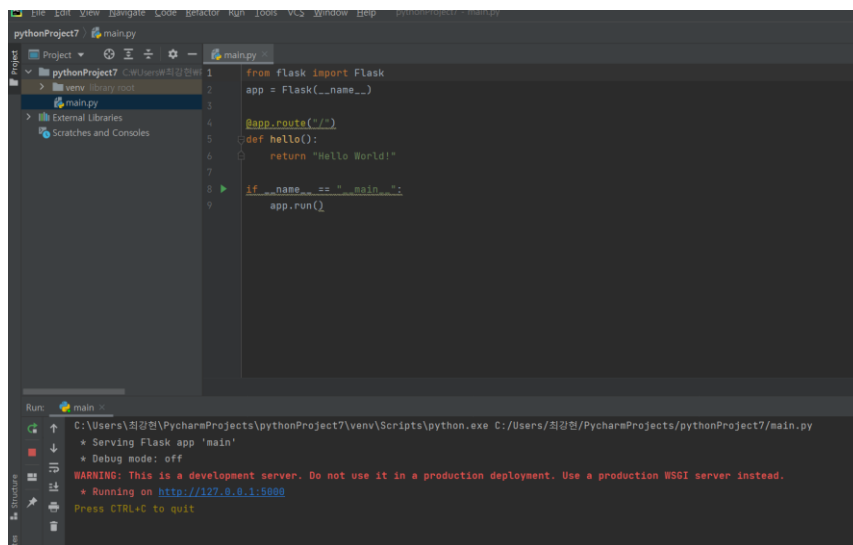
산업 제조 공정에서 제품의 품질은 지속적으로 모니터링되고 개선된다. 이런 이유로 제조과정 중 크고 작은 결함들이 안정적으로 감지되어야 한다. 그러나 제품마다 결점의 유형은 매우 다양하며, 대부분의 유형이 발생 사례가 거의 없을 정도로 드물게 나타난다. 심지어 결점 유형이 어느정도 알려진 이후에도 새로운 유형이 발생할 수 있다. 이와 같이 제조 중 예측하지 못한 사건이 발생하면 신뢰할 수 있는 결함 감지를 수행할 수가 없다. 이러한 배경으로 나타나게 된 Normalizing Flows를 이용한 Defect-Detection을 본 과제에도 적용시켜보고자 한다.

2.4. Flask

플라스크는 많은 사람들이 '마이크로 웹 프레임워크'라고 부른다. 여기서 '마이크로'는 '한 개의 파이썬 파일로 작성할 수 있다' 또는 '기능이 부족하다' 와 같은 의미가 아니라 프레임워크를 간결하게 유지하고 확장할 수 있도록 만들었다는 뜻이다.

플라스크에는 폼(form), 데이터베이스(database)를 처리하는 기능이 없다. 예를 들어 장고라는 웹 프레임워크는 프레임워크 자체에 폼과 데이터베이스를 처리하는 기능이 포함되어 있다. 장고는 쉽게 말해 덩치가 큰 프레임워크다. 그러면 플라스크는 이런 기능을

어떻게 보완할까? 플라스크는 확장 모듈을 사용하여 이를 보완한다. 이 말은 플라스크로 만든 프로젝트의 무게가 가볍다는 것을 의미한다. 왜냐하면 플라스크는 처음부터 모든 기능을 포함하고 있지 않기 때문이다. 그때그때 개발자가 필요한 확장 모듈을 포함해 가며 개발하면 된다. 플라스크는 자유도가 높은 프레임워크다. 프레임워크는 대부분 규칙이 복잡하고 개발자는 그 규칙을 반드시 따라야 한다. 규칙을 따라야 하는 건 플라스크도 마찬가지다. 하지만 플라스크에는 최소한의 규칙만 있으므로 개발의 자유도는 다른 프레임워크보다 높은 편이다.



```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     app.run()
```

Run: main

- * Serving Flask app 'main'
- * Debug mode: off
- WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
- * Running on <http://127.0.0.1:5000>
- Press CTRL+C to quit

그림-2. Flask 간단한 코드

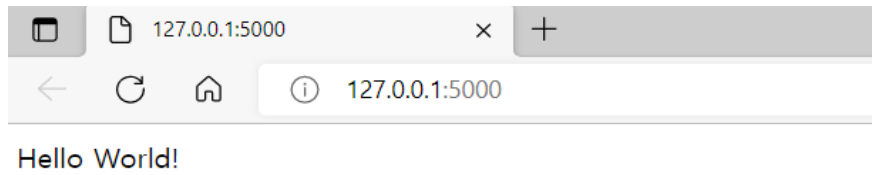


그림-3. 웹 페이지에 구현된 모습

3. 연구 내용

3.1. 데이터 준비

3.1.1. YOLOv5 데이터 준비

모델 학습을 위한 이미지 데이터는 Kaggle dataset을 이용해서 불량이 있는 과일들의 이미지를 다운받고, yolo로 학습을 시키기 위해 결함 위치 정보를 txt 파일로 만들어줘야 했다. 그래서 roboflow를 이용해 직접 결함 부위를 labeling하여 txt 파일로 만들어줬다. 아래 <그림4>처럼 train할 모든 data들의 결함 부위를 전부 labeling 한 후, txt 파일 생성과 동시에 train-70%, validation-20%, test-10%으로 분배해서 데이터셋을 만들었다. 그 다음 yaml 파일에서 앞서 labeling한 이미지들과 txt 파일 경로를 설정하고, class의 개수(nc)와 각 class들의 이름(names)을 설정해 준다. 아래 <그림5>는 Apple dataset의 예시이다.

