연합전공 경진대회 무인판매대 시스템

36팀 JYW
전기정보공학부 장영우, 김태준

Reference

1. Darknet - AlexeyAB
   https://github.com/AlexeyAB/darknet
2. Image Augmentation -
   https://imgaug.readthedocs.io/en/latest/index.html
3. Pruning - TNTWEN
   https://github.com/TNTWEN/Pruned-OpenVINO-YOLO

목차

1. 제공된 이미지의 파악 및 개선점
2. 다양한 이미지의 제작
   a. 조명에 대한 이미지
   b. 밝기에 대한 이미지
   c. 배경이미지 추가
   d. 물체들을 겹친 이미지
3. 다양한 모델을 사용한 결과 비교
4. Pruning을 활용한 모델 학습
5. 기본 YOLO 모델과의 비교
6. 결론

1. 제공된 이미지의 파악 및 개선점

제공된 이미지와 bounding box 정보를 가진 text

조명의 효과가 포함된 이미지에 대한 결과

['37 0.640625 0.7197916666666667 0.459375 0.5604166666666667\n', '33 0.384375 0.740625 0.084375 0.10208333333333333\n', '41 0.26015625 0.5510416666666667 0.2265625 0.5270833333333333\n', '40 0.13984375 0.4083333333333333 0.0546875 0.22916666666666666\n', '43 0.15078125 0.5875 0.0828125 0.17083333333333334']

위 이미지처럼 제공된 이미지 중 몇몇은 일부 물체의 bounding box 정보를 포함하지 않고 있어 인식률에 좋지 않은 영향을 미치는 것이 확인되었습니다.

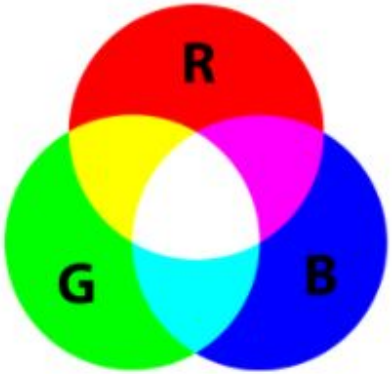
제공된 이미지로만 학습시킨 모델은 조명의 효과가 포함된 이미지를 잘 인식하지 못하는 것으로 확인되었습니다.

⇒추가적인 이미지 제작의 필요성 (조명의 효과 & 모든 물체에 대한 bounding box 정보가 포함)

