[1] Keras Yolo V3 Object Detection

```
                        # conda 최신버전으로 업데이트
conda update conda
d:
md \yolov3
cd \yolov3
                        # yolov3 가상환경 생성 및 관련 패키지 설치
conda create -n yolov3 git keras=2.1.5 tensorflow=1.9 opencv matplotlib pillow python=3.6  #CPU버젼
conda create -n yolov3 git keras=2.1.5 tensorflow-gpu opencv matplotlib pillow python=3.6 #GPU버젼
                        # 가상환경으로 진입
conda activate yolov3
                        # yolo v3 source download
git clone https://github.com/qqwweee/keras-yolo3.git            ==> keras-yolo3 폴더 생성
cd keras-yolo3
# 브라우저로 https://pjreddie.com/darknet/yolo/ 접속하여 기 학습된 yolov3-416의 weights파일 다운로드하여
#                       keras-yolov3 폴더에 저장 (파일명:yolov3.weigts)
                        # Darknet config and weights 로 부터 Tensorflow를 사용하는 Keras model 생성
                                --> model_data 폴더 아래에 yolo.h5 모델 파일 생성
python convert.py yolov3.cfg yolov3.weights model_data/yolo.h5
                        # 하나의 이미지 파일에 대한 실행여부 확인
python yolo_video.py  --image
                        # 웹검색해서 그림파일 저장 후, 그림 파일명 입력하면 결과 BMP 파일 생성되어야 정상임
                        # yolo.py 파일을 사용하려면 소스 맨 마지막에 아래 내용 추가 후 저장
notepad yolo.py
```

```
def detect_img(yolo):
    while True:
        img = input('Input image filename:')
        try:
            image = Image.open(img)
        except:
            print('Open Error! Try again!')
            continue
        else:
            r_image = yolo.detect_image(image)
            print(type(r_image))
            import cv2
            cv2.imwrite("out.jpg", np.asarray(r_image)[..., ::-1])
            r_image.show()
    yolo.close_session()

if __name__ == '__main__':
    detect_img(YOLO())
```

```
python yolo.py
                        # 현재 폴더에 결과파일 out.jpg 생성됨
```

[2] Yolo v3 Tiny 모델 사용

(# GPU 없는 cpu 환경에 적합)

https://pjreddie.com/darknet/yolo/ 에서 YOLOv3-tiny weights 파일 다운로드
yolov3-tiny.weights 파일 현재 폴더에 저장

```
# tiny yolo3 모델로 변환
python convert.py yolov3-tiny.cfg yolov3-tiny.weights model_data/yolo-tiny.h5
# yolo.py를 복사하여 yolo_tiny.py 생성
copy yolo.py yolo_tiny.py
```

yolo_tiny.py 파일에서 class YOLO(object) 부분에서 아래 내용으로 수정

```
self.model_path = 'model_data/yolo-tiny.h5' # model path or trained weights path
self.anchors_path = 'model_data/tiny_yolo_anchors.txt'
```

```
# tiny model 실행
python yolo_tiny.py
```

[3] Yolo v3 웹캠 사용

아래 내용으로 yolo_ cam.py 파일 생성

```
"""
Run a YOLO_v3 style detection model on test images.
"""

import colorsys
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from timeit import default_timer as timer

import numpy as np
from keras import backend as K
from keras.models import load_model
from keras.layers import Input
from PIL import Image, ImageFont, ImageDraw

from yolo3.model import yolo_eval, yolo_body, tiny_yolo_body
from yolo3.utils import letterbox_image
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
from keras.utils import multi_gpu_model
gpu_num=1

#- Added
import cv2
cap = cv2.VideoCapture(0)
camera_scale = 1.
