메디치소프트 기술연구소

2018.07.14

딥러닝 활용 : 사물 검출 (CNN, 컴퓨터 비젼)

1.1 YOLOv2 정의

- YOLO는 오브젝트의 경계 박스를 표시하고 동시에 클래스를 분류하는 실시간 오브젝트 검출기이다.

- YOLOv2는 YOLO 모델에서 성능과 속도 면에서 개선을 시킨 두 번째 버전이다.

1.2 YOLOv2 원리

▪ YOLOv2의 성능 향상 원리는 다음과 같다.

1) Batch Normalization : 모든 컨볼루션 레이어에 배치 정규화를 추가
2) High Resolution Classifier : ImageNet 데이터로 classfication network를 먼저 학습시켜서 고해상도 이미지에도 잘 동작하게 함
3) Convolutional : FCL(Fully Conneted Layer)를 Convolution Layer로 대체
   - 기존의 YOLO는 마지막 단의 FCL에서부터 경계 박스의 좌표정보를 직접 예측했다. 하지만 YOLOv2에서는 FCL을 Convolution으로 대체했다.

1.2 YOLOv2 원리

4) Anchor Boxes : 경계 박스를 처음부터 직접 예측하던 방식에서 앵커 박스를 초기값으로 사용하여 예측

   - 모든 anchor 박스에 대해서 그것이 오브젝트일 확률을 예측하고 클래스를 분류한다.

   - 예측된 경계 박스가 오브젝트일 때 그것이 어떤 클래스인지를 예측하기 때문에 조건부 확률을 이용한다.

5) new network : Darknet-19 를 특징 추출기로 사용

   - Darknet은 Maxpool이 빠르게 시작되며 중간 과정에서 1 x 1 convolution을 통해서 특징맵의 채널 수를 절반으로 줄인다.

6) Dimension Clusters : 실제 경계 박스들을 클러스터링하여 최적의 앵커박스를 찾음

7) Location prediction : 경계 박스가 그리드 셀에서 벗어나지 않게 제약을 둠

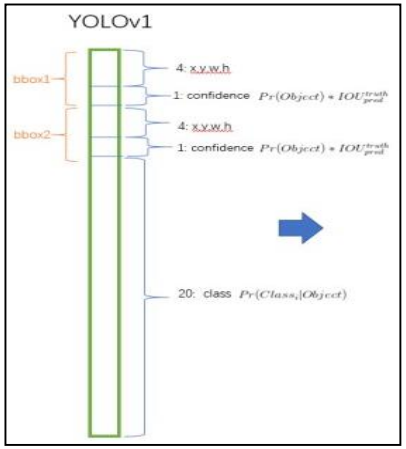
8) Passthrough : 26x26 크기의 중간 특징맵을 skip 하여 13x13레이어에 붙임(concatenate)

1.2 YOLOv2 원리

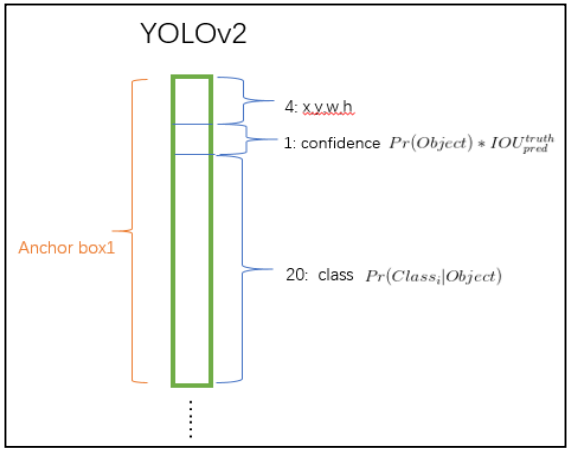
8) Multi-Scale Training : 학습데이터의 크기를 320x320, 352x352, ..., 608x608 로 resize 하면서 다양한 스케일로 학습시킴