총 15종류의 과일로 구성하였고, 각 과일당 50장씩 750장의 이미지를 각 object 별로 labeling을 한 후 Train-Dataset을 구성하였다.



그림-4. Roboflow를 이용해 labeling한 모습

```
train: coco128/images/train
val: coco128/images/valid
test: #test-dev2017.txt

nc: 3
names: ['fungi', 'rotted', 'withered']
```

그림-5. Labeling 한 후 설정한 yaml파일

3.1.2. DifferNet 데이터 준비

우선 DifferNet의 데이터 구성은 다음과 같다.

Fruit - train - good

-test - good

- defect

위처럼 train 폴더에는 정상적인 데이터를 넣어 train시켰고, defect 폴더에는 결함이 있는 데이터를 넣어 test를 진행하고 결함을 찾아내게 하였다. 총 15종류의 과일로 구성하였고, 각 과일당 50장씩 750장의 정상 이미지를 사용하였다.

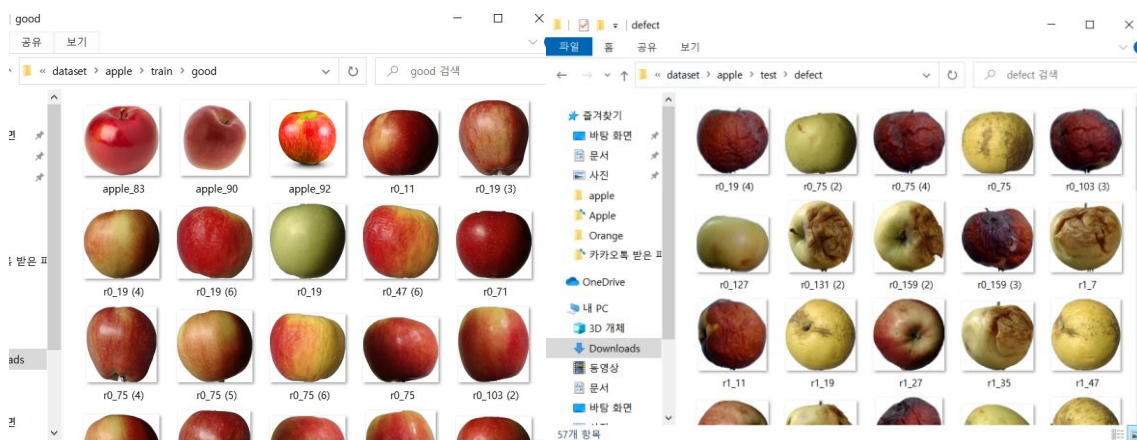


그림-6. train/good에 train위한 정상 Apple dataset 그림-7. test/defect에 test위한 비정상 Apple dataset

3.2. 모델 설계

3.2.1. YOLOv5 모델

위 그림-1을 통해서 YOLOv5의 model들의 성능을 볼 수 있는데 그 중에서 성능은 낮지만 학습 속도가 빠른 YOLOv5s를 선택했다. 그 이유는 다른 model과 비교를 해도 YOLOv5가 성능도 괜찮기 때문에 학습시간이 많이 느리고 성능이 좋은 걸 선택하기보다 학습 속도가 빠른 걸 이용해 여러 번 더 많이 학습을 시키고자 했다.

	from	n	params	module	arguments
0	-1	1	3520	models.common.Conv	[3, 32, 6, 2, 2]
1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]
2	-1	1	18816	models.common.C3	[64, 64, 1]
3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]
4	-1	2	115712	models.common.C3	[128, 128, 2]
5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]
6	-1	3	625152	models.common.C3	[256, 256, 3]
7	-1	1	1180672	models.common.Conv	[256, 512, 3, 2]
8	-1	1	1182720	models.common.C3	[512, 512, 1]
9	-1	1	656896	models.common.SPPF	[512, 512, 5]
10	-1	1	131584	models.common.Conv	[512, 256, 1, 1]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	models.common.Concat	[1]
13	-1	1	361984	models.common.C3	[512, 256, 1, False]
14	-1	1	33024	models.common.Conv	[256, 128, 1, 1]
15	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
16	[-1, 4]	1	0	models.common.Concat	[1]
17	-1	1	90880	models.common.C3	[256, 128, 1, False]
18	-1	1	147712	models.common.Conv	[128, 128, 3, 2]
19	[-1, 14]	1	0	models.common.Concat	[1]
20	-1	1	296448	models.common.C3	[256, 256, 1, False]
21	-1	1	590336	models.common.Conv	[256, 256, 3, 2]
22	[-1, 10]	1	0	models.common.Concat	[1]
23	-1	1	1182720	models.common.C3	[512, 512, 1, False]
24	[17, 20, 23]	1	21576	models.yolo.Detect	[3, [[10, 13, 16, 30, 33, 23],
[30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]					
Model summary: 214 layers, 7027720 parameters, 7027720 gradients, 16.0 GFLOPs					

그림-8. YOLOv5 모델 구조

3.3. 모델 학습

3.3.1. YOLOv5 모델 학습

Pre-trained된 YOLOv5s 모델을 사용하여 우리가 찾고자 하는 과일의 defect를 학습하였다. 아래와 같이 image_size는 640, 300 epoch, 16 batch_size로 설정을 하고, 3060 두 개의 GPU로 학습하였다. 아래의 명령어를 통해 수행되었다.

```
python train.py --img 640 --batch 16 --epochs 300 --data coco128.yaml --weights http://yolov5s.pt --device 0,1
```

과일들의 결함 dataset의 경우 처음에는 100 epoch으로 40장의 image를 학습시켰는데 "withered" object를 잘 찾아내지 못했다. 그 후 Dataset의 양과 epoch 수를 조금씩 늘리면서 학습한 결과, Dataset을 15종류의 과일로 구성을 하였고, 각 과일 당 50장 총 750장으로 학습시켰다. 300 epoch으로 학습을 시켰을 때의 결과가 가장 object를 잘 detection 하였다.

