TensorFlow_2.x_YOLOv3

2023.02.01

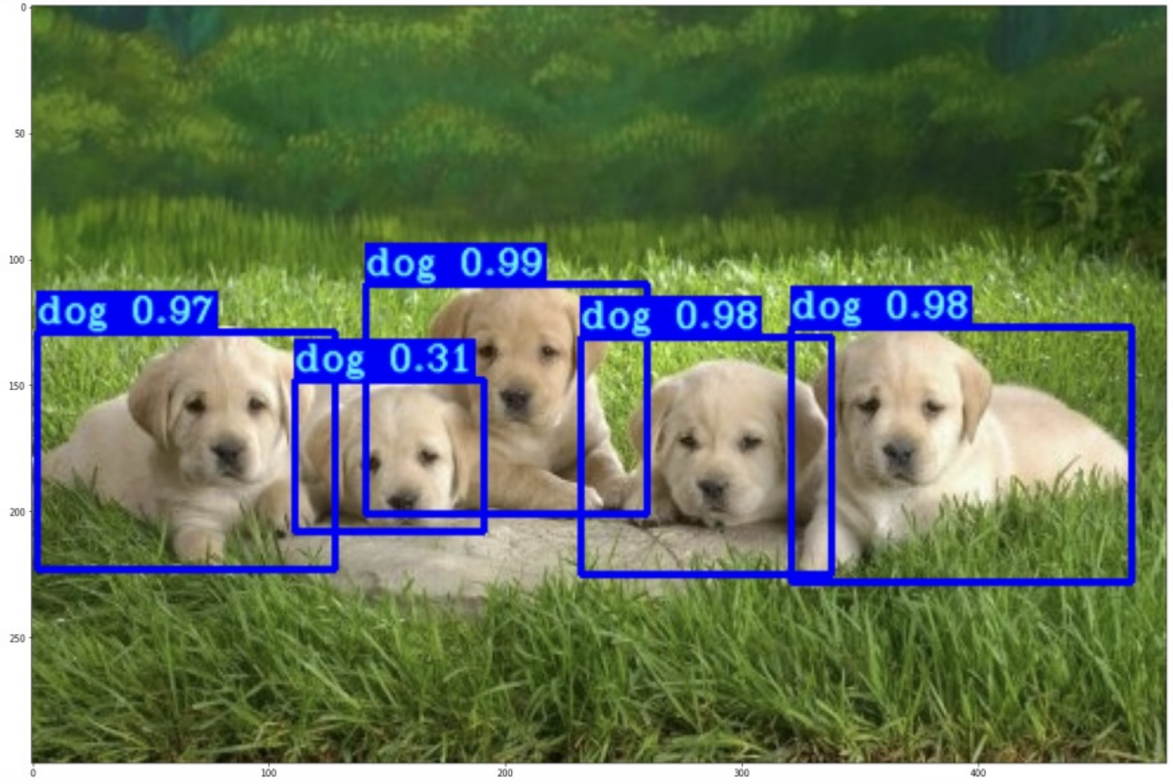
TensorFlow_2.x_YOLOv3

- detection_demo.py
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
    - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
#from yolov3.utils import detect_image, Load_Yolo_model
#from yolov3.configs import *


from google.colab.patches import cv2_imshow

yolo = Load_Yolo_model()

image_path = "./IMAGES/dogs.jpg"
output_path = "./IMAGES/dogs_pred.jpg"
image = detect_image(yolo, image_path, output_path, input_size=416, show=True, rectangle_colors=(255,0,0))

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(30,15))
plt.imshow(image)
```