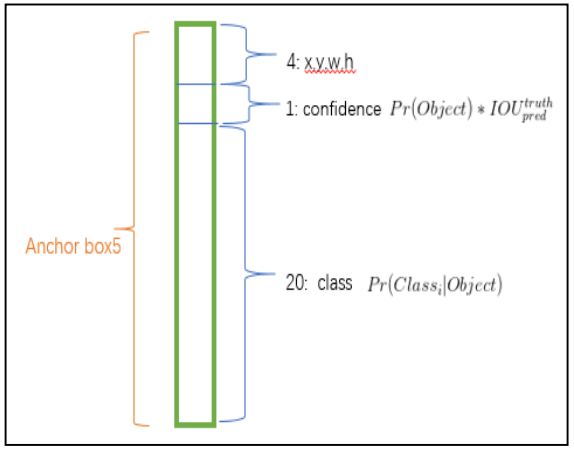
9) Fine-Grained Features : 최종 특징맵의 크기를 7x7에서 13x13으로 키움

- 다음 그림은 YOLO와 YOLOv2의 최종 특징맵의 차이를 나타낸다.



[그림] YOLO의 최종 특징 맵





[그림] YOLOv2의 최종 특징 맵

1.3 테스트 환경

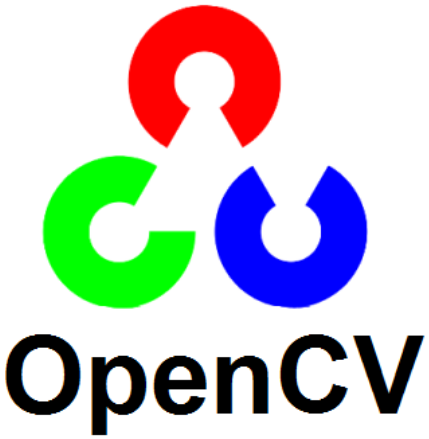
순번	구분	내용
1	운영체제	windows10
2	프로그래밍언어	Python3.4.3
3	설치라이브러리	Tensorflow, numpy, opencv, matplotlib

2.1 테스트 환경

순 번	제 목	내 용
1	운영체제	Ubuntu 16.04
2	프로그래밍언어	C++
3	설치라이브러리	OpenCV, YOLO

2.2 라이브러리 설치

- 2.2.1 OpenCV 라이브러리 설치



OpenCV란 오픈 소스 컴퓨터 비전 라이브러리 중 하나로 크로스플랫폼과 실시간 이미지 프로세싱에 중점을 둔 라이브러리이다. Windows, Linux, OS X, iOS, Android 등 다양한 플랫폼을 지원한다. 영상 관련 라이브러리로서 사실상 표준의 지위를 가지고 있다. OpenCV는 tensorFlow, Torch 및 Caffe의 딥러닝 프레임워크를 지원한다.

2.2 라이브러리 설치

- 기존 설치된 패키지 업그레이드

1. Ubuntu 저장소(repository)로 부터 패키지 리스트를 업데이트 한다.
   - sudo apt-get update

2. 기존에 설치된 패키지의 새로운 버전이 있으면 업그레이드를 진행한다.
   - sudo apt-get upgrade

2.2 라이브러리 설치

- OpenCV 컴파일 전 필요한 패키지 설치

1. OpenCV를 컴파일하는데 사용하는데 필요한 패키지들을 설치한다. build-essential 패키지에는 C/C++ 컴파일러와 관련 라이브러리, make 같은 도구들이 포함되어 있다.
   - sudo apt-get install build-essential cmake

2. pkg-config는 프로그램 컴파일 및 링크시 필요한 라이브러리에 대한 정보를 메타파일로부터 가져오는데 사용된다. 터미널에서 특정 라이브러리를 사용한 소스코드를 컴파일시 필요한 컴파일러 및 링커 플래그를 추가하는데 도움이 된다.
   - sudo apt-get install pkg-config

2.2 라이브러리 설치

- OpenCV 컴파일 전 필요한 패키지 설치

3. 특정 포맷의 이미지 파일을 불러오거나 기록하기 위해 필요한 패키지들을 설치한다.

- sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev

4. 특정 코덱의 비디오 파일을 읽어오거나 기록하기 위해 필요한 패키지들을 설치한다.

- sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libxvidcore-dev libx264-dev libxine2-dev

2.2 라이브러리 설치

- OpenCV 컴파일 전 필요한 패키지 설치

5. Video4Linux 패키지는 리눅스에서 실시간 비디오 캡처를 지원하기 위한 디바이스 드라이버와 API를 포함하고 있다.

- sudo apt-get install libv4l-dev v4l-utils

6. GStreamer는 비디오 스트리밍을 위한 라이브러리이다.

- sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev

2.2 라이브러리 설치

- OpenCV 컴파일 전 필요한 패키지 설치

7. OpenCV에서는 highgui 모듈을 사용하여 자체적으로 윈도우 생성하여 이미지나 비디오들을 보여줄 수 있다. 여기서는 qt4를 지정해준다. QImage와 Mat 간의 변환에는 영향을 주지 않는다.

- sudo apt-get install libqt4-dev

8. OpenGL 지원하기 위해 필요한 라이브러리입니다.

- sudo apt-get install mesa-utils libgl1-mesa-dri libqt4-opengl-dev

2.2 라이브러리 설치

- OpenCV 컴파일 전 필요한 패키지 설치

9. OpenCV 최적화를 위해 사용되는 라이브러리들을 설치한다.

- sudo apt-get install libatlas-base-dev gfortran libeigen3-dev

10. python2.7-dev와 python3-dev 패키지는 OpenCV-Python 바인딩을 위해 필요한 패키지들이다. Numpy는 매트릭스 연산등을 빠르게 처리할 수 있어서 OpenCV에서 사용된다.

- sudo apt-get install python2.7-dev python3-dev python-numpy python3-numpy

2.2 라이브러리 설치

- OpenCV 설정과 컴파일 및 설치

1. 소스 코드를 저장할 임시 디렉토리를 생성하여 이동 후 진행한다.

```
webnautes@webnautes-note:~$ mkdir opencv
webnautes@webnautes-note:~$ cd opencv
webnautes@webnautes-note:~/opencv$
```

2. OpenCV 3.4 소스코드를 다운로드 받아 압축을 풀어준다.

- wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.0.zip
- unzip opencv.zip

2.2 라이브러리 설치

- OpenCV 설정과 컴파일 및 설치

3. opencv_contrib(extra modules) 소스코드를 다운로드 받아 압축을 풀어준다.

- wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/3.4.0.zip
- unzip opencv_contrib.zip

4. 다음처럼 두 개의 디렉토리가 생성된다

```
webnautes@webnautes-note:~/opencv$ ls -d */
opencv-3.4.0/  opencv_contrib-3.4.0/
```

2.2 라이브러리 설치

- OpenCV 설정과 컴파일 및 설치

5. opencv-3.4.0 디렉토리로 이동하여 build 디렉토리를 생성하고 build 디렉토리로 이동한다.
컴파일은 build 디렉토리에서 이루어진다.

```
webnautes@webnautes-note:~/opencv$ cd opencv-3.4.0/
webnautes@webnautes-note:~/opencv/opencv-3.4.0$ mkdir build
webnautes@webnautes-note:~/opencv/opencv-3.4.0$ cd build
webnautes@webnautes-note:~/opencv/opencv-3.4.0/build$
```

2.2 라이브러리 설치

- OpenCV 설정과 컴파일 및 설치

6. cmake를 사용하여 OpenCV 컴파일 설정을 해준다.

- cmake -D CMAKE_BUILD_TYPE=RELEASE ₩ -D CMAKE_INSTALL_PREFIX=/usr/local ₩
-D WITH_TBB=OFF ₩ -D WITH_IPP=OFF ₩ -D WITH_1394=OFF ₩
-D BUILD_WITH_DEBUG_INFO=OFF ₩ -D BUILD_DOCS=OFF ₩
-D INSTALL_C_EXAMPLES=ON ₩ -D INSTALL_PYTHON_EXAMPLES=ON ₩
-D BUILD_EXAMPLES=OFF ₩ -D BUILD_TESTS=OFF ₩
-D BUILD_PERF_TESTS=OFF ₩ -D WITH_QT=ON ₩
-D WITH_OPENGL=ON ₩ -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-3.4.0/modules ₩
-D WITH_V4L=ON ₩ -D WITH_FFMPEG=ON ₩
-D WITH_XINE=ON ₩ -D BUILD_NEW_PYTHON_SUPPORT=ON ₩
../

2.2 라이브러리 설치

- OpenCV 설정과 컴파일 및 설치

  - 다음과 같은 메시지가 보이면 정상적으로 설정이 완료 된 것이다.

