

과제번호

4.22280

2021학년도 학부생연구프로그램(UGRP)

최종보고서

과제명	영상인식을 이용한 레시피 추천에 관한 연구				
연구기간	2021. 5. 21. ~ 2022. 1. 16.				
연구비	162,600원				
지도교수	성명	박재식	교수님	학과	컴퓨터공학과
참여자 인적사항	구분	학과	성명	학번	연락처
	연구 책임자	기계공학과	임승민	20200843	010-5131-3316
	공동 연구원	컴퓨터공학과	김대연	20200217	010-2507-7308
	공동 연구원	신소재공학과	김재영	20200484	010-7739-1402
	공동 연구원	전기전자공학과	김태환	20200850	010-4082-0047
	공동 연구원	기계공학과	권성건	20200258	010-9442-0377
	공동 연구원	물리학과	박기범	20200383	010-9442-0377
	공동 연구원	물리학과	신동환	20200337	010-5878-3008
2021학년도 학부생연구프로그램 최종보고서를 아래와 같이 제출합니다.					
2022년 01월 17일					
연구책임자 : 임승민 (인) 연구지도교수 : 박재식 (인)					
포항공과대학교 교육혁신센터장 귀하					

※ 팀장(연구책임자)과 지도교수님 모두 자필 또는 전자 서명을 넣어주세요.

2021학년도 학부생연구프로그램(UGRP)

연구비 집행 결산보고

연구책임자(학생)	임승민	소속/학번	기계공학과 / 20200843
과제명	영상인식을 이용한 레시피 추천에 관한 연구		

■ 연구비 집행표

세부 비목	당초 예산	최종 집행 예산
연구장비 및 시설비	50,000	0
시약·재료비 및 전산 처리	0	0
시작품 제작비	859,280	135,600
수용비 및 수수료	0	0
기술정보활동비	55,800	27,000
국내여비	0	0
합계(원)	965,080	162,600

☞ 연구비 예산에는 당초 연구비로 배정받은 금액을, 집행한 연구비는 중 최종 집행한 연구비를 기입함.

2021학년도 학부생 연구프로그램(UGRP)

최종보고서

연구책임자(학생)	임승민	소속/학번	기계공학과/20200843
과제명	영상인식을 이용한 레시피 추천에 관한 연구		

1. 연구 목적

1.1. 연구 동기

1인 가구의 수가 증가함에 따라 혼자 식사하는 사람의 수도 증가하고 있다. 그러나 1인 가구 중 많은 사람들이 균형잡힌 식사를 하는 데에 어려움을 겪고 있다. 1인 가구의 경우, 일반적으로 배달 음식이나 직접 요리하여 식사를 해결한다. 그러나 배달 음식은 비용적으로나 영양학적으로나 좋은 선택지가 아니기에 최근에는 직접 요리하여 식사를 해결하는 1인 가구의 수가 증가하고 있다. 하지만 요리초보자들은 재료를 효율적으로 사용하는 것을 어려워하며 남은 식재료를 처리하지 못하고 결국은 버리게 된다.

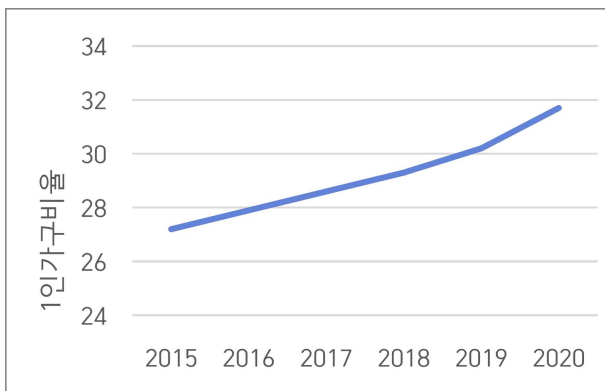


그림 1. 1인 가구 비율(KOSIS, 인구총조사)

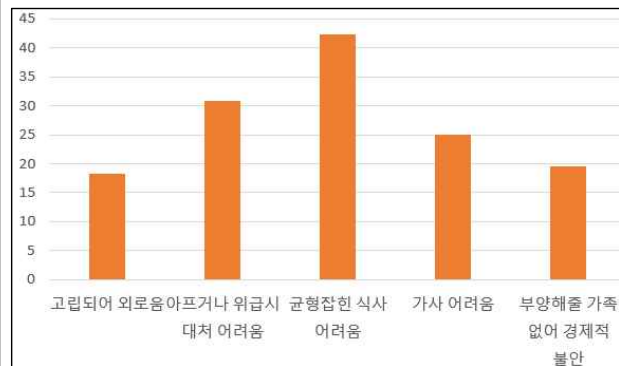


그림 2. 1인 가구의 어려움(여성가족부)

1.2. 연구 목표

이러한 문제를 저희는 영상인식 기술을 도입하여 해결하고자 한다. 기존의 레시피 추천 서비스와 달리 재료들을 사용자가 직접 입력할 필요 없이 남은 재료를 촬영하면 한 번에 재료 현황을 파악하고 사용자의 취향과 난이도에 알맞은 레시피를 추천하는 것이 본 연구의 목표이다.