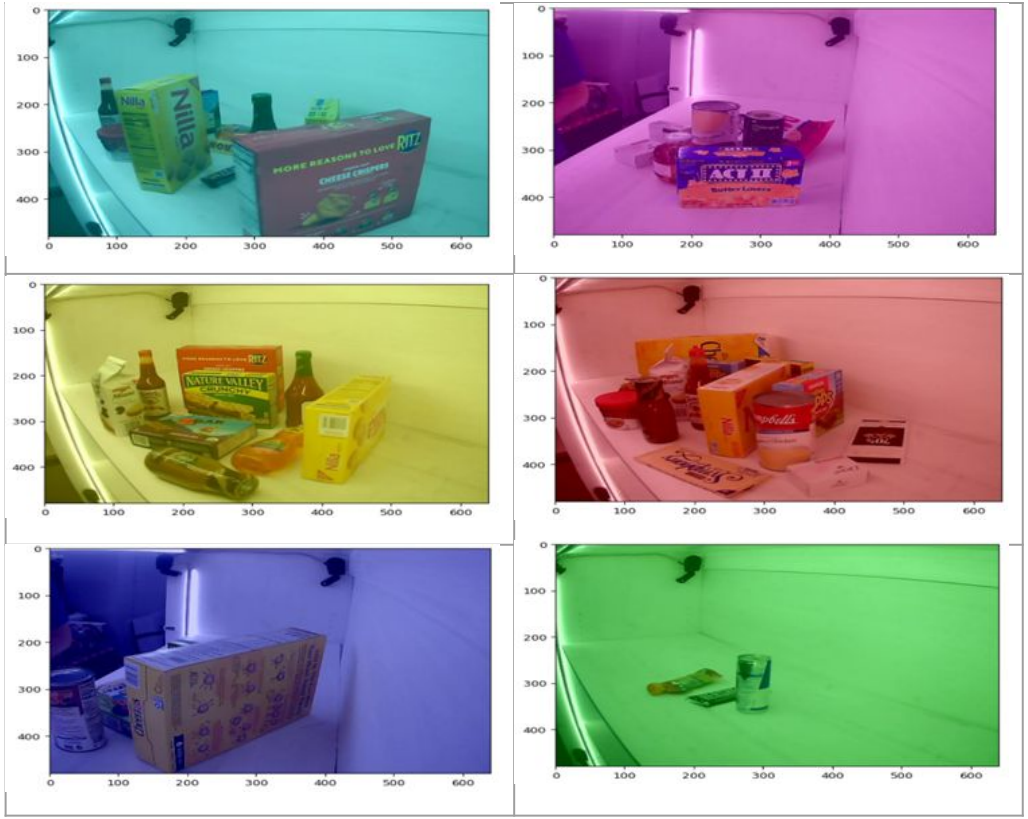
2. 다양한 이미지의 제작 - a. 조명에 대한 이미지

다양한 색깔의 조명 필요

⇒Imgaug 라이브러리의Add 함수 활용,
R, G, B, R+G, R+B, G+B의 조명 효과를 더함




```
from imgaug import augmenters as iaa

hue1=iaa.Sequential([
    iaa.WithChannels([0], iaa.Add(-50)),
    iaa.WithChannels([1], iaa.Add(50)),
    iaa.WithChannels([2], iaa.Add(50))
])
hue2=iaa.Sequential([
    iaa.WithChannels([0], iaa.Add(50)),
    iaa.WithChannels([1], iaa.Add(-50)),
    iaa.WithChannels([2], iaa.Add(50))
])
hue3=iaa.Sequential([
    iaa.WithChannels([0], iaa.Add(50)),
    iaa.WithChannels([1], iaa.Add(50)),
    iaa.WithChannels([2], iaa.Add(-50))
])
hue4=iaa.Sequential([
    iaa.WithChannels([0], iaa.Add(50)),
    iaa.WithChannels([1], iaa.Add(-50)),
    iaa.WithChannels([2], iaa.Add(-50))
])
hue5=iaa.Sequential([
    iaa.WithChannels([0], iaa.Add(-50)),
    iaa.WithChannels([1], iaa.Add(-50)),
    iaa.WithChannels([2], iaa.Add(50))
])
hue6=iaa.Sequential([
    iaa.WithChannels([0], iaa.Add(-50)),
    iaa.WithChannels([1], iaa.Add(50)),
    iaa.WithChannels([2], iaa.Add(-50))
])
```

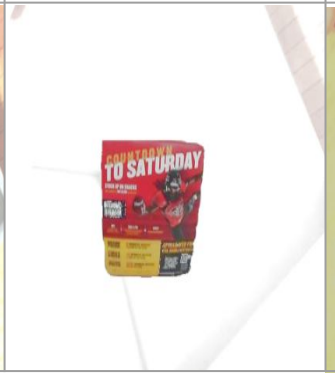
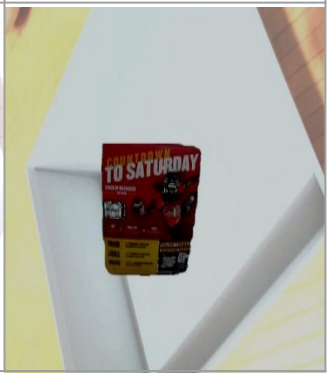


2. 다양한 이미지의 제작 - b. 밝기에 대한 이미지

제작한 이미지들의 밝기를 바꾼 이미지 필요

⇒Imgaug 라이브러리의 AddToBrightness 함수 활용,
적정 범위 내에서 이미지들의 밝기를 바꿈

```
from imgaug import augmenters as iaa
brightness=iaa.AddToBrightness((-50,75))
images_aug_brightness=brightness(images=images)
```

기본 이미지	밝은 버전	어두운 버전
		

※ 이외에도 imgaug라이브러리에 다양한 함수들이 있지만 (ex) 초점변경, 빈칸생성 등) 생성되는 이미지 개수 대비 모델의 인식률 향상에는 큰 영향을 미치지 않을 것이라 판단하고 사용하지 않았습니다.

