

차량지능기초 과제 보고서

20181567 강 민 수

Object detection Open Data set

현재 차량자율주행의 환경인지 방안으로는 레이더나 라이다 등 수많은 센서가 사용되고 있으나 그 중에서도 가장 많이 활용되는 분야는 카메라를 이용한 영상처리분야이다.

영상처리란 카메라를 통해 주변환경을 읽어오고 차량 주변사물을 인지하고 자신이 어떠한 상황에 처해있는지 아는 것이라 생각한다. 그렇다면 인공지능은 어떤 방법으로 주변의 사물을 인지할 수 있는가? 이는 수많은 반복학습을 통해 이루어지는데 모델의 정확성을 높이기 위해서는 질 좋은 데이터가 많이 필요하다. 그러한 데이터들의 모음을 데이터셋이라 하며 개인이 구축하기 힘들기 때문에 최근에는 무료로 개인에게 제공되는 데이터셋이 증가했다.

PASCAL VOC Dataset

데이터 설명 (개요)

Pascal Voc Challenge 을 위해 제공되던 데이터셋

보통 voc2007/2012 를 많이 이용함(<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/>)

AP(Average precision)를 기반으로 하는 평가방식을 사용함.

데이터 종류, 특징

Xml format

20 개의 object 카테고리

이미지 개수: 11k

이미지당 평균 object 개수: 2.4 개

IOU threshold 값: 0.5 (맞다고 추측하는 값 True Positive 의 조건을 0.5 로 설정)

```
└ Object
  └ Vehicle
    ├── 4wheeled
    │   ├── Car
    │   └── Bus
    ├── 2wheeled
    │   ├── Bicycle
    │   └── Motobike
    ├── Aeroplane
    ├── Boat
    └── Train
  └ Household
    └ Furniture
```

```

    |   |   | Seating
    |   |   | | Chair
    |   |   | | Sofa
    |   |   | Dining table
    |   | TV/monitor
    |   | Bottle
    |   | Potted plant
    | L Animals
    | | Domestic
    | | | cat
    | | | Dog
    | | Farmyard
    | | | Cow
    | | | Horse
    | | | Sheep
    | | Bird
    | L Person

```

- **Person:** person
- **Animal:** bird, cat, cow, dog, horse, sheep
- **Vehicle:** aeroplane, bicycle, boat, bus, car, motorbike, train
- **Indoor:** bottle, chair, dining table, potted plant, sofa, tv/monitor

계층구조

```

├ VOCdevkit
├ └ VOC2012
    | Annotations
    | ImageSets
    | | Layout
    | | Main
    | | Segmentation
    | JPEGImages
    | SegmentationClass
    | SegmentationObject

```

```

<annotation>
  <folder>VOC2012</folder>
  <filename>0001.jpg</filename>
  <source>
    <database>The VOC2012 Database</database>
    <annotation>PASCAL VOC2012</annotation>
    <image>flicker</image>
  </source>

  <size>
    <width>450</width>
    <height>319</height>
    <depth>3</depth>
  </size>
  <segmented>1</segmented>
  <object> //개별 object 정보
    <name>kangaroo</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>

    <bndbox> //object 의 bounding Box 정보
      <xmin>233</xmin>
      <ymin>89</ymin>
      <xmax>386</xmax>

```

```

        <ymax>262</ymax>
    </bndbox>
</object>
... // 개별 object 정보
</annotation>

```

(Annotation 파일 0001.xml 파일의 일부, 0001.jpg 파일에 대한 annotation 정보를 가지고 있음.)

Annotations	이미지의 Detection 정보를 별도의 설명 파일로 제공하는 것을 말함. Pascal Voc에서는 xml 포맷으로 object의 bounding Box의 위치와 object의 이름 등이 포함됨.
imageSet	어떤 이미지를 train, test, trainval, val에 사용할 것인지에 대한 매핑 정보를 개별 object 마다의 텍스트 파일로 가지고 있음.
JPEGImages	Detection과 Segmentation에 사용될 원본이미지
SegmentationClass	Semantic Segmentation에 사용될 masking 이미지
SegmentationObject	instance Segmentation에 사용될 masking 이미지

데이터 예시/활용

Pascal Voc Challenge(2007 - 2012)을 위해 제공되었으며, 2007년 전 많은 평가방식 중 AP 기반의 평가 방식을 사용하였으며, 이를 업계 표준으로 보편화 시켰음.

(MS coco의 평가방식 또한 AP 기반의 평가방식으로 표준화에 기여)

현재는 모델학습 목적이 아닌 벤치마킹용도로만 사용되는 데이터 셋
object detection network에서 MS coco와 함께 사용됨.

데이터 셋 다운로드(미러사이트): <https://pjreddie.com/projects/pascal-voc-dataset-mirror/>

Image net data set (LSVRC2012)

데이터 설명 (개요)

가장 대표적인 대규모(large-scale)데이터 셋이자 논문에서 가장 많이 사용되는 데이터 셋(최신 논문 중에도 image net LSVRC2012 data set을 이용하는 경우가 많다), [ImageNet Large Scale Visual recognition Challenge \(ILSVRC\)](#) 공모전(computer vision 분야에서 높은 권위를 가진 공모전, 2017년 종료)에서 본 데이터 셋을 사용하기 때문에 유명하며, Amazon Mechanical Turk 서비스를 이용해서 사람이 일일이 분류한 데이터 셋이기 때문에 이미지가 많고 다양하기 때문에 많은 CNN 모델을 pretrained 할 때 사용, 현대 인공지능을 있게 해준 데이터 셋이다. 이미지 부류가 다양해서 일상생활에서 볼 수 있는 대부분의 이미지를 포함하며, WordNet 계통구조에 따라 조직화되어 있다.