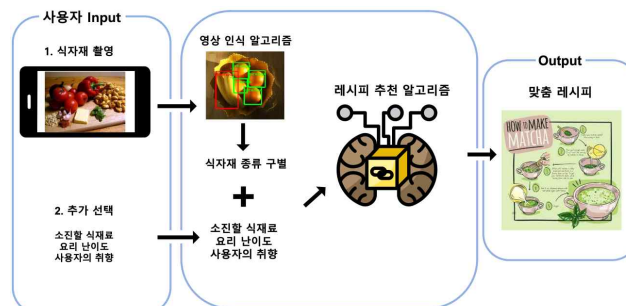


그림 3. 연구 목표 인포그래픽

2. 연구내용 및 진행

2.1. 배경 지식

2.1.1. YOLO

사람은 어떤 이미지를 봤을 때, 이미지 내부에 있는 object들의 detail(object가 무엇인지, 어디에 있는지, 어떤 관계에 있는지)을 한눈에 파악할 수 있다. 하지만 R-CNN과 같은 detection system들은 복잡한 처리 과정으로 인해 Human visual system을 모방하기에는 속도나 최적화 측면에서 어려움이 있다.

YOLO(You Only Look Once)는 학습한 물체의 종류와 위치를 실시간으로 파악할 수 있는 Real-Time Object Detection 모델이다. YOLO는 이미지 내의 bounding box와 class probability를 single regression problem으로 간주하여 이미지를 한 번 보는 것으로 object의 종류와 위치를 추측한다. Yolo는 간단한 처리 과정으로 속도가 매우 빠르며 높은 정확도를 보인다.

2.1.2. YOLO v3

2.1.2.1. Feature Extractor

Model은 convolution layer가 53개인 darknet-53을 사용하였다. Yolo v2와 달리 residual connection을 이용하여 layer를 더 deep하게 쌓았다.

Type	Filters	Size	Output
Convolutional	32	3 × 3	256 × 256
Convolutional	64	3 × 3 / 2	128 × 128
1x	Convolutional	32	1 × 1
	Convolutional	64	3 × 3
	Residual		128 × 128
	Convolutional	128	3 × 3 / 2
			64 × 64
2x	Convolutional	64	1 × 1
	Convolutional	128	3 × 3
	Residual		64 × 64
	Convolutional	256	3 × 3 / 2
			32 × 32
8x	Convolutional	128	1 × 1
	Convolutional	256	3 × 3
	Residual		32 × 32
	Convolutional	512	3 × 3 / 2
			16 × 16
8x	Convolutional	256	1 × 1
	Convolutional	512	3 × 3
	Residual		16 × 16
	Convolutional	1024	3 × 3 / 2
			8 × 8
4x	Convolutional	512	1 × 1
	Convolutional	1024	3 × 3
	Residual		8 × 8
	Avgpool	Global	
	Connected	1000	
	Softmax		

Table 1. Darknet-53.



그림 4.Darknet architecture그림 5. Residual block structure

Darknet 내부의 residual block에서는 bottleneck 구조를 사용한다. 중간에 shape이 channel의 반으로 줄었다가 다시 원래대로 돌아오는 bottleneck 구조가 사용된다. 먼저 416*416 크기의 이미지를 입력하여 지정한 layer에서 53*53, 26*26, 13*13 크기의 multi-scale feature map을 추출한다.