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/webnautes/opencv/opencv-3.4.0/build
```

2.2 라이브러리 설치

- OpenCV 설정과 컴파일 및 설치

7. 컴파일을 시작하기 전에 사용 중인 컴퓨터의 CPU 코어수를 확인합니다.

```
webnautes@webnautes-note:~/opencv$ cd opencv-3.4.0/
webnautes@webnautes-note:~/opencv/opencv-3.4.0$ mkdir build
webnautes@webnautes-note:~/opencv/opencv-3.4.0$ cd build
webnautes@webnautes-note:~/opencv/opencv-3.4.0/build$
```

8. make 명령을 사용하여 컴파일을 시작한다. -j 다음에 위에서 확인한 숫자를 붙여서 실행한다.

```
webnautes@webnautes-note:~/opencv/opencv-3.4.0/build$ make -j4
```

2.2 라이브러리 설치

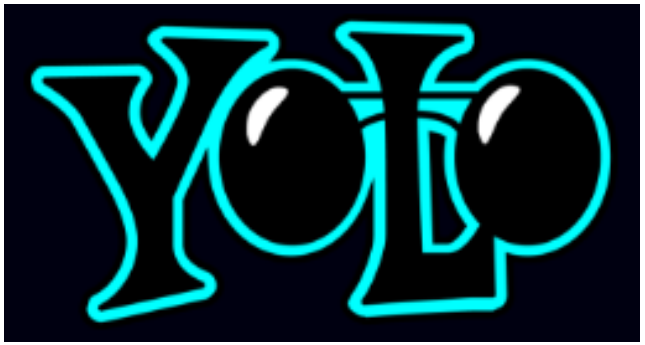
- OpenCV 설정과 컴파일 및 설치

9. 컴파일 결과물을 설치한다.