#-
```

```
class YOLO(object):
    def __init__(self):
        self.model_path = 'model_data/yolo.h5' # model path or trained weights path
        self.anchors_path = 'model_data/yolo_anchors.txt'
        self.classes_path = 'model_data/coco_classes.txt'
        self.score = 0.3
        self.iou = 0.45
        self.class_names = self._get_class()
        self.anchors = self._get_anchors()
        self.sess = K.get_session()
        self.model_image_size = (416, 416) # fixed size or (None, None), hw
        self.boxes, self.scores, self.classes = self.generate()

    def _get_class(self):
        classes_path = os.path.expanduser(self.classes_path)
        with open(classes_path) as f:
            class_names = f.readlines()
        class_names = [c.strip() for c in class_names]
        return class_names

    def _get_anchors(self):
        anchors_path = os.path.expanduser(self.anchors_path)
        with open(anchors_path) as f:
            anchors = f.readline()
        anchors = [float(x) for x in anchors.split(',')]
        return np.array(anchors).reshape(-1, 2)

    def generate(self):
        model_path = os.path.expanduser(self.model_path)
        assert model_path.endswith('.h5'), 'Keras model or weights must be a .h5 file.'

        # Load model, or construct model and load weights.
        num_anchors = len(self.anchors)
        num_classes = len(self.class_names)
        is_tiny_version = num_anchors==6 # default setting
        try:
            self.yolo_model = load_model(model_path, compile=False)
        except:
            self.yolo_model = tiny_yolo_body(Input(shape=(None,None,3)), num_anchors//2, num_classes) \
                if is_tiny_version else yolo_body(Input(shape=(None,None,3)), num_anchors//3, num_classes)
            self.yolo_model.load_weights(self.model_path) # make sure model, anchors and classes match
        else:
            assert self.yolo_model.layers[-1].output_shape[-1] == \
```

```
            num_anchors/len(self.yolo_model.output) * (num_classes + 5), \
            'Mismatch between model and given anchor and class sizes'

        print('{} model, anchors, and classes loaded.'.format(model_path))

        # Generate colors for drawing bounding boxes.
        hsv_tuples = [(x / len(self.class_names), 1., 1.)
                      for x in range(len(self.class_names))]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
        self.colors = list(
            map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)),
                self.colors))
        np.random.seed(10101)  # Fixed seed for consistent colors across runs.
        np.random.shuffle(self.colors)  # Shuffle colors to decorrelate adjacent classes.
        np.random.seed(None)  # Reset seed to default.

        # Generate output tensor targets for filtered bounding boxes.
        self.input_image_shape = K.placeholder(shape=(2, ))
        if gpu_num>=2:
            self.yolo_model = multi_gpu_model(self.yolo_model, gpus=gpu_num)
        boxes, scores, classes = yolo_eval(self.yolo_model.output, self.anchors,
                len(self.class_names), self.input_image_shape,
                score_threshold=self.score, iou_threshold=self.iou)
        return boxes, scores, classes

    def detect_image(self, image):
        start = timer()

        if self.model_image_size != (None, None):
            assert self.model_image_size[0]%32 == 0, 'Multiples of 32 required'
            assert self.model_image_size[1]%32 == 0, 'Multiples of 32 required'
            boxed_image = letterbox_image(image, tuple(reversed(self.model_image_size)))
        else:
            new_image_size = (image.width - (image.width % 32),
                              image.height - (image.height % 32))
            boxed_image = letterbox_image(image, new_image_size)
        image_data = np.array(boxed_image, dtype='float32')

        print(image_data.shape)
        image_data /= 255.
        image_data = np.expand_dims(image_data, 0)  # Add batch dimension.