3.3.2. DifferNet 모델 학습

```
Train epoch 0
Epoch: 0.0      train loss: 2.1827
Epoch: 0.1      train loss: 0.3876
Epoch: 0.2      train loss: 0.3514
Epoch: 0.3      train loss: 0.2017
Epoch: 0.4      train loss: 0.1467
Epoch: 0.5      train loss: 0.0744
Epoch: 0.6      train loss: 0.0327
Epoch: 0.7      train loss: 0.0261

Compute loss and scores on test set:
Epoch: 0      test_loss: 12476503040.0000
AUROC: last: 0.9725 max: 0.9725 epoch_max: 0

Train epoch 1
Epoch: 1.0      train loss: -0.0508
Epoch: 1.1      train loss: -0.1160
Epoch: 1.2      train loss: -0.1580
Epoch: 1.3      train loss: -0.1584
Epoch: 1.4      train loss: -0.2048
Epoch: 1.5      train loss: -0.2690
Epoch: 1.6      train loss: -0.3255
Epoch: 1.7      train loss: -0.3810

Compute loss and scores on test set:
Epoch: 1      test_loss: 29119905792.0000
AUROC: last: 0.9802 max: 0.9802 epoch_max: 1
```

그림-9. DifferNet 학습 모습

training epoch를 100로, sub_epoch 10로 주었다. Data 사진의 경우에는 정상 데이터셋은 15종류의 과일로 학습을 하였고, 각 과일 당 50장 총 750장으로 학습시켰다.

3.4. 웹서비스 개발

3.4.1 페이지 구성

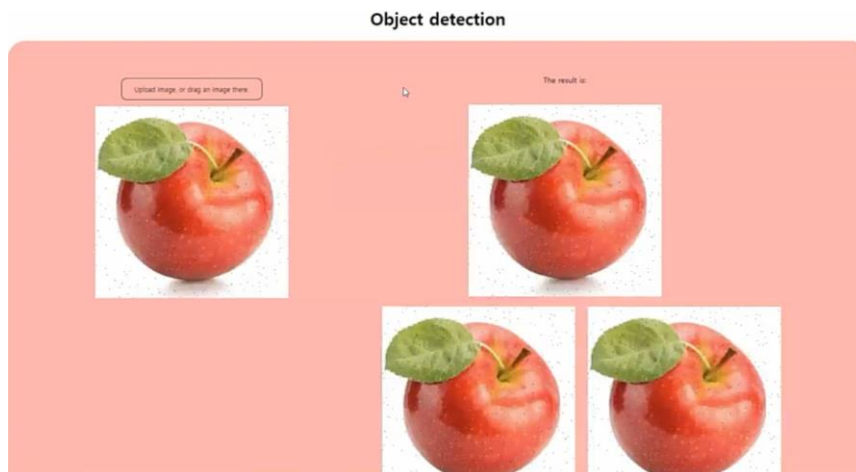


그림-10. 웹 페이지 UI

웹의 기본 UI는 다음과 같다. 왼쪽에 혼자 떨어져 있는 곳이 판별하기 위한 과일의 사진을 업로드하는 곳이고 오른쪽에서 위쪽은 과일의 종류를 판별(object detection)해준 결과, 오른쪽에서 왼쪽 밑은 결함을 판별(defect detection)해준 결과, 오른쪽 밑은 결함부위에 대한 Gradient map이 보여질 것이다. 자세한 결과는 그림-11에서 확인할 수 있다.