2.1.2.2. Predictions Across Scales

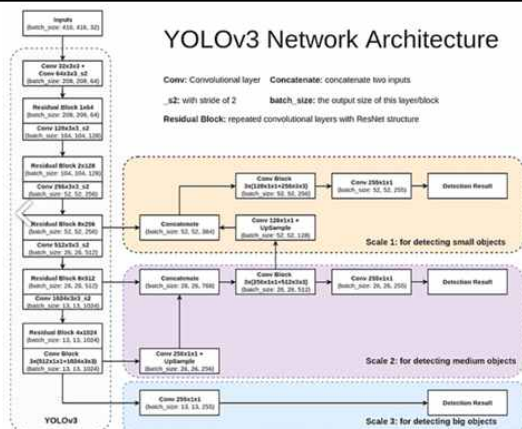


그림 6 YOLOv3 Network Architecture

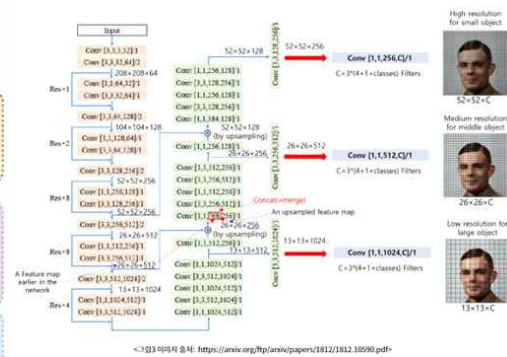


그림 7 Architecture의 세부적인 도표

앞서 얻은 3개의 서로 다른 scale을 가진 feature map을 1×1 , 3×3 convolutional layer로 구성된 FCN(Fully Convolutional Network)에 입력하여 feature pyramid를 설계한다. Feature map을 Upsampling하고 아래 level의 feature map과 concatenate해서 FCN에 입력하는 과정을 모든 level에 대해 수행하면 총 3개의 scale을 가지고 channel의 개수가 255로 동일한 feature map들을 얻을 수 있다.

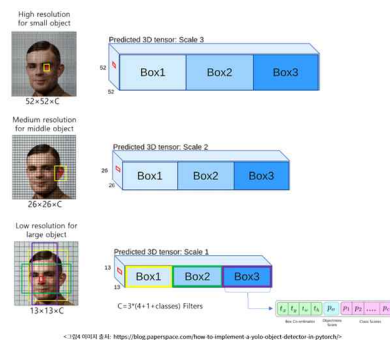


그림 8. scale에 따른 3D tensor

앞선 FCN을 통해 얻은 feature map 즉, 각 scale의 3D 텐서는 box의 좌표, objectness score, class score로 이루어진 box가 3개가 연결된 형태를 띄며 objectness score는 1 또는 0을 가지게 되고 class score는 0과 1 사이의 값을 가지게 된다. 그러므로 텐서의 형태는 박스의 길이가 255로 동일하게 된다. 텐서에 3가지의 box가 있는 이유는 3가지의 각 scale에 대해 3가지의 크기가 다른 박스를 생성하여 총 9개의 클러스터(anchor box)를 선택하게 된다. 결과적으로 bounding box prediction에서 9개의 anchor에 대해 각각의 output과 loss를 구하여 판단하게 된다.

2.1.2.3. Class Prediction

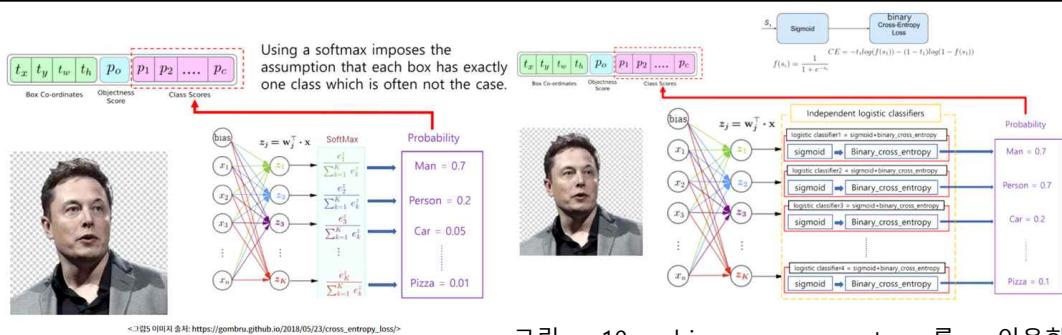


그림 9. softmax를 이용한 classification

그림 10. binary-cross entropy를 이용한 classification

앞선 tensor에서 class score를 얻는 과정이 class prediction이다. COCO dataset은 80개의 클래스를 가지고 있는데 coco dataset의 이미지들을 분류하겠다는 것은 80개의 class들을 분류하는 multi-classification task이다. 하지만 80개의 class가 모두 서로 관계가 없는 것은 아니다. Man과 Person은 독립적인 클래스가 아니라 부분집합 개념이기 때문이다. 일반적으로 classification은 softmax를 취해준 후, cross entropy를 계산하는 것이 대부분이다. 그러나 softmax는 입력 이미지에 하나의 클래스만 있을 것이라는 가정하기 때문에 logit값이 제일 높은 하나의 클래스 확률 값만 지나치게 높게 설정해준다. 그래서 multi-label을 모두 정확하게 예측하기 위해 independent logistic classifiers를 사용한다. 즉, binary-cross entropy를 사용하여 classification을 수행한다. 각각의 정보는 3개의 텐서, 9개의 box의 class scores에 해당하는 곳에 저장된다.