```
webnautes@webnautes-note:~/opencv/opencv-3.4.0/build$ sudo make install
```

2.2 라이브러리 설치

- 2.2.2 YOLO



YOLO는 실시간으로 물체를 탐지 하여 라벨링을 해준다. 각 이미지를 S x S 개의 그리드로 분할하고, 그리드의 신뢰도를 계산한다. 신뢰도는 그리드 내 객체 인식 시 정확성을 반영한다. 다음 그림과 같이 처음에는 객체 인식과는 동떨어진 경계 상자가 설정되지만, 신뢰도를 계산하여 경계 상자의 위치를 조정함으로써, 가장 높은 객체 인식 정확성을 가지는 경계 상자를 얻을 수 있다.

2.2 라이브러리 설치

- YOLO 설치

1. Darknet의 YOLO를 다운받는다.

- git clone https://github.com/pjreddie/darknet

2. 다운받은 폴더의 Makefile을 연다. OPENCV=1로 바꿔준다.

- nano Makefile

2.2 라이브러리 설치

- YOLO 설치

3. Make 하기

- cd darknet
- make

4. YOLO에서 제공하는 학습된 가중치(Weight) 설치

- wget https://pjreddie.com/media/files/yolo.weights

2.3 이미지 및 영상 분할 추적 예제

- 다음 예제는 YOLO를 이용하여 사진이나, 동영상파일, Webcam에서 실시간으로 사물을 경계상자를 통해 라벨링하는 예제이다.

2.3 이미지 및 영상 분할 추적 예제

- YOLO

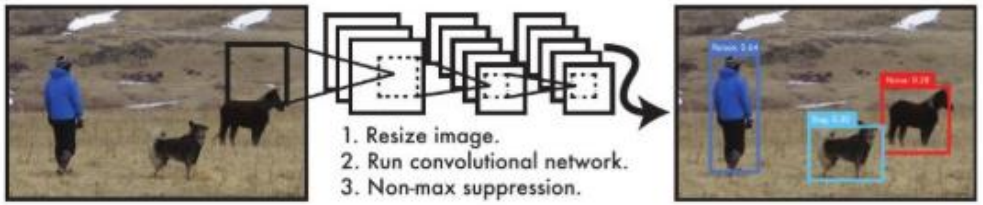


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO(You Only Look Once)는 이미지 내의 bounding box와 class probability를 single regression problem으로 간주하여, 이미지를 한 번 보는 것으로 object의 종류와 위치를 추측한다. 옆의 그림과 같이 single convolutional network를 통해 multiple bounding box에 대한 class probablility를 계산하는 방식이다.

2.3 이미지 및 영상 분할 추적 예제