현재도 classification에 관심있는 사람이라면 대개 Image net data set을 응용한다.

데이터 종류

Xml format

21841 개의 object 카테고리

이미지 개수: 14,197k (Pascal VOC 의 약 1300 배)

총 용량: 총 합 200 GB 이상/ (LSVRC2012 training data set: 138GB)

(전체 lable 확인 : <https://www.anishathalye.com/media/2017/07/25/imagenet.json>)

계층구조

```
<annotation>
  <filename>0001.jpg</filename>
  <size>
    <width>450</width>
    <height>319</height>
  </size>
  <object> //개별 object 정보
    <name>kangaroo</name>
    <bndbox> //object 의 bounding Box 정보
      <xmin>233</xmin>
      <ymin>89</ymin>
      <xmax>386</xmax>
      <ymax>262</ymax>
    </bndbox>
  </object>
  ... // 개별 object 정보
</annotation>
```

Pascal Voc 에서 본 xml forma 과 유사함

데이터 활용/예시

전분야에 걸쳐 사용되며, 논문에 가장 많이 사용하는 데이터 셋임.

feature extractor network 을 pretrained 할 때 사용함.

Image net LSVRC data set 은 **test dataset** 을 제공하지 않고

training, validation data set 만 제공함

학습 Training data set (138GB, 120 만)

//다운로드 (배포종료)

```
wget http://www.image-  
net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012\_img\_train.tar
```

//백그라운드 다운 (배포종료)

```
nohup wget http://www.image-  
net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012\_img\_train.tar
```

//토렌트 링크

<https://academictorrents.com/collection/imagenet-2012>

검증 Validation set(6.3GB, 5 만)

```
wget http://www.image-net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012_img_val.tar
```

압축풀기

Training data set

```
mkdir train && mv ILSVRC2012_img_train.tar train/ && cd train
tar -xvf ILSVRC2012_img_train.tar

rm -f ILSVRC2012_img_train.tar (만약 원본 압축파일을 지우려면)
find . -name "*.tar" | while read NAME ; do mkdir -p "
${NAME%.tar}"; tar -xvf "${NAME}" -C "${NAME%.tar}"; rm -f "
${NAME}"; done
cd..
```

validation data set

```
mkdir val && mv ILSVRC2012_img_val.tar val/ && cd val && tar -xvf
ILSVRC2012_img_val.tar
wget -
qO- https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.s
h | bash
```

validation data set 정리(전처리)

<https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh>

위 경로를 통해서 쉘 스크립트를 다운로드 및 실행하면 이미지를 폴더별로 정리한다.

MS coco

데이터 설명 (개요)

COCO 데이터 셋은 ImageNet의 문제점을 해결하기 위해서 제안되었으며, 평가방식은 pascal voc와 함께 AP 기반 평가방식을 채택하였으나, mAP 계산시 IoU를 유동적으로 사용하거나 AR(Average Recall)을 이용해 평가하는 방식을 사용함.

다양한 크기와 배치를 가진 Object

다양한 크기의 물체가 하나의 이미지에 담겨있으며, 이는 높은 확률로 소형 object를 포함함



낮은 명확성을 가진 이미지

image Net 데이터셋의 object class 의 명확성이 높았던 것과는 다르게 COCO 데이터셋에서 제공되는 이미지는 object 의 class 의 명확성이 낮아 실제상황과 더 유사한 학습데이터를 가지고 있음.



(명확성이 높은 이미지)

(명확성이 낮은 이미지)

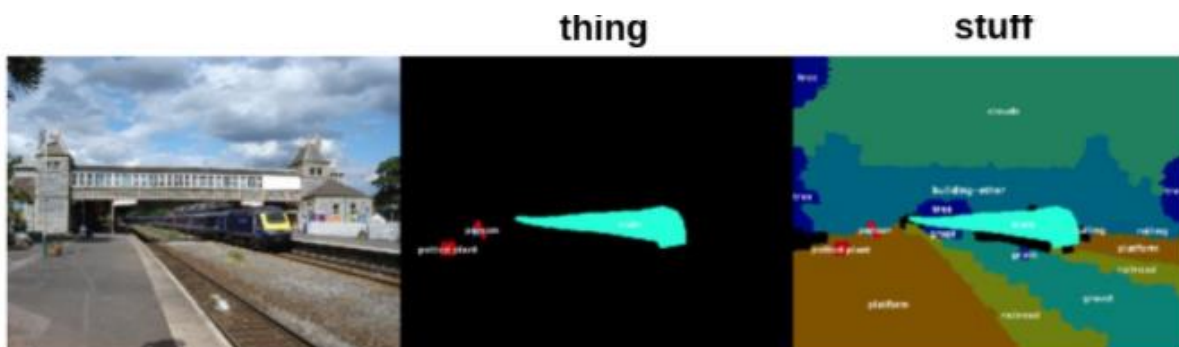
현실적 Object 배치

Object 들이 정리되어 있거나 서로 떨어져있는 것이 아닌 교합되어있음.