3.4.2 기능 및 동작 설명

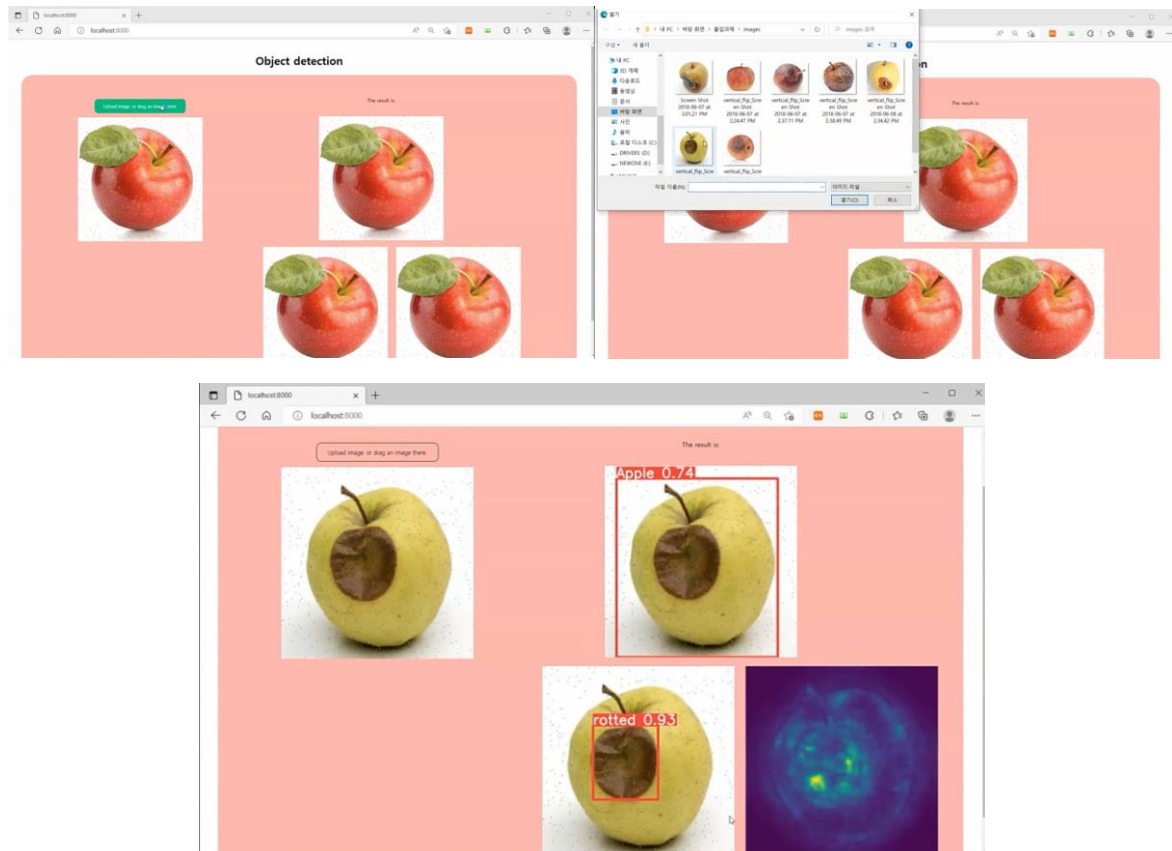


그림-11. Img 업로드 방법, 결과

위의 2개의 사진에는 과일 이미지를 업로드 하는 방법이 나타나 있고, 아래 사진은 이미지 업로드 몇 초 후 결과가 뜨는 사진이다. 업로드한 사진이 '사과'인 것을 오른쪽 위의 사진이 판별해주고 그 밑에는 결함의 종류와 위치, 그리고 정도를 판별해준 것이다.

4. 연구 결과 분석 및 평가

4.1.Yolov5 Object-Detection 결과 분석 및 평가

처음에는 다양한 종류의 과일로부터 다양한 결함 종류를 얻어 이용하려 했으나, data 를 구하는데 어려움을 겪어 과일의 종류를 줄였고 결함 종류 역시 구한 data 에서 특정 2 개 정도 반복되는 데이터 밖에 구하지 못해 결함 종류 또한 세 종류로 줄였다. 특히, 처음에는 scratch 라는 결함 종류를 사용했으나 그 data 수가 적어 scratch 를 scratch 로 인식하지 못하고 심지어 정상 사과를 scratch 가 있다고 인식하는 경우가 있어

scrath 를 제거 했고, 과일 중 바나나도 이용하려고 했으나 rotted 와, browning 둘의 결합 부위 인식이 애매하게 되는 경우가 발생해 이 역시 제거하게 되었다.

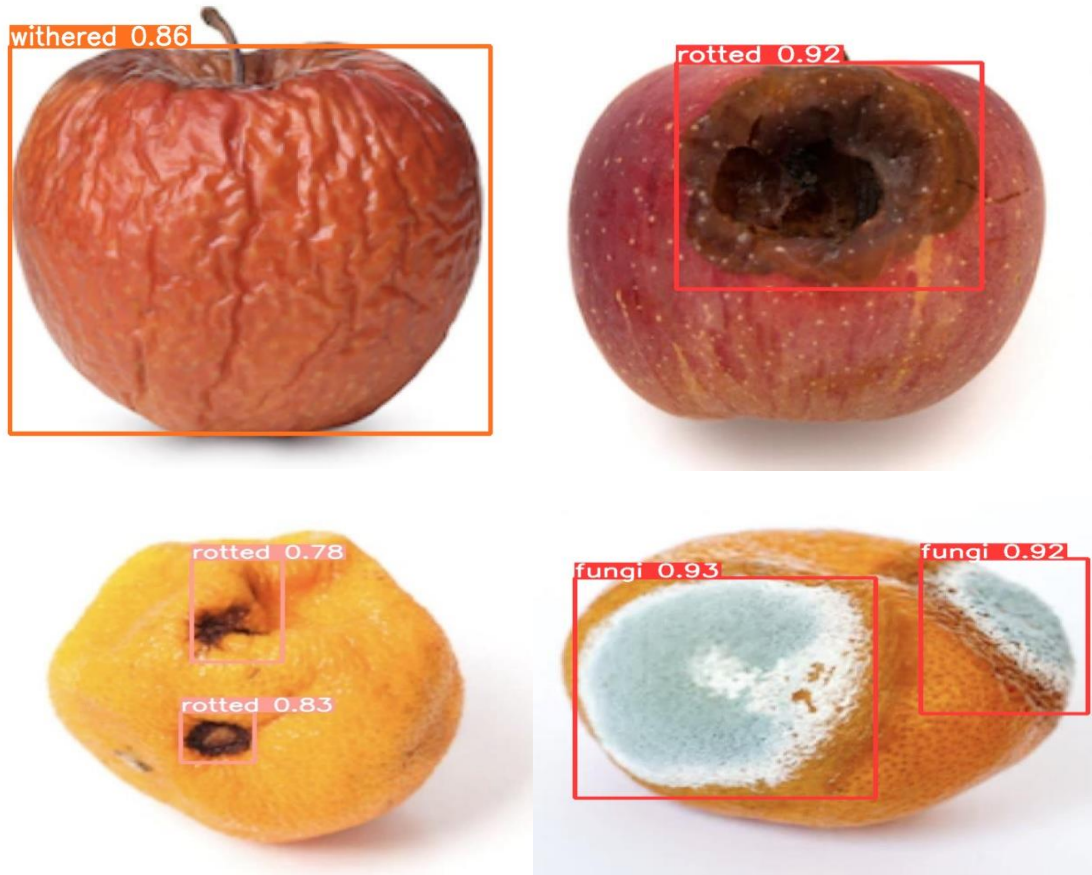


그림-12. Object-Detection 결과

위 그림에서는 apple과 orange에서 object-detection을 한 결과이다. rotted와 fungi, withered를 잘 인식한 것을 볼 수 있다. 하지만 왼쪽 밑의 오렌지에서 rotted box 범위가 조금 큰 것을 볼 수 있다.