        out_boxes, out_scores, out_classes = self.sess.run(
            [self.boxes, self.scores, self.classes],
            feed_dict={
                self.yolo_model.input: image_data,
```

```
            self.input_image_shape: [image.size[1], image.size[0]],
            K.learning_phase(): 0
        })

    print('Found {} boxes for {}'.format(len(out_boxes), 'img'))

    font = ImageFont.truetype(font='font/FiraMono-Medium.otf',
                size=np.floor(3e-2 * image.size[1] + 0.5).astype('int32'))
    thickness = (image.size[0] + image.size[1]) // 300

    for i, c in reversed(list(enumerate(out_classes))):
        predicted_class = self.class_names[c]
        box = out_boxes[i]
        score = out_scores[i]

        label = '{} {:.2f}'.format(predicted_class, score)
        draw = ImageDraw.Draw(image)
        label_size = draw.textsize(label, font)

        top, left, bottom, right = box
        top = max(0, np.floor(top + 0.5).astype('int32'))
        left = max(0, np.floor(left + 0.5).astype('int32'))
        bottom = min(image.size[1], np.floor(bottom + 0.5).astype('int32'))
        right = min(image.size[0], np.floor(right + 0.5).astype('int32'))
        print(label, (left, top), (right, bottom))

        if top - label_size[1] >= 0:
            text_origin = np.array([left, top - label_size[1]])
        else:
            text_origin = np.array([left, top + 1])

        # My kingdom for a good redistributable image drawing library.
        for i in range(thickness):
            draw.rectangle(
                [left + i, top + i, right - i, bottom - i],
                outline=self.colors[c])
        draw.rectangle(
            [tuple(text_origin), tuple(text_origin + label_size)],
            fill=self.colors[c])
        draw.text(text_origin, label, fill=(0, 0, 0), font=font)
        del draw

    end = timer()
    print(end - start)
    return image
```

```
    def close_session(self):
        self.sess.close()


def detect_video(yolo, video_path, output_path=""):
    import cv2
    vid = cv2.VideoCapture(video_path)
    if not vid.isOpened():
        raise IOError("Couldn't open webcam or video")
    video_FourCC    = int(vid.get(cv2.CAP_PROP_FOURCC))
    video_fps       = vid.get(cv2.CAP_PROP_FPS)
    video_size      = (int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)),
                       int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    isOutput = True if output_path != "" else False
    if isOutput:
        print("!!! TYPE:", type(output_path), type(video_FourCC), type(video_fps), type(video_size))
        out = cv2.VideoWriter(output_path, video_FourCC, video_fps, video_size)
    accum_time = 0
    curr_fps = 0
    fps = "FPS: ??"
    prev_time = timer()
    while True:
        return_value, frame = vid.read()
        image = Image.fromarray(frame)
        image = yolo.detect_image(image)
        result = np.asarray(image)
        curr_time = timer()
        exec_time = curr_time - prev_time
        prev_time = curr_time
        accum_time = accum_time + exec_time
        curr_fps = curr_fps + 1
        if accum_time > 1:
            accum_time = accum_time - 1
            fps = "FPS: " + str(curr_fps)
            curr_fps = 0
        cv2.putText(result, text=fps, org=(3, 15), fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                    fontScale=0.50, color=(255, 0, 0), thickness=2)
        cv2.namedWindow("result", cv2.WINDOW_NORMAL)
        cv2.imshow("result", result)
        if isOutput:
            out.write(result)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    yolo.close_session()
```

```
def detect_img(yolo):
    while True:
        ret, image = cap.read()
        if cv2.waitKey(10) == 27:
            break
        h, w = image.shape[:2]
        rh = int(h * camera_scale)
        rw = int(w * camera_scale)
        image = cv2.resize(image, (rw, rh))
        image = image[:,:,(2,1,0)]
        image = Image.fromarray(image)
        r_image = yolo.detect_image(image)
        out_img = np.array(r_image)[:,:,(2,1,0)]
        cv2.imshow("YOLOv2", np.array(out_img))
        #cv2.waitKey(0)
    yolo.close_session()



if __name__ == '__main__':
    detect_img(YOLO())
```

[4] Yolo v3 Tiny 모델 웹캠 사용

아래 내용으로 yolo_tiny_cam.py 파일 생성

```
"""
Run a YOLO_v3 style detection model on test images.
"""


import colorsys
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from timeit import default_timer as timer

import numpy as np
from keras import backend as K
from keras.models import load_model
from keras.layers import Input
from PIL import Image, ImageFont, ImageDraw

from yolo3.model import yolo_eval, yolo_body, tiny_yolo_body
from yolo3.utils import letterbox_image
import os
```

```
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
from keras.utils import multi_gpu_model
gpu_num=1

#- Added
import cv2
cap = cv2.VideoCapture(0)
camera_scale = 1.