Thing & Stuff

Thing(쉽게 레이블링 될 수 있는 object) 와 Stuff(쉽게 레이블링이 되지않는 object)를 구분하여 Thing 만 레이블링 하였음.



데이터 종류

json format

80 개의 object 카테고리

이미지 개수: 320k (Pascal VOC 의 약 30 배)

이미지당 평균 object 개수: 8 개

IOU threshold 값: 0.5 – 0.95 (유동성을 가짐)

계층구조

{//json 파일 구조

```
    "info": info,  
    "licenses": [license],  
    "categories": [category],  
    "image": [image],  
    "annotations": [annotation]
```

}

"info" //데이터세트에 대한 정보를 공개함

```
{  
  "year": int, //2019  
  "version": str, //"1.0"  
  "description": str, //"Flower and Fruits dataset"  
  "contributor": str, //"Flowers Inc."  
  "url": str, //"http://test.org"  
  "data_created": datetime, //"2019/12/04"  
}
```

"license" //데이터 세트에 대한 다양한 이미지 라이선스 정보를 가짐

```
[{  
  "id": int, //1  
  "name": str, //"Free License"  
  "url": str, //"http://flower.org/"  
}]
```

"Categories" //각 object 의 카테고리

```
[{  
  "id": int, //카테고리 ID 는 고유해야함  
  "name": str, //카테고리 이름 ex) 장미, 백합, 튜립 등
```

```

"supercategory": str, //카테고리는 상위 카테고리에 속할 수 있음. Ex) 꽃
}]
/*"categories:
[
    {"id": 1, "name": "rose", "supercategory": "flower"}
    {"id": 2, "name": "tulip", "supercategory": "flower"}
    {"id": 10, "name": "Apple", "supercategory": "fruit"}
]*/

```

```

image{ //데이터 세트의 모든 이미지 목록을 포함
"id": int, //397133 (고유성을 가져야함)
"width": int, //640
"height": int, //640
"file_name": str, //"101.jpg
"license": int, //1
"flickr_url": str, //선택사항
"coco_url": str, //선택사항
"data_captured": datetime, //"2019-12-04 17:02:52
}

```

```

annotation{ // 데이터셋에 있는 모든 이미지별 object 에 대한 annotation 목록
"id": int,
"image_id": int,
"category_id": int,
"segmentation": RLE or [polygon], // segmentation mask 에 대한 모든 개체인스턴스 주변에
                                대한 x 및 y 좌표를 가짐
"area": float, //bounding box 의 픽셀값
"bbox": [x,y,width,height], //bounding box 의 왼쪽상단값(x,y), 너비 및 높이좌표
"iscrowd": 0 or 1, //단일 object segmentation 이 있는 경우 iscrowd 를 0 으로 설정
            //이미지내 object 모음에 대해서는 iscrowd 를 1 로 설정, RLE 사용

// RLE(Run Length Encoding) : 반복는 값을 반복 횟수로 대체하여 압축하는 방식
// (0 11 0111 00) -> (12132)
}

```


Annotations	데이터셋에 있는 모든 이미지의 모든 개별 Object annotation 목록	
	유형	목록
	-object detection -keypoint detection -stuff segmentation -panoptic segmentation -image captioning	-segmentation (-RLE) -area -iscrowd -imageid -bbox
Images	bounding box 나 segmentation 정보 없이 데이터셋의 모든 이미지 정보를 가지며, 이미지 ID 는 고유성을 가짐	
Categories	Categories 목록을 포함, Categories 는 상위 Categories 에 속할 수 있음	
Licenses	데이터셋의 이미지에 적용되는 이미지 라이선스 목록을 포함	
Info	데이터셋에 대한 high-level 정보를 포함	

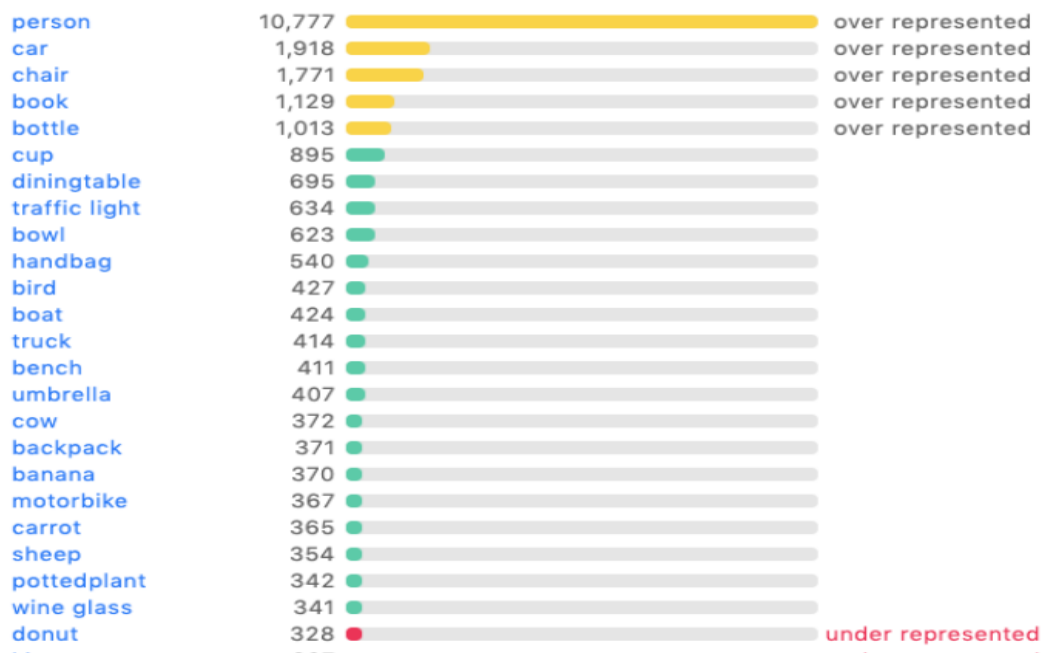
데이터 활용/예시

Image net 을 대체하며 차세대 학습에 사용하나 Class balance 가 뛰어나지는 않음.