2.1.2.4. Bounding Box Prediction

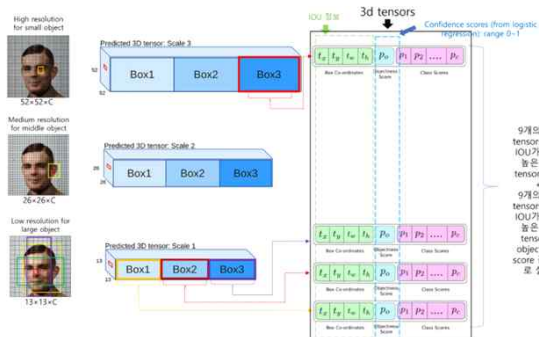


그림 11. 3D tensor의 structure

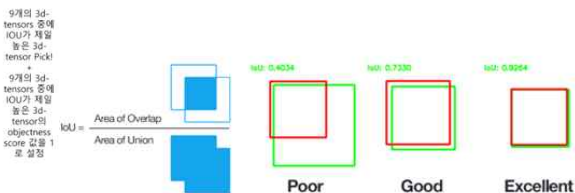


그림 12 IOU의 정의

Yolo는 클러스터링을 통해 anchor box를 생성하여 bounding box를 예측한다. 다른 detection들은 ground truth와 anchor간 IOU가 일정 수준 이상이면 positive anchor로 사용하는 반면, yolo는 object 당 단 하나의 anchor에 할당하여 positive anchor로 사용한다. 여기서 IOU는 overlap된 면적을 전체 면적으로 나눈 값으로 값이 클수록 좋다. 그 후, Ground truth(정답값)와 네트워크의 output인 prediction(예측값)을 비교하여 loss를 구하는데 object detection은 classification과 localization이 합쳐진 task이므로 classification과 물체의 위치를 예측하는 부분으로 나뉘어 학습이 진행된다. Anchor box는 pre-define된 object를 탐지하기 위한 box이며 scale별로 3가지 anchor에 대해 각각 output과 loss를 구하게 된다.

한 번 학습 시 생성되는 bounding box의 개수는 $10647 = 52 \times 52 \times 3 + 26 \times 26 \times 3 + 13 \times 13 \times 3$ 이고 그 중, IOU가 제일 높은 box의 3d-tensor만 loss function에 사용된다. IOU가 제일 높은 3d-tensor의 objectness score만 1로 설정한다.

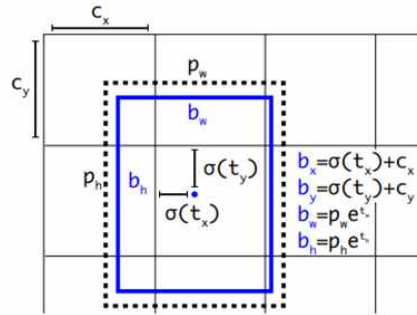


그림 13. bounding box prediction

Yolo는 box를 prediction할 때 grid의 상대적인 위치로 bounding box의 위치를 찾아낸다. x, y에 대한 prediction은 grid cell안에서 0~1의 위치를 갖는 것으로 학습을 하고 w, h는 log scale을 사용한다.

2.1.2.5. YOLO v3 loss function

$$\begin{aligned} & \lambda_{coord} \sum_i^S \sum_j^B \mathbb{I}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \\ & \lambda_{coord} \sum_i^S \sum_j^B \mathbb{I}_{ij}^{obj} [(w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2] + \\ & \sum_i^S \sum_j^B \mathbb{I}_{ij}^{obj} [- (o_i \log(\hat{o}_i) + (1 - o_i) \log(1 - \hat{o}_i))] + \\ & Mask_{ig} * \lambda_{noobj} \sum_i^S \sum_j^B \mathbb{I}_{ij}^{noobj} [- (o_i \log(\hat{o}_i) + (1 - o_i) \log(1 - \hat{o}_i))] + \\ & \sum_i^S \sum_j^B \mathbb{I}_{ij}^{obj} [- (c_i \log(\hat{c}_i) + (1 - c_i) \log(1 - \hat{c}_i))] + \end{aligned}$$

Yolo v3 loss에서 x, y, w, h 즉, bounding box offset의 loss는 sum square error를 사용하고 objectness, no objectness, multi-classification loss는 모두 binary cross entropy를 사용했다. 이 loss를 3가지 scale, 9개 box에 대해 모두 진행한다.