2. 다양 이미지의 제작 - c. 배경이미지 추가

배경의 부재를 해소하기 위한 배경이미지 필요

⇒직접 일상생활의 선반이나 박스 사진을 다양한 각도로 찍어 배경이미지에 추가 (직접 찍은 사진 34장 * 4단계로 밝기를 바꾼 버전 = 136장)
⇒이를 통해 배경에 따른 인식률의 오차를 줄이고자 함

```
from imgaug import augmenters as iaa

changebrightness1 = iaa.AddToBrightness(-50)
changebrightness2 = iaa.AddToBrightness(50)
changebrightness3 = iaa.AddToBrightness(100)

images_aug1 = changebrightness1(images=images)
images_aug2 = changebrightness2(images=images)
images_aug3 = changebrightness3(images=images)
```

2. 다양 이미지의 제작 - d. 물체를 겹친 이미지

제공된 이미지에서 물체들의 bounding box 정보 부족

⇒직접 단품의 물체들을 겹친 이미지를 제작하면서 이들의 bounding box 정보들도 다 저장할 수 있도록 함

초기 : 물체들을 랜덤으로 합성⇒작은 물건이 큰 물건에 가려지는 경우 발생

수정 : 넓이를 기준으로 물체를 정렬 후 합성⇒가려지는 정도가 랜덤적이며 특히 많이 가려지는 경우에는 오히려 인식률의 하락을 초래

초기	수정
	

2. 다양 이미지의 제작 - d. 물체를 겹친 이미지

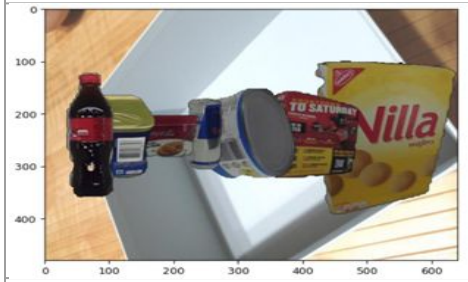
제공된 이미지에서 물체들의 bounding box 정보 부족

⇒직접 단품의 물체들을 겹친 이미지를 제작하면서 이들의 bounding box 정보들도 다 저장할 수 있도록 함

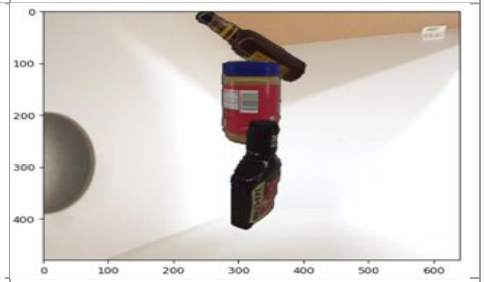
최종 : bounding box가 이미지를 벗어나지 않는 선에서 가로, 세로, 대각선 방향으로 물체들을 이동시키면서 합성 + 2.c에서의 배경도 추가
(이때 겹치는 비율은 실험적으로 정하였으며 0.5이상은 오히려 인식률을 하락시켜 0.1~0.4로 선정하였음)

```
# 첫 번째 이미지는 왼쪽(또는 오른쪽)에 붙이고 y좌표는 0.5로 해서 정확히 중간에 오게함
first_img_start_point = 0.01
target_x = first_img_start_point + bbox_list[0].w / 2
target_y = 0.5                          # target_y는 일단 0.5로 통일. 이후에 랜덤으로 조금씩 바꾸는 것도 가능하다

# 첫 번째 이미지의 위치를 옮김과 동시에 이미지의 바운딩 박스 정보도 업데이트한다.
images[0] = move(images[0], bbox_list[0], target_x, target_y)

# 첫 번째 이미지가 base가 되어 나머지 이미지들을 겹친다
base_img = images[0]

# 두 번째 ~ 마지막 번째 이미지 까지는 그 이전에 있는 물체와 겹치는 정도(x축 기준)를 기준으로 겹치게 위치시킨다.

overlap = 0.5 # 몇 퍼센트 겹치게 할건지
next_img_start_point = bbox_list[0].x + bbox_list[0].w * (0.5 - overlap) # overlap 비율만큼 겹치도록 다음 이미지가 시작되는 지점을 계산.

for i in range(1,N):
    target_x = next_img_start_point + bbox_list[i].w / 2
    # target_y = 0.5로 통일이므로 따로 업데이트는 하지 않는다.
    if target_x + bbox_list[i].w / 2 > 1: # 만약 바운딩 박스가 이미지 바깥으로 나가게 된다면 다음 이미지를 시도해본다.
        continue

    images[i] = move(images[i], bbox_list[i], target_x, target_y) # 이미지를 target_x, target_y 좌표로 옮김, 동시에 bbox 좌표도 업데이트
    base_img = addimage(base_img, images[i])

    next_img_start_point = bbox_list[i].x + bbox_list[i].w * (0.5 - overlap)
```