Image net 과는 다르게 object detection network 에서 사용됨

person, car, chair, book 등 상위 몇 가지 class 에 치중 되어있음, 자율주행모델 테스트에서는 좋은 성능을 보임. 또한, Image Net 을 대체하려고 구축되었으나 규모가 아직 그에 못 미침

Class Balance



coco dataset 을 이용하여 매년 detection, keypoints, stuff, Panopti, Captions 의 카테고리별
매년 기업과 학생들을 상대로 대회를 운영하고 있음.

데이터 셋 받기

coco 데이터 셋은 크기가 커서 curl 유틸을 사용해서 사용함

데이터셋 다운로드 링크 : <https://cocodataset.org/#download>

Curl 다운로드 (colab 에서는 기본으로 wget 과 같이 사용가능함)

```
$ sudo apt install curl
$ curl https://sdk.cloud.google.com | bash
$ source ~/.bashrc
```

COCO 데이터셋 다운

```
$ mkdir COCO $ cd COCO $ mkdir val2017 $ gsutil -m rsync
gs://images.cocodataset.org/val2017 val2017 $ mkdir annotation
$ gsutil -m rsync gs://images.cocodataset.org/annotations annotation $ cd val2017
$ unzip ../anns/annotations_trainval2017.zip
```

다른 데이터 다운

```
$ gsutil -m rsync gs://images.cocodataset.org/train2017 train2017
```

데이터 셋 종류

Images

2014 Train images [83K/13GB]
2014 Val images [41K/6GB]
2014 Test images [41K/6GB]
2015 Test images [81K/12GB]
2017 Train images [118K/18GB]
2017 Val images [5K/1GB]
2017 Test images [41K/6GB]
2017 Unlabeled images [123K/19GB]

Annotations

2014 Train/Val annotations [241MB]
2014 Testing Image info [1MB]
2015 Testing Image info [2MB]
2017 Train/Val annotations [241MB]
2017 Stuff Train/Val annotations [1.1GB]
2017 Panoptic Train/Val annotations [821MB]
2017 Testing Image info [1MB]
2017 Unlabeled Image info [4MB]

Open Source Object detection project

RCNN 계열

개요/특징

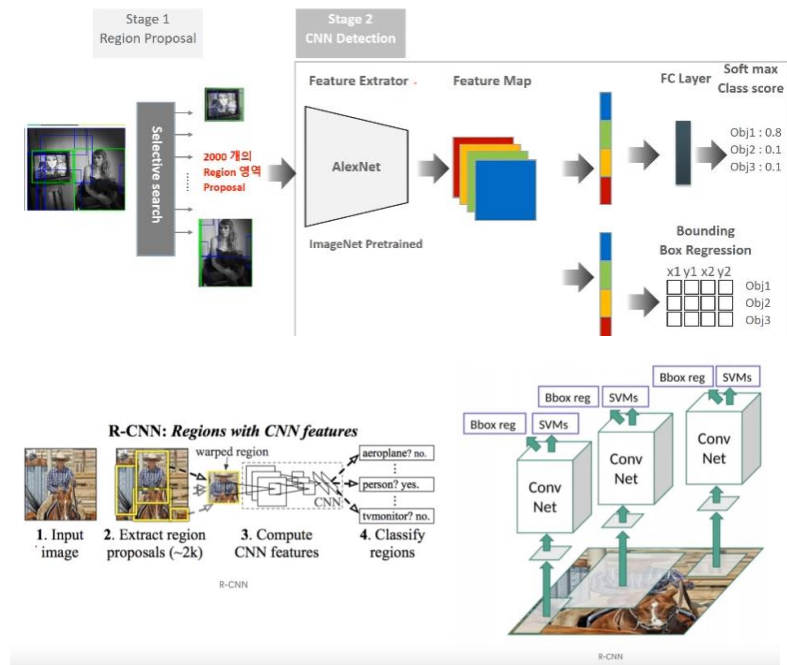
RCNN

Sliding Window 방식이 아닌 Region Proposal 방식을 사용하기 때문에 object 의 경계값을 구분하여 위치를 예측함. 반복횟수가 줄고 효율성이 증가하였음.

RCNN 은 region Proposal(selective search)방식과 CNN Detection 을 융합하여 만든 신경망 알고리즘임.