2.1.3. YOLO v3의 장점

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101 [5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

Table 2. Comparison of backbones. Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

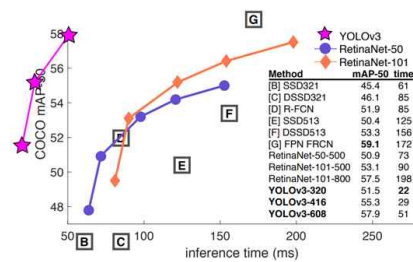


Figure 16. Again adapted from the [15], this time displaying speed/accuracy tradeoff on the mAP@.5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try this guy - [17]. Oh, I forgot, we also fit a data loading lag in YOLOv2, that helped by like 2 mAP. Just seeking this in here to not throw off layout.


그림 15 backbone들의 비교

그림 16 model별 mAP-50

Backbone을 비교해보면 darknet-53이 다른 model들 보다 상대적으로 좋은 정확도를 보인다. 이는 ResNet-101보다 1.5배 빠르며, ResNet-152와 비슷한 성능을 보이지만 두 배 이상 빠르다. 또한, 초당 가장 높은 floating point operation 속도를 보여주는 것으로 보아 GPU를 가장 효율적으로 사용한다고 생각할 수 있다. Yolo v3는 SSD와 성능이 비슷하지만 3배 이상 빠른 속도를 보였다. 즉, inference time 측면에서는 굉장히 빠른 결과를 확인할 수 있다.

2.2. Yolo 활용

training을 진행시키기 위하여 GPU를 사용할 수 있는 Colab으로 코딩을 진행하였다.



```
Invidia-smi
```

Wed Jan 12 15:55:49 2022

NVIDIA-SMI 495.46				Driver Version: 460.32.03				CUDA Version: 11.2			
GPU Name		Persistence-M		Bus-Id		Disp.A		Volatile Uncorr. ECC			
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage		GPU-Util		Compute M.			
MIG M.											
=====											
0	Tesla K80	Off	00000000:00:04.0		Off			0			
N/A	34C	P8	25W / 149W		0MiB / 11441MiB	0%		Default N/A			

Processes:											
GPU	GI	CI	PID	Type	Process name				GPU Memory		
		ID	ID							Usage	
=====											
No running processes found											

그림 17 tesla k80 gpu를 사용

구글드라이브에 image파일과 라벨링을 통하여 만들어진 txt파일을 업로드한 뒤 training을 진행한다.
yolo를 사용하기 위하여 Darknet을 clone 해준뒤에 GPU를 통하여 compile 해준다.

```
[ ] !git clone https://github.com/AlexeyAB/darknet
```

```
[ ] %cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!make
```

이후에 training data set인 이미지 파일과 txt파일을 불러 온 뒤에 학습을 진행한다.

```
[ ] !./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show
```

그림 20 Training을 진행시키는 코드

```
1294: 0.229718, 0.236887 avg loss, 0.001000 rate, 8.389623 seconds, 82816 images, 12.801003 hours left
Loaded: 4.701502 seconds - performance bottleneck on CPU or Disk H2O/SSD
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.751081), count: 3, class_loss = 0.212974, iou_loss = 0.156243, total_loss = 0.369217
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.783493), count: 1, class_loss = 0.388075, iou_loss = 0.017992, total_loss = 0.406067
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82710, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.739478), count: 4, class_loss = 0.397313, iou_loss = 0.174961, total_loss = 0.572274
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.006133, iou_loss = 0.000000, total_loss = 0.006133
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82714, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.767780), count: 2, class_loss = 0.114580, iou_loss = 0.060598, total_loss = 0.175177
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.636308), count: 2, class_loss = 0.645187, iou_loss = 0.315608, total_loss = 0.960796
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82718, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.808823), count: 4, class_loss = 0.078799, iou_loss = 0.111143, total_loss = 0.189942
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.000211, iou_loss = 0.000000, total_loss = 0.000211
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82722, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.767049), count: 3, class_loss = 0.506778, iou_loss = 0.169458, total_loss = 0.676236
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.609172), count: 1, class_loss = 0.115722, iou_loss = 0.238896, total_loss = 0.354618
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82726, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.739692), count: 4, class_loss = 0.290601, iou_loss = 0.294672, total_loss = 0.585272
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.011151, iou_loss = 0.000000, total_loss = 0.011151
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82730, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.789670), count: 3, class_loss = 0.101743, iou_loss = 0.097009, total_loss = 0.198752
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.849237), count: 1, class_loss = 0.231293, iou_loss = 0.074767, total_loss = 0.306061
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82734, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.766940), count: 3, class_loss = 0.299093, iou_loss = 0.108041, total_loss = 0.407134
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.732370), count: 1, class_loss = 0.096023, iou_loss = 0.155488, total_loss = 0.251513
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82738, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.841941), count: 1, class_loss = 0.050542, iou_loss = 0.020054, total_loss = 0.070596
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.544433), count: 3, class_loss = 0.602734, iou_loss = 0.660880, total_loss = 1.263614
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
total_bbox = 82742, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.798336), count: 4, class_loss = 0.444344, iou_loss = 0.132007, total_loss = 0.576441
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.075041, iou_loss = 0.000000, total_loss = 0.075041
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
```