가로로 합성한 예시

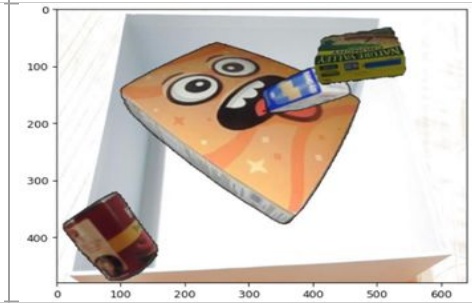


세로로 합성한 예시



대각선(왼쪽 위)로 합성

대각선(오른쪽 위)로 합성



3. 다양한 모델을 사용한 결과 비교

이미지를 활용한 인식률 상승의 한계에 도달

⇒모델을 바꿔가면서 인식률에 큰 변화는 주지 않으면서 inference speed를 빠르게 할 수 있는 모델을 찾고자 함

	Processing Time [sec]	mAP [%]
YOLO v3	8	98.9
YOLO v4	9	99.5

Experimented on Tesla V100 with FP32 precision

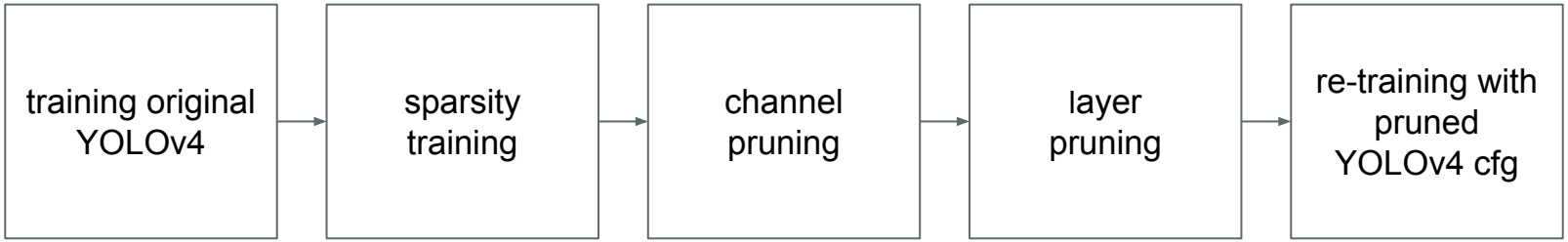
<Test Set로 확인해본 Processing Time과 mAP>

결론

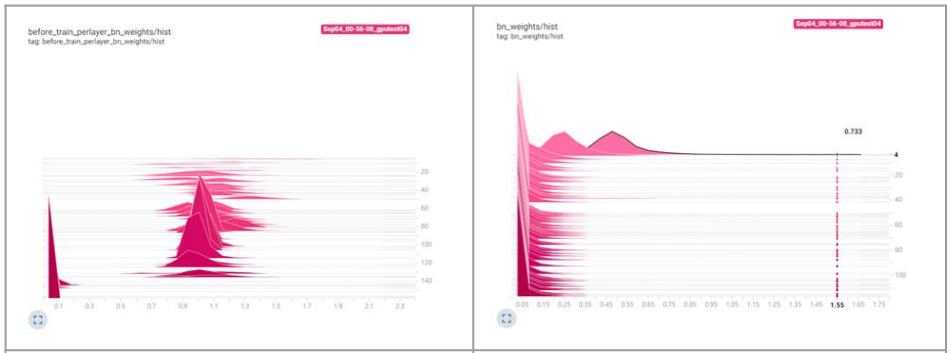
모델 버전의 변화만으로는 큰 변화를 얻지 못했으므로 다른 방안을 찾아보고자 함