RCNN 은 crop 과 warp 을 통해 이미지 크기를 동일하게 맞추고 2000 개의 region 영역을 추출하고 CNN network 을 통해 object detection 을 시행함

2000 번을 시행하기 때문에 수행시간이 오래걸림

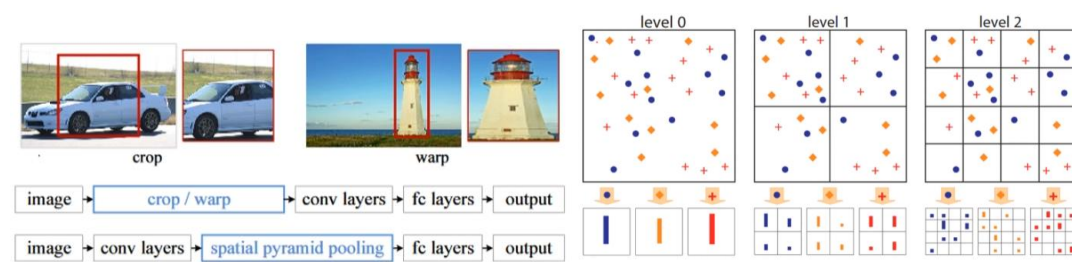


SPP-NET/Fast-RCNN

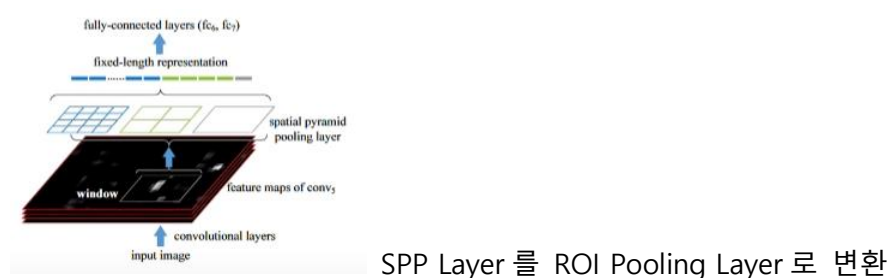
기존 RCNN 의 문제 :

너무 오래 걸림 / 고정된 이미지파일을 요구함

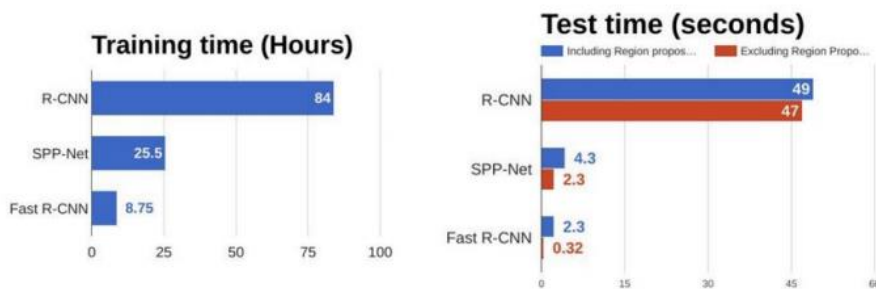
SPP(spatial pyramid pooling)와 기존 RCNN 을 융합하여 위와 같은 문제를 해결하였음.



(8x8 feature map 에서 2+4+16 21 개의 원소로 vector 값으로 표현가능/분면에 따라 다름)



SPP Layer 를 ROI Pooling Layer 로 변환



RCNN 과 비교를 했을 때 학습/수행 시간을 대폭 줄일 수 있었음. (위 표는 CPU 기반환경)

Faster RCNN

Faster RCNN = RPN(Region Proposal Network) + Fast RCNN

기존 selective search 의 문제점을 보완하기 위해서 neural network 구조로 변경

GPU 를 통한 연산이 가능해지면서 CPU 연산의 한계를 뛰어넘음. 또한,

RCNN 기반의 Network 단일 구조 모델(End to End) 구현에 성공할 수 있었음.

Tensor flow 환경에서 사용

Inference graph 를 읽음.

```
with tf.gfile.GFile(os.path.join(default_rcnn_dir, 'pretrained/fast
er_rcnn_resnet50_coco_2018_01_28/frozen_inference_graph.pb'), 'rb') as
f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
```

session 시작 후 Inference graph 모델 로딩

```
with tf.Session() as sess:
    # Session 시작하고 inference graph 모델 로딩
    sess.graph.as_default()
    tf.import_graph_def(graph_def, name='')
```

모델 수행

```
out = sess.run([sess.graph.get_tensor_by_name('num_detections:0'),
    sess.graph.get_tensor_by_name('detection_scores:0'),
    sess.graph.get_tensor_by_name('detection_boxes:0'),
    sess.graph.get_tensor_by_name('detection_classes:0')],
    feed_dict={'image_tensor:0': inp.reshape(1, inp.shape[0], inp.shape[1], 3)})
```