```
GPUs [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
Loading Darknet_weights from: model_data/yolov3.weights
1/1 [==============================] - 13s 13s/step
<matplotlib.image.AxesImage at 0x7f635941b1f0>
```



- detection_demo.py v
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
      - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
def Load_Yolo_model():
    gpus = tf.config.experimental.list_physical_devices('GPU')
    if len(gpus) > 0:
        print(f'GPUs {gpus}')
        try: tf.config.experimental.set_memory_growth(gpus[0], True)
        except RuntimeError: pass

    if YOLO_FRAMEWORK == "tf": # TensorFlow detection
        if YOLO_TYPE == "yolov4":
            Darknet_weights = YOLO_V4_TINY_WEIGHTS if TRAIN_YOLO_TINY else YOLO_V4_WEIGHTS
        if YOLO_TYPE == "yolov3":
            YOLO_V3_WEIGHTS = "model_data/yolov3.weights"
            Darknet_weights = YOLO_V3_TINY_WEIGHTS if TRAIN_YOLO_TINY else YOLO_V3_WEIGHTS

        if YOLO_CUSTOM_WEIGHTS == False:
            print("Loading Darknet_weights from:", Darknet_weights)
            yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE, CLASSES=YOLO_COCO_CLASSES)
            load_yolo_weights(yolo, Darknet_weights) # use Darknet weights
        else:
            checkpoint = f"./checkpoints/{TRAIN_MODEL_NAME}"
            if TRAIN_YOLO_TINY:
                checkpoint += "_Tiny"
            print("Loading custom weights from:", checkpoint)
            yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE, CLASSES=TRAIN_CLASSES)
            yolo.load_weights(checkpoint)  # use custom weights

    elif YOLO_FRAMEWORK == "trt": # TensorRT detection
        saved_model_loaded = tf.saved_model.load(YOLO_CUSTOM_WEIGHTS, tags=[tag_constants.SERVING])
        signature_keys = list(saved_model_loaded.signatures.keys())
        yolo = saved_model_loaded.signatures['serving_default']

    return yolo
```

- detection_demo.py
  - ⇒ Load_Yolo_model() v
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
    - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
def Create_Yolo(input_size=416, channels=3, training=False, CLASSES=YOLO_COCO_CLASSES):
    NUM_CLASS = len(read_class_names(CLASSES))
    input_layer  = Input([input_size, input_size, channels])

    if TRAIN_YOLO_TINY:
        if YOLO_TYPE == "yolov4":
            conv_tensors = YOLOv4_tiny(input_layer, NUM_CLASS)
        if YOLO_TYPE == "yolov3":
            conv_tensors = YOLOv3_tiny(input_layer, NUM_CLASS)
    else:
        if YOLO_TYPE == "yolov4":
            conv_tensors = YOLOv4(input_layer, NUM_CLASS)
        if YOLO_TYPE == "yolov3":
            conv_tensors = YOLOv3(input_layer, NUM_CLASS)

    output_tensors = []
    for i, conv_tensor in enumerate(conv_tensors):
        pred_tensor = decode(conv_tensor, NUM_CLASS, i)
        if training: output_tensors.append(conv_tensor)
        output_tensors.append(pred_tensor)

    Yolo = tf.keras.Model(input_layer, output_tensors)
    return Yolo
```

- detection_demo.py
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo() v
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
      - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
def YOLOv3(input_layer, NUM_CLASS):
    # After the input layer enters the Darknet-53 network, we get three branches
    route_1, route_2, conv = darknet53(input_layer)
    # See the orange module (DBL) in the figure above, a total of 5 Subconvolution operation
    conv = convolutional(conv, (1, 1, 1024,  512))
    conv = convolutional(conv, (3, 3,  512, 1024))
    conv = convolutional(conv, (1, 1, 1024,  512))
    conv = convolutional(conv, (3, 3,  512, 1024))
    conv = convolutional(conv, (1, 1, 1024,  512))
    conv_lobj_branch = convolutional(conv, (3, 3, 512, 1024))

    # conv_lbbox is used to predict large-sized objects , Shape = [None, 13, 13, 255]
    conv_lbbox = convolutional(conv_lobj_branch, (1, 1, 1024, 3*(NUM_CLASS + 5)), activate=False, bn=False)

    conv = convolutional(conv, (1, 1,  512,  256))
    # upsample here uses the nearest neighbor interpolation method, which has the advantage that the
    # upsampling process does not need to learn, thereby reducing the network parameter
    conv = upsample(conv)

    conv = tf.concat([conv, route_2], axis=-1)
    conv = convolutional(conv, (1, 1, 768, 256))
    conv = convolutional(conv, (3, 3, 256, 512))
    conv = convolutional(conv, (1, 1, 512, 256))
    conv = convolutional(conv, (3, 3, 256, 512))
    conv = convolutional(conv, (1, 1, 512, 256))
    conv_mobj_branch = convolutional(conv, (3, 3, 256, 512))

    # conv_mbbox is used to predict medium-sized objects, shape = [None, 26, 26, 255]
    conv_mbbox = convolutional(conv_mobj_branch, (1, 1, 512, 3*(NUM_CLASS + 5)), activate=False, bn=False)

    conv = convolutional(conv, (1, 1, 256, 128))
    conv = upsample(conv)

    conv = tf.concat([conv, route_1], axis=-1)
    conv = convolutional(conv, (1, 1, 384, 128))
    conv = convolutional(conv, (3, 3, 128, 256))
    conv = convolutional(conv, (1, 1, 256, 128))
    conv = convolutional(conv, (3, 3, 128, 256))
    conv = convolutional(conv, (1, 1, 256, 128))
    conv_sobj_branch = convolutional(conv, (3, 3, 128, 256))

    # conv_sbbox is used to predict small size objects, shape = [None, 52, 52, 255]
    conv_sbbox = convolutional(conv_sobj_branch, (1, 1, 256, 3*(NUM_CLASS +5)), activate=False, bn=False)

    return [conv_sbbox, conv_mbbox, conv_lbbox]
```