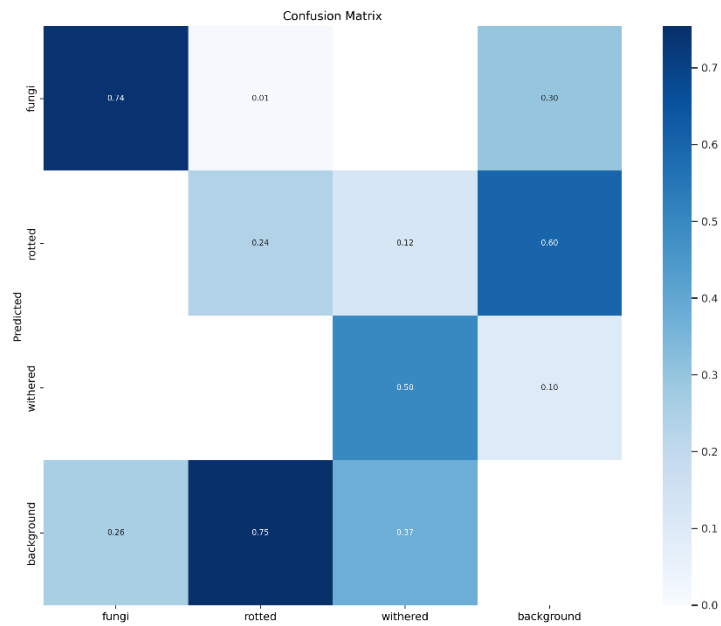


그림-13. Confusion matrix

위 그림을 보면 x축이 주어진 결함 부위이고, y축이 예측 결과 값이다 예를 들어, x 값이 fungi일 경우를 보면 y 값으로 fungi만 나오므로 인식이 잘되고 있는 것이다. x 값이 rotted일 경우에는 y값이 rotted뿐만 아니라 fungi도 같이 나온다. 이를 보면 잘못 인식이 될 수도 있는 것을 알 수 있다.

위처럼 잘못 인식이 되는 이유를 알아보자면 결함 부위 labeling 을 많이 한 순서는 fungi-rotted-withered 순서이다. rotted 같은 경우에는 학습시킨 데이터 수에 비해서는 성능이 좋지 않다. 그 이유는 rotted 는 rotted 가 발생했을 때 fungi 가 같이 겹쳐 있는 경우가 종종 있었고, withered 가 발생했을 때 rotted 가 겹쳐 있는 경우가 종종 있었다. 겹쳐 있는 data 들을 많이 모으지 않은 상태에다가 labeling 이 조금씩 겹쳐진 상태까지 되어 학습이 제대로 되지 않아서 그림-13 과 같이 rotted 인데 fungi, rotted 를 예측하고, withered 인데 rotted, withered 를 예측하는 결과가 나왔다.

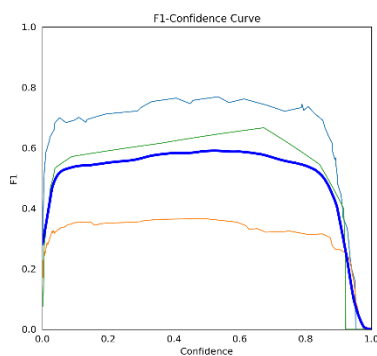


그림-14. F1-Confidence Curve

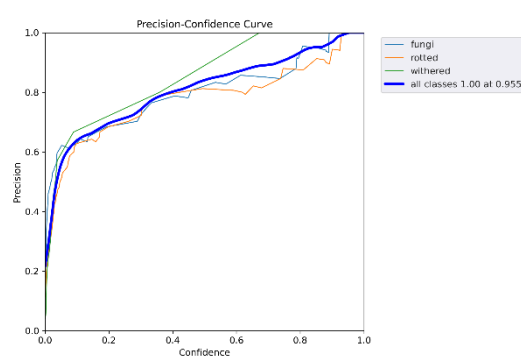


그림-15. Precision-Confidence Curve

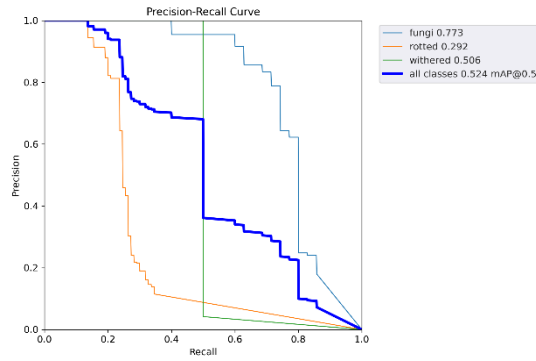


그림-16. Precision-Recall Curve

그림-15에서 Precision-Confidence Curve란 결함을 예측했을 때 그게 실제로 정답일 경우를 나타내는데 withered, fungi, rotted 순서 대로이다.