인자로 들어가는 tensor 의 목록

num_detections: detection 된 개수

detection_scores: detection 된 object 들의 score

detection_boxe: 좌표

detection_classes: detected 된 object 들의 class

Bounding box 시각화

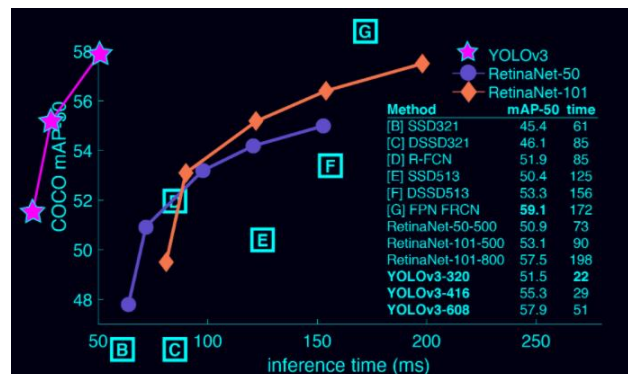
```
num_detections = int(out[0][0])
for i in range(num_detections):
    classId = int(out[3][0][i])
    score = float(out[1][0][i])
    bbox = [float(v) for v in out[2][0][i]]
    if score > 0.5:
        left = bbox[1] * cols
        top = bbox[0] * rows
        right = bbox[3] * cols
        bottom = bbox[2] * rows
        cv2.rectangle(draw_img, (int(left), int(top)), (int(right),
int(bottom)), green_color, thickness=2)
        caption = "{}: {:.4f}".format(labels_to_names[classId], score)
        print(caption)
        cv2.putText(draw_img, caption, (int(left), int(top - 5))
, cv2.FONT_HERSHEY_SIMPLEX, 0.4, red_color, 1)
```

YOLO(You Only Look Once)

개요/특징

YOLO 는 단 network 단일 구조 (End to End)을 통해 이미지내에 bounding box 와 class propability 예측 하나의 회귀문제로 해결했기 때문에 기존의 다중신경망(ex. RCNN) 기반보다 더 빠르게 예측할 수 있음.

(Pretrained 에서 사용된 data set 은 논문 발표기준 MS coco)



Open CV 환경에서 사용 (Keras 환경은 실습에서 설명)

Darknet 에서 받을 수 있는 pretrained inference model 을

사용(<https://pjreddie.com/darknet/yolo/>)

Cv2.dnn.readNetFromDarknet(config,weight)으로 pretrained inference model 다운

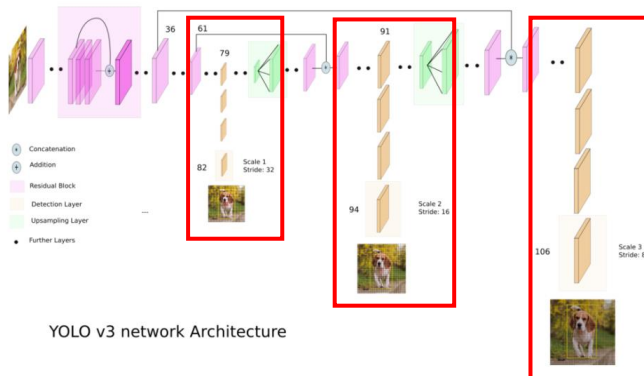
```
weights_path = os.path.join(CUR_DIR, 'pretrained/yolov3.weights')
config_path = os.path.join(CUR_DIR, 'pretrained/yolov3.cfg')
cv_net_yolo = cv2.dnn.readNetFromDarknet(config_path, weights_path)
```

3 개의 scale output layer(82,94,106)에서 직접 detection 결과를 가져와야함.

(다 행렬형태로 합쳐서 가져옴.)

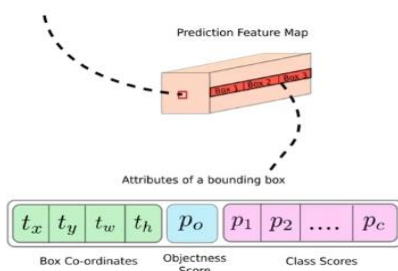
```
layer_names = cv_net_yolo.getLayerNames()
outlayer_names =
[layer_names[i[0] - 1] for i in cv_net_yolo.getUnconnectedOutLayers()]
```

```
# Object Detection 수행하여 결과를 cvOut 으로 반환
cv_outs = cv_net_yolo.forward(outlayer_names)
```



사용자가 직접 NMS(open cv 모듈)로 최종 결과 필터링을 해야함.

COCO 데이터는 80 개의 object class 를 갖고 있기 때문에



(1-4 : bounding box / 5 : object score / 6 – 85 : class Score, confidence 확률)

가장 큰 Class Score 를 가진 object 를 기준 값과 비교

```
if confidence > conf_threshold:
```

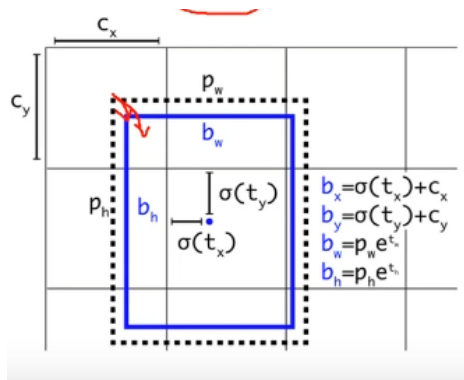
Class score 에서 가장 큰 점수를 가지고 object class 를 판단.

```
print('ix:', ix, 'jx:', jx, 'class_id', class_id, 'confidence:', confidence)
```

추출 좌표의 변환

Bounding box (index: 0 - 4)내 enter x, y 을 좌 상단/우 하단 위치 값으로 변화 해야함.

```
center_x = int(detection[0] * cols)
center_y = int(detection[1] * rows)
width = int(detection[2] * cols)
height = int(detection[3] * rows)
left = int(center_x - width / 2)
top = int(center_y - height / 2)
```



$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

3 개의 scale output layer(82,94,106) 행렬로 모으기

```
boxes.append([left, top, width, height]) //다 행렬 값으로 모아서 가져옴
```