#-

class YOLO(object):
    def __init__(self):
        self.model_path = 'model_data/yolo-tiny.h5' # model path or trained weights path
        self.anchors_path = 'model_data/tiny_yolo_anchors.txt'
        self.classes_path = 'model_data/coco_classes.txt'
        self.score = 0.3
        self.iou = 0.45
        self.class_names = self._get_class()
        self.anchors = self._get_anchors()
        self.sess = K.get_session()
        self.model_image_size = (416, 416) # fixed size or (None, None), hw
        self.boxes, self.scores, self.classes = self.generate()

    def _get_class(self):
        classes_path = os.path.expanduser(self.classes_path)
        with open(classes_path) as f:
            class_names = f.readlines()
        class_names = [c.strip() for c in class_names]
        return class_names

    def _get_anchors(self):
        anchors_path = os.path.expanduser(self.anchors_path)
        with open(anchors_path) as f:
            anchors = f.readline()
        anchors = [float(x) for x in anchors.split(',')]
        return np.array(anchors).reshape(-1, 2)

    def generate(self):
        model_path = os.path.expanduser(self.model_path)
        assert model_path.endswith('.h5'), 'Keras model or weights must be a .h5 file.'

        # Load model, or construct model and load weights.
        num_anchors = len(self.anchors)
        num_classes = len(self.class_names)
        is_tiny_version = num_anchors==6 # default setting
        try:
```

```
            self.yolo_model = load_model(model_path, compile=False)
        except:
            self.yolo_model = tiny_yolo_body(Input(shape=(None,None,3)), num_anchors//2,
num_classes) \
                if is_tiny_version else yolo_body(Input(shape=(None,None,3)), num_anchors//3,
num_classes)
            self.yolo_model.load_weights(self.model_path) # make sure model, anchors and classes
match
        else:
            assert self.yolo_model.layers[-1].output_shape[-1] == \
                num_anchors/len(self.yolo_model.output) * (num_classes + 5), \
                'Mismatch between model and given anchor and class sizes'

        print('{} model, anchors, and classes loaded.'.format(model_path))

        # Generate colors for drawing bounding boxes.
        hsv_tuples = [(x / len(self.class_names), 1., 1.)
                      for x in range(len(self.class_names))]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
        self.colors = list(
            map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)),
                self.colors))
        np.random.seed(10101)  # Fixed seed for consistent colors across runs.
        np.random.shuffle(self.colors)  # Shuffle colors to decorrelate adjacent classes.
        np.random.seed(None)  # Reset seed to default.

        # Generate output tensor targets for filtered bounding boxes.
        self.input_image_shape = K.placeholder(shape=(2, ))
        if gpu_num>=2:
            self.yolo_model = multi_gpu_model(self.yolo_model, gpus=gpu_num)
        boxes, scores, classes = yolo_eval(self.yolo_model.output, self.anchors,
                len(self.class_names), self.input_image_shape,
                score_threshold=self.score, iou_threshold=self.iou)
        return boxes, scores, classes

    def detect_image(self, image):
        start = timer()

        if self.model_image_size != (None, None):
            assert self.model_image_size[0]%32 == 0, 'Multiples of 32 required'
            assert self.model_image_size[1]%32 == 0, 'Multiples of 32 required'
            boxed_image = letterbox_image(image, tuple(reversed(self.model_image_size)))
        else:
            new_image_size = (image.width - (image.width % 32),
                              image.height - (image.height % 32))
            boxed_image = letterbox_image(image, new_image_size)
```

```
        image_data = np.array(boxed_image, dtype='float32')

        print(image_data.shape)
        image_data /= 255.
        image_data = np.expand_dims(image_data, 0)  # Add batch dimension.