그림 21 training 실행

위는 1294번째 training결과이다. 스팸을 인식하기 위해서 1000번 이상 Training을 하였고 10시간 이상 시간이 소요되었다. training을 진행할수록 average loss가 감소하는데 이를 충분히 감소시킬때까지 training을 진행시킨뒤에 weight파일을 얻는다. Colab으로 training하였기 때문에 구글 드라이브에 weight파일을 저장 시켜 준다. 이를 이용하기 위하여 OpenCV를 이용한 코딩을 해준다. weight파일을 다운받고 알맞은 cfg파일을 이용하여 실행을 시켜준다.

```
import cv2
import numpy as np
import glob
import random

net = cv2.dnn.readNet("/Users/kindaeyeon/Desktop/train_yolo_to_detect_custom_object-2/yolo_custom_detection/yolov3_training_last.weights", "/Users/kindaeyeon/Desktop/train_yolo_to_detect_custom_object-2/yolo_custom_detection/yolov3_training_last.cfg")
classes = ["Spam"]
images_path = glob.glob("/Users/kindaeyeon/Desktop/리창/IMG_1414.jpg")
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))
random.shuffle(images_path)

for img_path in images_path:
    img = cv2.imread(img_path)
    img = cv2.resize(img, None, fx=0.4, fy=0.4)
    height, width, channels = img.shape
    blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)
    class_ids = []
    confidences = []
    boxes = []

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.3:
                print(class_id)
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    print(indexes)
    font = cv2.FONT_HERSHEY_PLAIN

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[class_ids[i]]
            cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
            cv2.putText(img, label, (x, y + 30), font, 3, color, 2)

cv2.imshow("Image", img)
key = cv2.waitKey(0)
cv2.destroyAllWindows()
```

그림 22 파이썬 코드

위와 같은 파이썬 코드를 실행시키면 아래와 같은 결과가 나오게 된다.아래 사진과 같이 spam이라는 class에 대하여 학습이 잘 되었음을 알 수 있다.



그림 23 weight파일을 통해 실행한 결과

2.3. 앱 디자인

2.3.1. 초기 디자인 방향

먼저 앱을 디자인하기 전에 앱에 필요한 기능들을 정리했다. 앱에 들어갈 기능으로 우선 가지고 있는 음식을 카메라를 통해서 인식할 수 있도록 하는 기능, 인식된 음식 중에서 빠르게 사용하고 싶으면 하는 재료들을 선택할 수 있는 기능, 재료들을 이용하여 만들 수 있는 요리들을 추천하는 기능, 그리고 보고 싶은 요리 레시피를 보여주는 기능을 주요 기능으로 선정했다. 이러한 기능들을 바탕으로 구상한 앱의 초기 스케치이다.

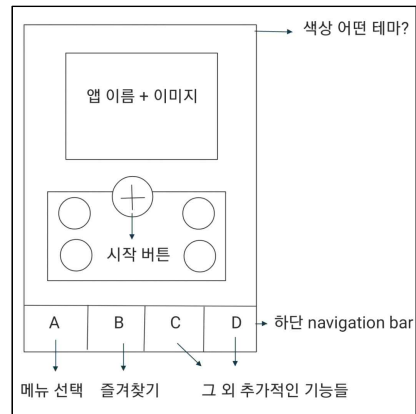


그림 24 앱 초기 스케치

다음으로 추가하면 좋을 것 같은 기능들을 고려했다. 먼저 음식을 인식한 후 추가적으로 인식할 수 있는 기능을 고려하였으며 잘못 인식된 음식의 경우 취소할 수 있는 기능, 즐거찾기 한 음식을 보여주는 기능, 필터를 이용하여 보고싶은 레시피를 정하는 기능, 추천 레시피를 보여주는 기능 등을 추가했다. 그리고 이를 ppt를 이용하여 구현했다.