그림-16의 Precision-Recall Curve는 x값은 실제 결함 부위를 예측해서 맞힌 확률이고, y 값들은 예측한 결함들 중 실제로 맞힌 확률이다. 즉, x 값과 y 값이 모두 높은 것이 좋은 것이다. 여기 그래프를 보면 x 값이 작을 때, y 값이 큰 rotted는 결함부위는 적게 예측하나 정확도가 높은 것이고, x 값이 크면서 y 값이 작은 withered는 결함부위는 많이 예측하나 정확도가 좀 떨어지는 것을 보여준다. 이 둘과 다르게 x 값도 조금 더 크고 y 값도 조금 더 큰 fungi는 결함부위를 rotted보단 많이 예측하면서 정확도가 좋지만 withered와 비교하면 적게 예측하지만 정확도는 조금 더 높다고 볼 수 있다.

그림-14를 마지막으로 설명하는 이유는 그림-14에서 그림-15와 그림-16을 종합적으로 보고 평가 지표로 잘 쓰이는 F1-score를 나타냈기 때문이다. F1-score를 사용하는 이유는 임곗값을 이용해 정밀도, 재현율의 값들은 극단적인 수치 조작이 가능하다. 그래서 정밀도와 재현율 어느 한쪽으로 치우치지 않을 때 F1-score는 높은 값을 가진다. F1-score를 기준으로 성능을 순서대로 나타낸다면 fungi, withered, rotted 순서이다.

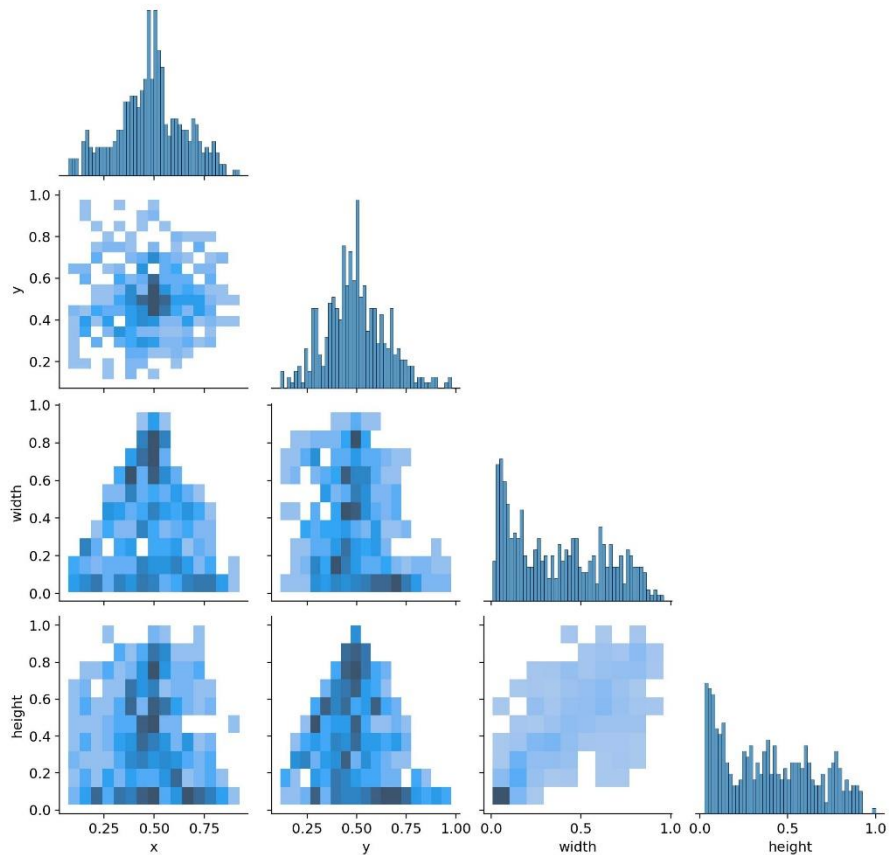


그림-17. labels_correlogram

그림-17 은 labels_correlogram 을 나타낸 것이다. Correlogram 이란 상관성(correlation)에 대한 통계(statistics)를 이미지화 한 것을 총칭한다. 시계열로 한정하는 경우 x 축에 시차(lag), y 축에 상관계수(Width, Height)를 놓은 뒤 plotting 한 것을 의미하게 된다. 즉, autocorrelation plot 과 같다. 단점은 수치적인 값들은 정확하지만, 약간의 데이터의 특성들에 대한 전반적인 사항들을 거의 나타내지 못한다는 것이 특징이다.

위 그림-16 을 보게 되면 object 들을 box 처리할 때 box 의 중심 좌표 x, y 와 width, height 값이 어떻게 분포되어 있는지를 나타내고 있다. object 들이 이미지 상의 어디에 위치하고 크기는 어떻게 되는지 알 수 있다 전반적으로 box 처리 된 곳에 Dataset 의 object 들이 잘 분포 되어있는 것을 확인할 수 있다.

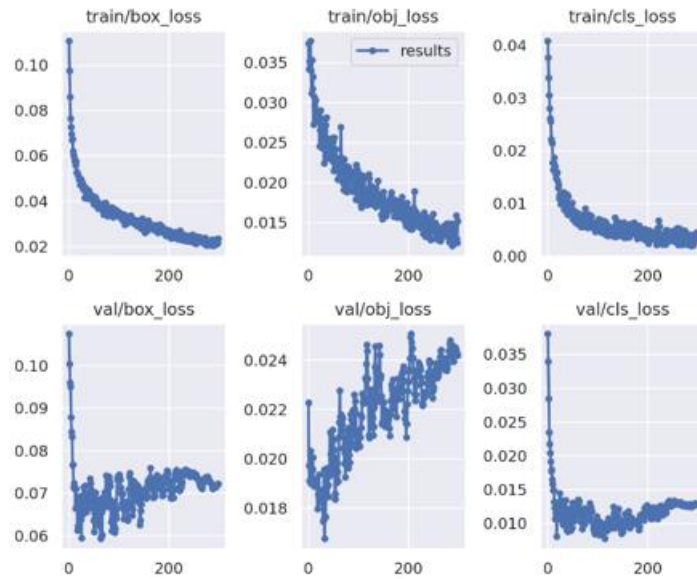


그림-18. 각 항목 loss

box_loss는 앞서 설정한 defect 부위를 얼마나 tight하게 box처리하는지에 대한 loss를 나타내고, obj_loss는 결함부위 예측에 대한 판별에 대한 loss를 나타낸다. cls_loss는 만약 결함이라면 어떤 종류의 결함인지 판별에 대한 loss를 나타낸다.