- Unified Detection(통합 탐지)

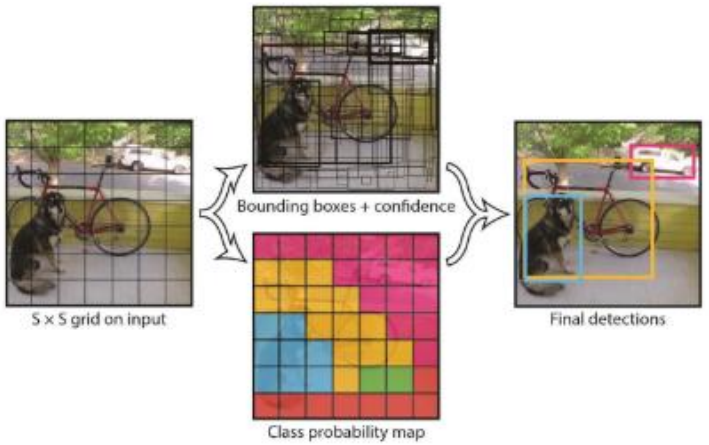


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

1. Input image를 S X S grid로 나눈다.

2. 각각의 grid cell은 B개의 bounding box와 각 bounding box에 대한 confidence score를 갖는다. (만약 cell에 object가 존재하지 않는다면 confidence score는 0이 된다.)

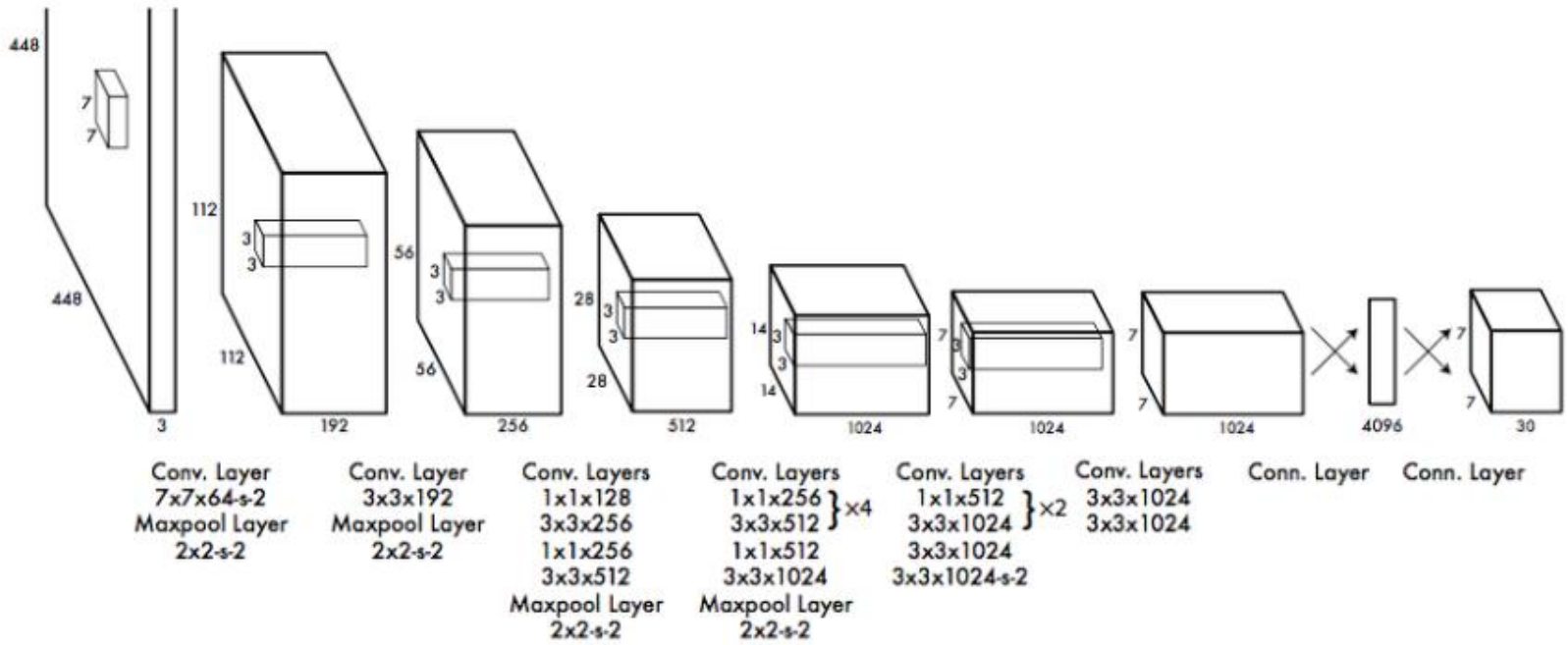
3. 각각의 grid cell은 C개의 conditional class probability를 갖는다.

4. 각각의 bounding box는 x, y, w, h, confidence로 구성된다.
(x,y): Bounding box의 중심점을 의미하며, grid cell의 범위에 대한 상대값이 입력된다.
(w,h): 전체 이미지의 width, height에 대한 상대값이 입력된다.

5. Test time에는 conditional class probability와 bounding box의 confidence score를 곱하여 class-specific confidence score를 얻는다.

2.3 이미지 및 영상 분할 추적 예제

- Network Design



2.3 이미지 및 영상 분할 추적 예제

- Training Process
  - Grid cell의 여러 bounding box들 중, ground-truth box와의 IOU가 가장 높은 bounding box를 predictor로 설정한다.

2.3 이미지 및 영상 분할 추적 예제

- Training Process
  - Loss Function

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \quad (1)$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \quad (2)$$

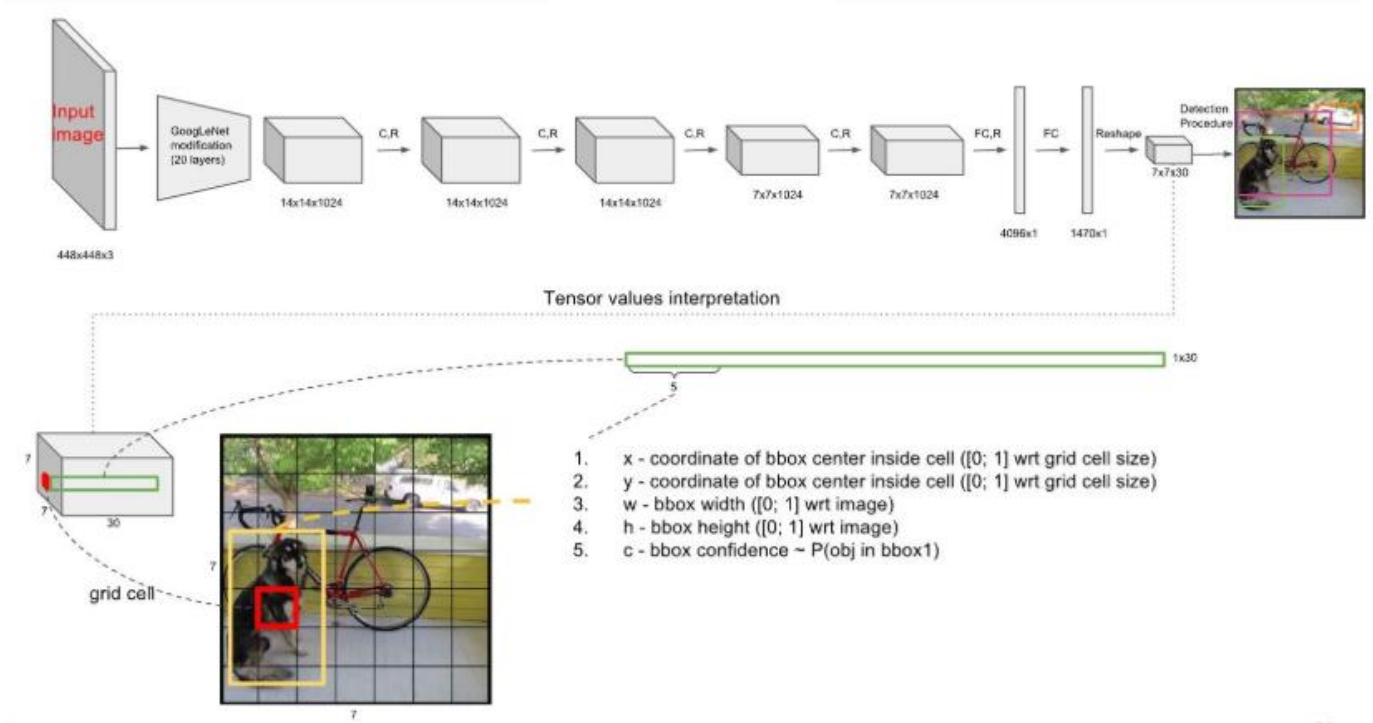