그림 25 앱 디자인1



그림 26 앱 디자인2

2.3.2. 디자인 진행

초기에 구상한 디자인의 경우 화면이 많이 복잡하고 UI가 불친절할 것이라는 우려가 있었다. 이에 따라서 앞에서 언급한 기능들을 좀 더 접근성이 좋고 편리하게 사용할 수 있도록 디자인을 수정하기로 하였다. 이를 위하여 실제로 사용하고 있었던 앱들을 참고하기로 하였고 등록된 프로필을 확인할 수 있는 앱에서 자주 확인하는 프로필을 메인 화면에 고정할 수 있다는 점에 착안하여 이번에 디자인을 할 앱의 주요기능인 가지고 있는 음식 재료들 추가를 메인 화면에 크게 넣기로 했다. 또한 배달앱인 '배달의민족' 앱의 디자인을 참고하여 레시피를 보여주는 UI를 디자인하였다. 이에 따라 수정하게 된 디자인은 다음과 같다.

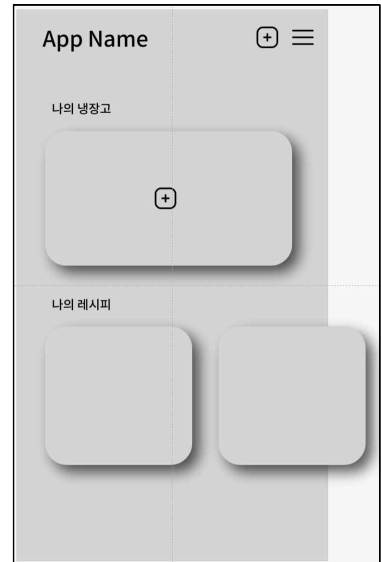


그림 27 앱 디자인3

기본적인 틀을 완성하고 앱에 사용할 색을 정했다. 음식에 전체적으로 어울릴만한 톤을 찾기 위해서 어도비를 이용하여 색 조합을 찾았다. 이에 따라 저희는 옥색 빛을 띄는 색을 이용하여 앱을 디자인하기로 했다.

2.3.3. 디자인 구현

구상한 디자인의 구현을 위하여 어도비 XD를 사용하기로 하였습니다. 앱의 구조는 일반적인 앱들에서 사용하는 사이드 마진 값들을 이용하여 틀을 잡았습니다. 앱의 이름은 임의로 '나만의 냉장고'로 하였으며 사용한 폰트는 구글에서 제공하는 'Roboto', 'NotoSans KR'을 이용하였습니다. 앱 중간중간에 인터페이스 및 버튼 등으로 사용하기 위한 아이콘 등은 오픈소스로 제공되는 'XD material design light and dark themes'를 이용하였습니다.

앱 화면의 크기는 아이폰 X, 아이폰 11 pro를 기준으로 구현하였습니다. 어도비 XD의 프로토타입 기능을 활용한 인터랙션을 이용하여 앱의 UI를 구현하였습니다.



그림 28 앱 디자인4

3. 연구 결과 및 해결방안

3.1. 연구 결과

음식 재료에 대한 데이터를 직접 만들고 이를 통해 음식 재료를 인식하는 인공지능을 만드는 연구를 진행하였다. 이를 위하여 yolo v3를 colab으로 학습을 시키며 연구를 진행시켰다.

자취하는 사람이 가지고 있을 만한 음식 재료를 바탕으로 training을 진행하여 파프리카, 달걀 등의 재료를 인식할 수 있도록 트레이닝 하였다. 이를 통하여 weight파일을 얻고 파이썬을 통하여 인식하는 것을 실제로 진행한다.

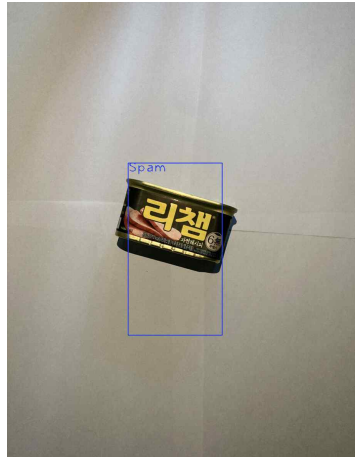


그림 29

Training을 통해 얻은 Weight 파일이 위의 그림과 같이 작동함을 확인할 수 있다. 연구 과정에서 여러 대의 컴퓨터로 많은 training을 시켜 다양한 weight파일을 얻을 수 있었다. 서로 다른 training 데이터셋 간의 차이점을 알 수 있었다. 아래의 그림과 같이 적은 training 데이터셋을 사용하여 짧은 실행시간 동안 training한 결과



그림 30

인식이 많은 training 데이터셋으로 오랜 시간동안 training한 결과보다 인식률이 떨어짐을 알 수 있었다. 이렇게 많은 다양한 training 데이터셋에 대한 training결과를 통하여 데이터셋의 크기가 클수록, training 시간이 적절히 길수록 training해서 얻은 weight파일의 인식률이 상승함을 알 수 있었다.