위 그림에서 x축은 epoch를 나타낸다 즉, 학습량이다. 그래서 x값이 증가할수록 train loss에 대해서는 계속 줄어드는 것이 보인다. 하지만 validation obj_loss에 대해서는 감소하다가 증가하는 지점이 overfitting이 발생했다. 본 실험에서 Apple, Orange, Strawberry를 모두 넣고 학습시켰기 때문에 train하면서 test하는 validation에서 loss가 증가하는 것으로 추정된다. 비교 사진으로 그림-18을 보면 Orange 하나만을 돌렸을 때 loss를 보면 x축의 epoch가 늘어날수록 obj_loss가 감소하는 것을 알 수 있다.

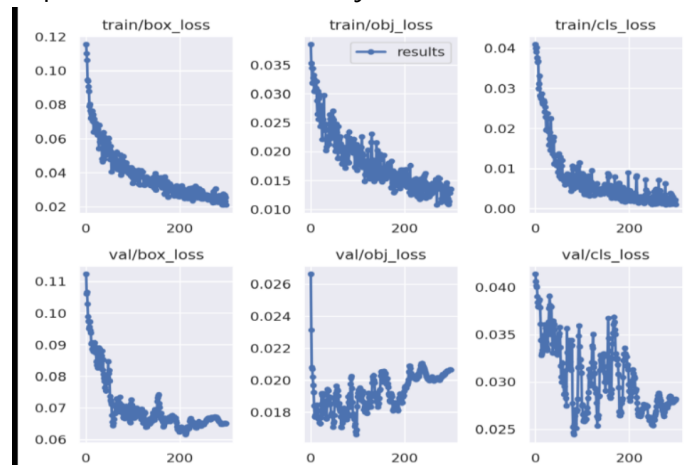


그림-19. Orange만 train하였을 때 loss

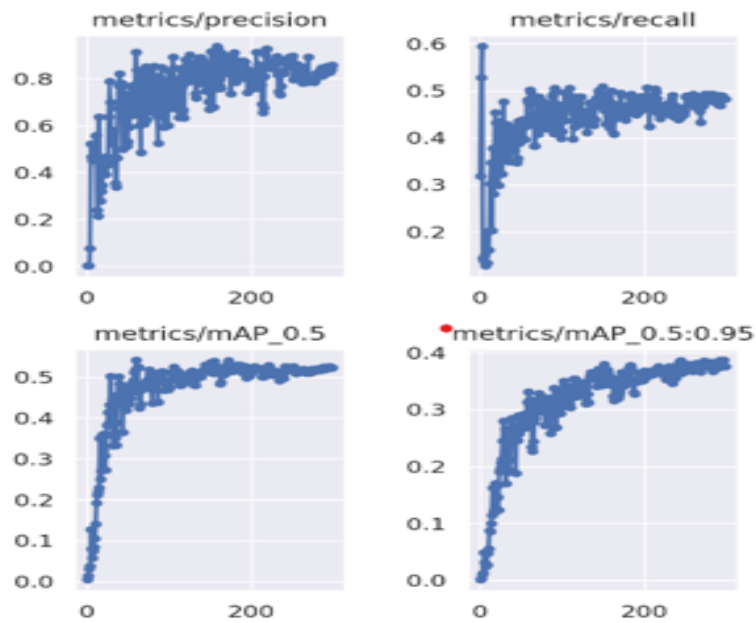
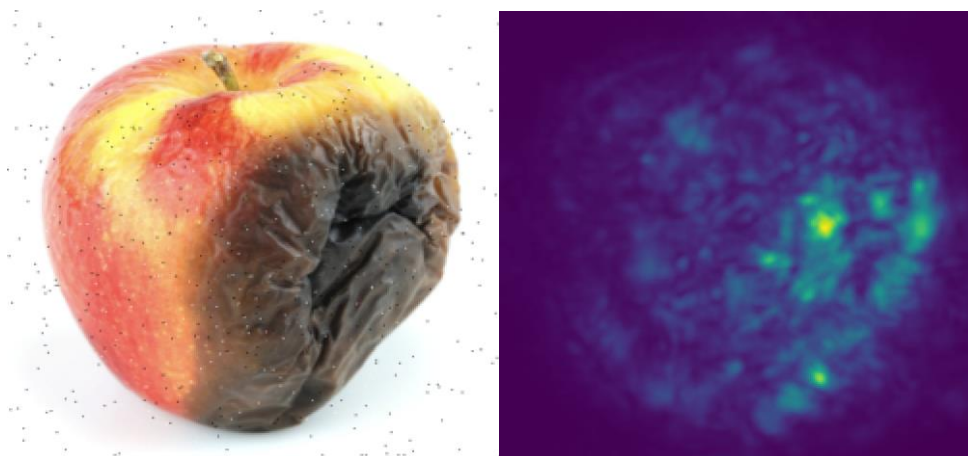


그림-20. mAP

위 그림에서 mAP는 모든 class의 평균 precision에 대해 평균값을 낸 것이다. x축은 epoch로 학습량을 증가시킬수록 성능이 개선되기 때문에 위 그래프처럼 precision과 recall이 점점 좋아지는 것을 볼 수 있다. 하지만 어느 일정 epoch에 도달하면 수렴하는 것을 볼 수 있다. 이는 epoch 수를 늘리는 것만으로는 정확도를 개선하는 데 한계가 있음을 의미한다.