NMS 로 object 간 겹치는 부분의 bounding box 삭제

```
idxs = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
```

object 정보 추출 및 시각화

```
if len(idxs) > 0:
    for i in idxs.flatten():
        box = boxes[i]
        left = box[0]
        top = box[1]
        width = box[2]
        height = box[3]
        caption = "{}: {:.4f}".format(labels_to_names_seq[class_ids[i]], confidences[i])
        cv2.rectangle(draw_img, (int(left), int(top)), (int(left+width),
                                                             int(top+height)), color=green_color, thickness=2)
        cv2.putText(draw_img, caption, (int(left), int(top - 5)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, red_color, 1)
        print(caption)
```

실습

구현 환경 (시스템, lib, docker 등 실행 환경) 및 실행에 관련된 코드 설명 등

실습환경: [google colab](https://colab.research.google.com/)

사용 모델: [Keras-YOLOv3](https://github.com/qywee/keras-yolo3) (Darknet 보다는 느리지만 코드가 간결하고 커스텀마이징이 쉬움)

Yolo open source package: [qqwwee keras-yolo3](https://github.com/qywee/keras-yolo3)(<http://github.com/qqwwee/keras-yolo3>)

참고 자료: <https://github.com/chulminkw/DLCV.git>

Git hub link: https://github.com/pharrcy/KMU_Vehicle-intelligence.git

현재 디렉토리는 /content 이며 이 디렉토리를 기준으로 실습코드와 데이터를 다운로드 합니다.

```
!pwd
!rm -rf DLCV
!git clone https://github.com/chulminkw/DLCV.git
```

Tensor flow 1.15.2 설치/ keras2.3.0 설치

```
!pip install tensorflow-gpu==1.15.2
!pip install keras==2.3.0
```

Tensor flow/keras 를 프로젝트에 import

```
import tensorflow as tf
import keras
```

Tensor flow/keras 버전확인 및 GPU 연산장치 확인

```
print(tf.__version__)
print(keras.__version__)
tf.test.gpu_device_name()
```

프로젝트에서 사용되는 여러가지 라이브러리 import

```
import os
import sys
import random
import math
import time
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

keras-yolo3(colab ver.) 패키지 다운 및 /content/DLCV/Detection/yolo 하위 주소에 설치

```
%cd /content/DLCV/Detection/yolo
!git clone https://github.com/qgwweee/keras-yolo3.git
!ls -lia /content/DLCV/Detection/yolo/keras-yolo3
```

Local Directory 상에서 yolo 패키지 import (절대경로 사용, keras-yolo3 는 setup 을 제공 X)

```
default_dir = '/content/DLCV'
default_yolo_dir = os.path.join(default_dir, 'Detection/yolo')

LOCAL_PACKAGE_DIR = os.path.abspath(os.path.join(default_yolo_dir, 'keras-yolo3'))
print(LOCAL_PACKAGE_DIR)
sys.path.append(LOCAL_PACKAGE_DIR)

from yolo import YOLO
```

Yolov3.weights 파일을 다운로드하고 convert.py 를 수행

```
%cd /content/DLCV/Detection/yolo/keras-yolo3
```

yolo 공식 사이트에서 다운 받을 시 속도가 느려 다른 github 링크로 내려받음.

```
!wget https://github.com/chulminkw/DLCV/releases/download/1.0/yolov3.weights
```

Yolov3.weights 를 keras-yolo3 에서 사용할 수 있도록 yolo.h5 로 변환

```
!python convert.py yolov3.cfg yolov3.weights model_data/yolo.h5
```


필요 라이브러리 import

```
import sys
import argparse
from yolo import YOLO, detect_video
```

keras-yolo 에서 image 처리를 주요 PIL 로 수행

```
from PIL import Image
```

YOLO 객체 생성 config 는 default 로 keras-yolo3 디렉토리에 있는 yolov3.cfg 를 적용
colab 버전은 절대 경로로 각 생성 인자 값 입력

```
default_yolo_dir = '/content/DLCV/Detection/yolo'
config_dict = {}
yolo = YOLO(model_path=os.path.join(default_yolo_dir, 'keras-yolo3/model_data/yolo.h5'),
            anchors_path=os.path.join(default_yolo_dir, 'keras-yolo3/model_data/yolo_anchors.txt'),
            classes_path=os.path.join(default_yolo_dir, 'keras-yolo3/model_data/coco_classes.txt'))
```

원본이미지 출력

```
default_dir = '/content/DLCV'
img = Image.open(os.path.join(default_dir, 'data/image/beatles01.jpg'))

plt.figure(figsize=(12, 12))
plt.imshow(img)
```



yolo.detect_image() 메소드는 PIL package 를 이용하여 image 작업 수행.

keras-yolo3/font 디렉토리를 상위 디렉토리로 복사 해야함.

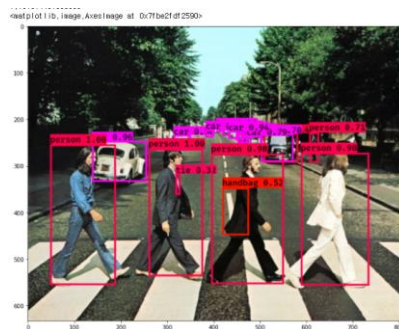
```
%cd /content/DLCV/Detection/yolo
!cp -rf keras-yolo3/font ./font
```

Object detection