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \quad (3)$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \quad (4)$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

2.3 이미지 및 영상 분할 추적 예제

- Inference Process



2.3 이미지 및 영상 분할 추적 예제

- YOLO
  - 장점

  - 간단한 처리과정으로 속도가 매우 빠르다. 또한 기존의 다른 real-time detection system들과 비교할 때,2배 정도 높은 mAP(Mean Average Precision)를 보인다.

  - Image 전체를 한 번에 바라보는 방식으로 class에 대한 맥락적 이해도가 높다. 이로 인해 낮은 backgound error(False-Positive)를 보인다.

  - Object에 대한 좀 더 일반화된 특징을 학습한다. 가령 natural image로 학습하고 이를 artwork에 테스트 했을때, 다른 Detection System들에 비해 훨씬 높은 성능을 보여준다.

2.3 이미지 및 영상 분할 추적 예제

- YOLO
  - 단점

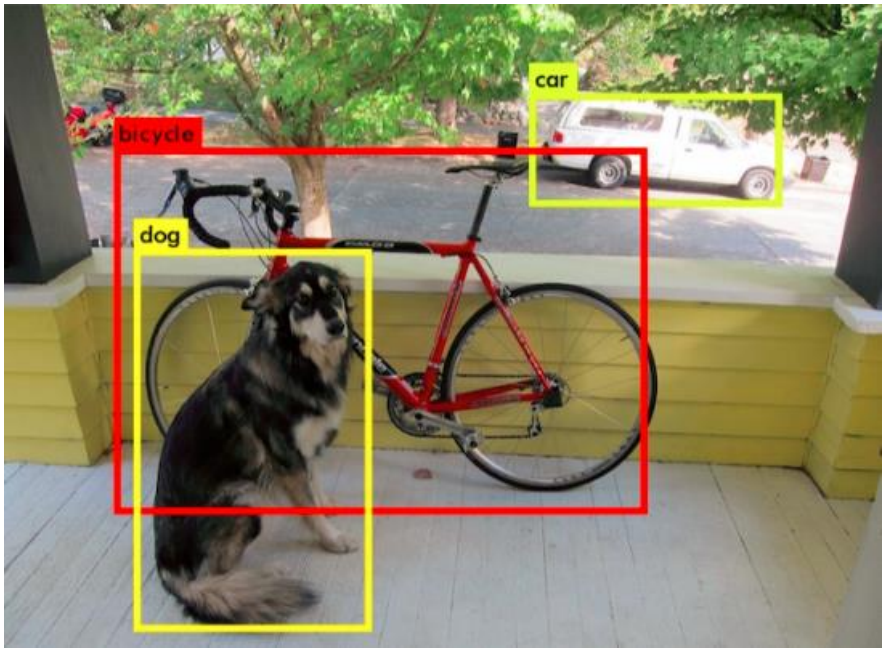
    - 각각의 grid cell이 하나의 클래스만을 에측할 수 있으므로, 작은 object 여러개가 붙어있으면 제대로 예측하지 못한다.

    - bounding box의 형태가 training data를 통해서만 학습되므로, 새로운/독특한 형태의 bounding box의 경우 정확히 예측하지 못한다.

    - 몇 단계의 layer를 거쳐서 나온 feature map을 대상으로 bounding box를 예측하므로 localization이 다소 부정확해지는 경우가 있다.

2.3 이미지 및 영상 분할 추적 예제

- 이미지 파일 예제 실행
  - ./darknet detect cfg/yolo.cfg yolo.weights data/dog.jpg



2.3 이미지 및 영상 분할 추적 예제

- Webcam을 이용한 실시간 물체 분할 추적
  - ./darknet detector demo cfg/coco.data cfg/yolo.cfg yolo.weights