4.2. DifferNet Anomaly-Detection 결과 분석 및 평가



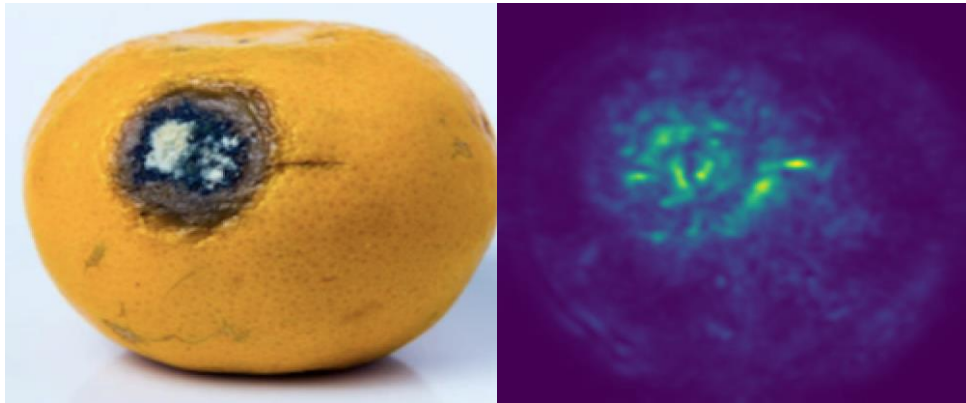


그림-21 .Gradient map 결과

그림-19과 같이 Anomaly-Detection으로 gradient-map을 추가로 진행했다. 왼쪽 열의 사과나 오렌지의 결함 부위가 오른쪽 map에서 밝게 빛나면서 잘 나타나는 것을 볼 수 있다. 이 Gradient-map 또한 위와 같은 data들을 학습을 시켜 성능이 많이 좋지 않아, 모든 데이터에서 위와 같은 제대로 된 결과를 얻는 것은 힘들었다. 그 이유는 학습 data의 부족이다. 적은 data의 양으로도 여러 번 학습을 시켰을 때 성능이 조금 개선되긴 하지만, 앞서 봤던 것처럼 특정 한계를 넘어가면 더 이상 성능이 개선되지 않았다. 그래서 데이터 수를 더 늘려 보긴 했지만 그 수가 적어 정확도 개선은 느끼지 못했다.

5. 결론 및 향후 연구 방향

불량 검출 시스템을 만드는 과정에서 불량인 과일들의 데이터들을 모아 직접 labeling을 진행했는데 많은 데이터를 찾기 위해 많은 시간을 투자했지만 몇 가지 특정 결함 부위들만 반복되는 데이터들 밖에 구하지 못했다. 그래서 정확도를 최대한 늘리기 위해 roboflow라는 사이트를 이용해 labeling을 꼼꼼하게 진행하고 yolov5와 DifferNet을 이용해 학습을 시켰다. 그리고 flask를 이용해 웹을 연동해 결과를 확인하니 fungi는 잘 인식하지만 rotted는 fungi, rotted로 인식하거나 결함 부위를 인식 못 하고, withered는 rotted, withered를 인식하면서 rotted와 같이 결함부위를 인식 못하는 경우가 존재했다. rotted와 withered는 성능이 좋지 못한 것을 보았다.

비록 본 과제에서는 시간문제로 많은 데이터들의 결함부위를 학습시키지 못하여 정확한 인식을 하지 못하는 경우가 있었지만 더 많은 데이터를 구해 rotted와 withered를 학습시키면 더 좋은 성능을 보일 것이다. 추후 추가 데이터를 구해 학습을 시도할 예정이다. 그리고 더 많은 epoch를 주고 학습을 시켜볼 것이다.

6. 구성원별 역할 및 개발 일정

6.1. 구성원별 역할

이름	역할
최강현	딥러닝 기반 모델 구성 모델 최적화
김승연	학습 데이터 수집 및 전처리 웹 연동 및 개발
이동훈	학습 데이터 수집 및 전처리 모델 학습 및 검증
공통	부족한 지식 습득 보고서 작성 발표 및 시연 준비

6.2. 개발일정

5월			6월				7월				8월				9월				
14	21	28	4	11	18	25	2	9	16	23	30	6	13	20	27	3	10	17	24
머신러닝 관련 기술 공부																			
			기본 모델 작성 ,데이터 수집																
							모델 최적화												
									중간 보고서 작성										
									웹,앱 연동										
																최종 테스트			
																	최종 보고서 작성 발표 준비		

7. 참고 문헌

- [1] Thuan, Do. "Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm." (2021).
- [2] **Marco Rudolph, Bastian Wandt, Bodo Rosenhahn**; Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2021, pp. 1907-1916
- [3] flow model , Available: <https://lilianweng.github.io/posts/2018-10-13-flow-models/>
- [4] 10-fruits- detection, Available: <https://github.com/DevTimlas/10-fruits-detection>
- [5] Roboflow image labeling, Available: <https://roboflow.com/annotate>
- [6] Same Same but Different, Available: <https://github.com/AndSonder/Flow-based-Anomaly-Detection-Model/tree/main/differnet>
- [7] flask-object-detection, Available: <https://github.com/AIZOOTech/flask-object-detection>
- [8] Yolov5, Available: <https://github.com/ultralytics/yolov5>