1. pruning 기법을 활용하여 inference speed를 빠르게 할 수 있는지를 확인해보고자 함
2. 다양한 tiny 모델을 활용하여 작은 모델이 mAP의 하락없이 inference speed를 빠르게 할 수 있는지 확인해보고자 함

4. Pruning을 활용한 모델 학습



<YOLO v4 pruning workflow>



before training | after training

- 12000장의 트레이닝셋 표본을 이용해 100epoch만큼 training

sparsity training을 통해 sparseness가 확보된 weight파일의 히스토그램

4. Pruning을 활용한 모델 학습
- channel and layer pruning

glob_percent	mAP(%)	# of parameters	Inference(ms)
before	0.871513	64255401	0.0183
0.79	0.865434	3428269	0.0162
0.8	**0.865779**	**3254324**	**0.0166**
0.81	0.864373	3071714	0.0162
0.82	0.859185	2892926	0.0160
0.83	0.845168	2734300	0.0163
0.84	0.784893	2542095	0.0161

global percent = 0.8일때 inference time대비 가장 적은 mAP loss를 얻음

shortcut	mAP(%)	# of parameters	inference(ms)
before	0.871513	64255401	0.0183
11	0.869626	3228658	0.0139
12	**0.873946**	**3219116**	**0.0137**
13	0.857189	3158606	0.0138
14	0.830592	3110916	0.0133

shortcut = 12일때 inference time대비 가장 적은 mAP loss를 얻음

4. Pruning을 활용한 모델 학습 - result

pruning 하기 전 YOLO v4와 비교

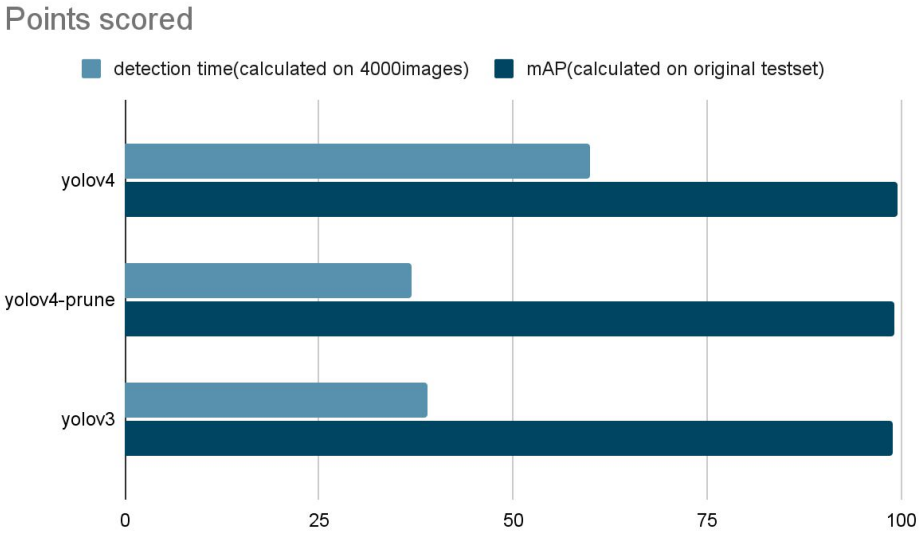
	YOLOv4	YOLOv4-prune
mAP(%)	99.53	99.13
Inference time(sec)	9	5
Weight file size(KB)	251256	12634
Total FLOPS(billion)	59.992	17.909
Num of layers	162	126