3.2. 기대효과 및 활용방안

본 연구에서는 재료에 대한 영상인식을 구현하기 위해서 재료 하나당 700~800장 정도의 사진을 사용하여 데이터 셋을 만든 뒤에 데이터 학습을 진행하였다. 이보다 더 많은 사진을 모아 데이터셋을 만든다면 다양한 식재료 종류들을 빠르게 구분할 수 있을 것이며 더 정확하게 재료들을 인식할 수 있을 것으로 기대된다. 이번 연구에서 사용한 영상인식 기술은 YOLO v3이다. 이보다 더 발전된 기술인 YOLO v5 등을 이용하고 성능이 더 뛰어난 GPU를 사용하면 식재료의 영상인식의 정확도와 학습 속도를 향상시킬 수 있을 것이다.

현재는 단순히 재료들의 종류를 구분하는 것에 그쳤지만 식재료 포장지에 적혀있는 유통기한 등을 인식하여 자동으로 빠르게 사용해야 할 음식들을 선택해주고 레시피를 추천해주거나 깻잎과 같은 영상인식을 통해 채소들의 신선도도 확인할 수 있도록 발전할 수 있을 것이다.

또한, 앱을 사용할 경우 사람들이 선호하는 음식의 종류들이 있을 것이다. 사람들이 선호하는 음식들에 대한 통계 자료나 앱을 사용하기 전에 음식의 기호, 알레르기과 같은 개인별 간단한 검사 등을 통해서 개인에게 맞춤형 추천 리스트를 제공해줄 수 있을 것으로 기대된다.

이번 연구에서 구상한 것은 재료들과 사용자가 표시한 먼저 사용해야 할 음식들을 기준으로 레시피를 추천해준다. 또한, 원하는 음식의 종류나 만드는 방법과 같은 여러 옵션을 통해 해당하는 레시피를 찾을 수 있다. 여기에 영양을 고려하여 최대한 균형을 잡힌 음식 순서로도 레시피를 확인할 수 있다면 좋을 것으로 기대된다.

현재 재료 인식은 겉으로 음식들이 구분될 수 있는 상태여야 인식을 할 수 있다. 하지만 자취생들이 일반적으로 부모님께 반찬 등을 받을 때 통 안에 받는 경우가 있을 수 있다. 특히, 큰 통에 담긴 음식들, 대표적으로 김치 등의 경우 인식에 있어서 어려움이 있을 수 있다. 그렇기에 인식이 불가능한 음식 및 재료들의 리스트를 따로 준비하여 추가할 수 있도록 한다면 대부분의 음식과 재료에 대해 적용이 가능하므로 앱의 완성도를 높일 수 있을 것으로 기대된다.

4. 참고 문헌

- [1] Joseph Redmon & Ali Farhadi, YOLOv3: An Incremental Improvement, University of Washington
- [2] Time traveler. (2021년 8월 26일). 13. YOLOv3. <https://89douner.tistory.com/109>
- [3] pulluper. (2021년 4월 19일). [Object Detection] YOLO v3 논문리뷰 (YOLOv3: An Incremental Improvement/arXiv 2018). <https://csm-kr.tistory.com/11>

학부생연구프로그램(UGRP) 개인정보 및 성과물 활용 동의서

개인정보 수집 및 활용 동의서

본인은 교육혁신센터가 '학부생연구프로그램(UGRP)'의 교육적 운영을 위해 팀의 결과물로 제출한 내용에 포함된 개인정보 및 관련한 행사에서의 초상권, 연구결과물을 이용하는데 동의합니다.

1. 개인정보 수집·활용 목적: 프로그램 운영, 관련 보고문서 작성, 비영리 교육홍보자료 이용, 교내 공유 등
2. 개인정보 수집 항목: 성명, 학년, 학과, 학번, 초상권
3. 개인정보 보유 및 이용기간 : 개인정보 수집 및 이용 목적의 종료 시까지

☒ 개인정보 수집 및 이용에 동의함

☐ 동의하지 않음

본인은 위의 모든 내용을 충분히 확인하였으며, 이에 서명합니다.

2022 년 1 월 17 일

대표자
(팀장)

학번/성명: 20200843 / 임승민 (서명)

포항공과대학교 귀하