```
img = Image.open(os.path.join(default_dir, 'data/image/beatles01.jpg'))
detected_img = yolo.detect_image(img)

plt.figure(figsize=(12, 12))
plt.imshow(detected_img)
```

```
(416, 416, 3)
Found 19 boxes for img
tie 0.32 (318, 316) (328, 362)
handbag 0.52 (419, 347) (476, 449)
truck 0.43 (508, 229) (582, 293)
car 0.30 (406, 215) (418, 227)
car 0.35 (452, 231) (472, 251)
car 0.42 (419, 221) (435, 235)
car 0.53 (460, 232) (481, 256)
car 0.54 (448, 227) (467, 245)
car 0.56 (314, 233) (351, 257)
car 0.70 (502, 230) (572, 291)
car 0.79 (471, 235) (503, 261)
car 0.93 (383, 222) (400, 238)
car 0.94 (432, 225) (452, 242)
car 0.96 (138, 246) (255, 334)
person 0.71 (606, 228) (627, 233)
person 0.98 (335, 274) (351, 554)
person 0.98 (588, 272) (735, 557)
person 1.00 (48, 254) (189, 556)
person 1.00 (260, 263) (377, 537)
7.791874457000063
<matplotlib.image.AxesImage at 0x71be21df2590>
```



영상 내 object detection

```
import cv2
import time

def detect_video_yolo(model, input_path, output_path=""):

    start = time.time()
    cap = cv2.VideoCapture(input_path)

    #codec = cv2.VideoWriter_fourcc(*'DIVX')
    codec = cv2.VideoWriter_fourcc(*'XVID')
    vid_fps = cap.get(cv2.CAP_PROP_FPS)
    vid_size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    vid_writer = cv2.VideoWriter(output_path, codec, vid_fps, vid_size)

    frame_cnt = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    print('총 Frame 갯수:', frame_cnt, '원본 영상 FPS:',vid_fps)
    index = 0
    while True:
        hasFrame, image_frame = cap.read()
        if not hasFrame:
            print('프레임이 없거나 종료 되었습니다.')
            break
        start = time.time()
        # PIL Package 를 내부에서 사용하므로 cv2 에서 읽은 image_frame array 를 다시 PIL 의 Image 형태로 변환해야 함.
        image = Image.fromarray(image_frame)
        # 아래는 인자로 입력된 yolo 객체의 detect_image()로 변환한다.
        detected_image = model.detect_image(image)
        # cv2 의 video writer 로 출력하기 위해 다시 PIL 의 Image 형태를 array 형태로 변환
        result = np.asarray(detected_image)
        index +=1
        print('#### frame:{0} 이미지 처리시간:{1}'.format(index, round(time.time()-start,3)))

        vid_writer.write(result)

    vid_writer.release()
    cap.release()
    print('### Video Detect 총 수행시간:', round(time.time()-start, 5))
```

영상 및 좌표 지정

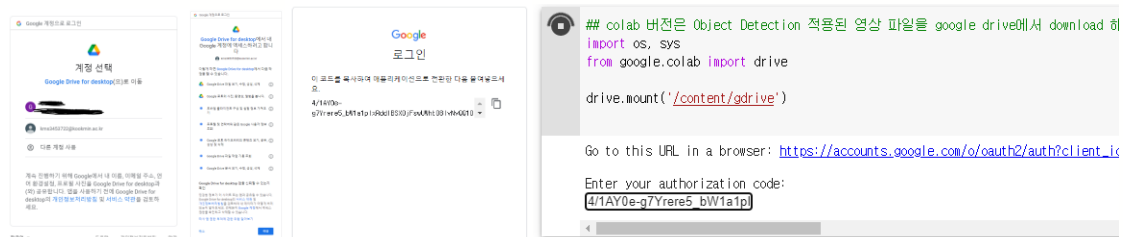
```
default_dir = '/content/DLCV'
detect_video_yolo(yolo, os.path.join(default_dir, 'data/video/Night_Day_Chase.mp4'),
                  os.path.join(default_dir, 'data/output/Night_Day_Chase_yolo_01.avi'))

#### frame:1392 이미지 처리시간:0.049
(416, 416, 3)
Found 1 boxes for img
suitcase 0.56 (325, 14) (1212, 500)
0.049398816000044554
#### frame:1393 이미지 처리시간:0.052
프레임이 없거나 종료 되었습니다.
### Video Detect 총 수행시간: 0.05873
```

구글 드라이브 내 결과 저장

```
import os, sys
from google.colab import drive

drive.mount('/content/gdrive')
```

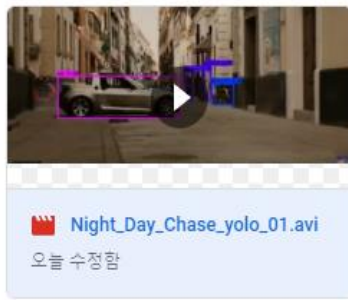


```
## colab 버전은 Object Detection 적용된 영상 파일을 google drive에서 download 하
import os, sys
from google.colab import drive

drive.mount('/content/gdrive')

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client\_id=4/1AY0e-g7Yrre5\_bW1a1p

Enter your authorization code:
4/1AY0e-g7Yrre5_bW1a1p
```



결과 확인 링크

<https://drive.google.com/file/d/1lerNI3ADTUzMQYa9yli3Lsa8Cyomo7jS/view?usp=sharing>