Experimented on Tesla V100 with FP32 precision

결론 => layer의 수를 줄임으로써 inference time을 크게 줄일 수 있음

동시에 YOLOv4가 워낙 큰 모델이기 때문에 pruning을 통해 layer의 수를 획기적으로 줄이는데는 한계가 있음을 인식함

5. 기본 YOLO모델과의 비교



다른 모델들과 detection time 비교

162 layers

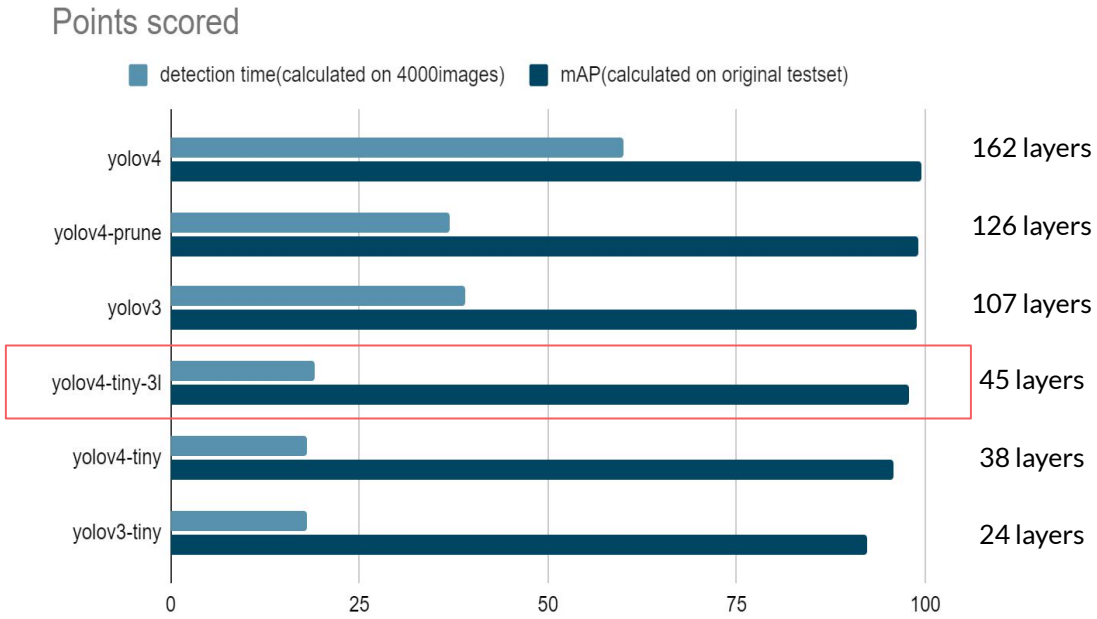
126 layers

107 layers

=> layer의 수가 inference speed에 큰 영향을 끼침

큰 모델로 부터 pruning을 통해 좀 더 작은 모델을 만듦
=> 작은 모델로 부터 layer추가를 통해 정확도의 향상을 꾀함

5. 기본 YOLO모델과의 비교



기존 YOLOv4-tiny가 yolo output layer가 2개여서 다양한 크기의 물체를 인식하는데 한계가 있었다면,
YOLOv4-tiny-3l은 3개의 yolo output layer를 통해 다양한 크기의 물체에 대해서 더 잘 인식할 수 있다는 장점이 있음

	YOLOv4-prune	YOLOv4-tiny-3l
mAP(%)	99.13	97.76
detection time on 4000 images(sec)	38	19

1.37 %p mAP loss, **50% less detection time!**

5. 기본 YOLO모델과의 비교

	YOLOv4-prune	**YOLOv4-tiny-3l**	YOLOv4-tiny-3l-prune
mAP(%)	99.13	**97.76**	96.84
detection time on 4000 images(sec)	38	**19**	18

YOLOv4-tiny-3l is optimal for both mAP & detection time

6. 결론

결론

YOLOv4 네트워크에서 pruning을 한 네트워크보다, YOLOv4-tiny에서 layer를 더 추가한 YOLOv4-tiny-3l이 mAP loss대비 획기적인 inference speed의 향상을 보여줬기 때문에 최종 모델로 YOLOv4-tiny-3l을 선택

	YOLOv3	YOLOv4-tiny-3l
mAP(%)	98.9	97.76
det time	39	19

<기본 네트워크(YOLOv3)와의 비교>

최종 모델 : YOLOv4-tiny-3l

최종 mAP : **97.76 %**

기본 모델보다 처리속도 약 **105.26% 향상!**

제언

darknet에서 cuDNN_half=1로 설정함으로써 fp32 precision대신 fp32+16 mixed precision을 사용하여 학습 및 추론을 할 수 있는데, 본 프로젝트에서는 이에 대한 뚜렷한 개선을 확인할 수 없었지만, fp16 또는 int8로 추론을 할 수 있게 한다면 더 빠른 추론이 가능할 것으로 생각됨.

최종 weight 파일 위치
/home/ai_competition/Final_submission/darknet/backup/yolov4-tiny-3l.weights

감사합니다.

36팀 (JYW)

전기정보공학부 장영우, 김태준