        out_boxes, out_scores, out_classes = self.sess.run(
            [self.boxes, self.scores, self.classes],
            feed_dict={
                self.yolo_model.input: image_data,
                self.input_image_shape: [image.size[1], image.size[0]],
                K.learning_phase(): 0
            })

        print('Found {} boxes for {}'.format(len(out_boxes), 'img'))

        font = ImageFont.truetype(font='font/FiraMono-Medium.otf',
                    size=np.floor(3e-2 * image.size[1] + 0.5).astype('int32'))
        thickness = (image.size[0] + image.size[1]) // 300

        for i, c in reversed(list(enumerate(out_classes))):
            predicted_class = self.class_names[c]
            box = out_boxes[i]
            score = out_scores[i]

            label = '{} {:.2f}'.format(predicted_class, score)
            draw = ImageDraw.Draw(image)
            label_size = draw.textsize(label, font)

            top, left, bottom, right = box
            top = max(0, np.floor(top + 0.5).astype('int32'))
            left = max(0, np.floor(left + 0.5).astype('int32'))
            bottom = min(image.size[1], np.floor(bottom + 0.5).astype('int32'))
            right = min(image.size[0], np.floor(right + 0.5).astype('int32'))
            print(label, (left, top), (right, bottom))

            if top - label_size[1] >= 0:
                text_origin = np.array([left, top - label_size[1]])
            else:
                text_origin = np.array([left, top + 1])

            # My kingdom for a good redistributable image drawing library.
            for i in range(thickness):
                draw.rectangle(
                    [left + i, top + i, right - i, bottom - i],
                    outline=self.colors[c])
```

```
                draw.rectangle(
                    [tuple(text_origin), tuple(text_origin + label_size)],
                    fill=self.colors[c])
                draw.text(text_origin, label, fill=(0, 0, 0), font=font)
                del draw

        end = timer()
        print(end - start)
        return image

    def close_session(self):
        self.sess.close()


def detect_video(yolo, video_path, output_path=""):
    import cv2
    vid = cv2.VideoCapture(video_path)
    if not vid.isOpened():
        raise IOError("Couldn't open webcam or video")
    video_FourCC    = int(vid.get(cv2.CAP_PROP_FOURCC))
    video_fps       = vid.get(cv2.CAP_PROP_FPS)
    video_size      = (int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)),
                       int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    isOutput = True if output_path != "" else False
    if isOutput:
        print("!!! TYPE:", type(output_path), type(video_FourCC), type(video_fps), type(video_size))
        out = cv2.VideoWriter(output_path, video_FourCC, video_fps, video_size)
    accum_time = 0
    curr_fps = 0
    fps = "FPS: ??"
    prev_time = timer()
    while True:
        return_value, frame = vid.read()
        image = Image.fromarray(frame)
        image = yolo.detect_image(image)
        result = np.asarray(image)
        curr_time = timer()
        exec_time = curr_time - prev_time
        prev_time = curr_time
        accum_time = accum_time + exec_time
        curr_fps = curr_fps + 1
        if accum_time > 1:
            accum_time = accum_time - 1
            fps = "FPS: " + str(curr_fps)
            curr_fps = 0
        cv2.putText(result, text=fps, org=(3, 15), fontFace=cv2.FONT_HERSHEY_SIMPLEX,
```

```
                        fontScale=0.50, color=(255, 0, 0), thickness=2)
            cv2.namedWindow("result", cv2.WINDOW_NORMAL)
            cv2.imshow("result", result)
            if isOutput:
                out.write(result)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        yolo.close_session()


def detect_img(yolo):
        while True:
            ret, image = cap.read()
            if cv2.waitKey(10) == 27:
                break
            h, w = image.shape[:2]
            rh = int(h * camera_scale)
            rw = int(w * camera_scale)
            image = cv2.resize(image, (rw, rh))
            image = image[:,:,(2,1,0)]
            image = Image.fromarray(image)
            r_image = yolo.detect_image(image)
            out_img = np.array(r_image)[:,:,(2,1,0)]
            cv2.imshow("YOLOv2", np.array(out_img))
            #cv2.waitKey(0)
        yolo.close_session()



if __name__ == '__main__':
        detect_img(YOLO())
```