- detection_demo.py
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3() v
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
      - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
def decode(conv_output, NUM_CLASS, i=0):
    # where i = 0, 1 or 2 to correspond to the three grid scales
    conv_shape       = tf.shape(conv_output)
    batch_size       = conv_shape[0]
    output_size      = conv_shape[1]

    conv_output = tf.reshape(conv_output, (batch_size, output_size, output_size, 3, 5 + NUM_CLASS))

    #conv_raw_dxdy = conv_output[:, :, :, :, 0:2] # offset of center position
    #conv_raw_dwdh = conv_output[:, :, :, :, 2:4] # Prediction box length and width offset
    #conv_raw_conf = conv_output[:, :, :, :, 4:5] # confidence of the prediction box
    #conv_raw_prob = conv_output[:, :, :, :, 5: ] # category probability of the prediction box
    conv_raw_dxdy, conv_raw_dwdh, conv_raw_conf, conv_raw_prob = tf.split(conv_output, (2, 2, 1, NUM_CLASS), axis=-1)

    # next need Draw the grid. Where output_size is equal to 13, 26 or 52
    #y = tf.range(output_size, dtype=tf.int32)
    #y = tf.expand_dims(y, -1)
    #y = tf.tile(y, [1, output_size])
    #x = tf.range(output_size,dtype=tf.int32)
    #x = tf.expand_dims(x, 0)
    #x = tf.tile(x, [output_size, 1])
    xy_grid = tf.meshgrid(tf.range(output_size), tf.range(output_size))
    xy_grid = tf.expand_dims(tf.stack(xy_grid, axis=-1), axis=2)  # [gx, gy, 1, 2]
    xy_grid = tf.tile(tf.expand_dims(xy_grid, axis=0), [batch_size, 1, 1, 3, 1])
    xy_grid = tf.cast(xy_grid, tf.float32)

    #xy_grid = tf.concat([x[:, :, tf.newaxis], y[:, :, tf.newaxis]], axis=-1)
    #xy_grid = tf.tile(xy_grid[tf.newaxis, :, :, tf.newaxis, :], [batch_size, 1, 1, 3, 1])
    #y_grid = tf.cast(xy_grid, tf.float32)

    YOLO_STRIDES = [8, 16, 32]
    YOLO_ANCHORS = [[[10, 13], [16, 30], [33, 23]], #9개의 Anchors
                    [[30, 61], [62, 45], [59, 119]],
                    [[116, 90], [156, 198], [373, 326]]]

    STRIDES = np.array(YOLO_STRIDES)
    ANCHORS = (np.array(YOLO_ANCHORS).T/STRIDES).T

    # Calculate the center position of the prediction box:
    pred_xy = (tf.sigmoid(conv_raw_dxdy) + xy_grid) * STRIDES[i]
    # Calculate the length and width of the prediction box:
    pred_wh = (tf.exp(conv_raw_dwdh) * ANCHORS[i]) * STRIDES[i]

    pred_xywh = tf.concat([pred_xy, pred_wh], axis=-1)
    pred_conf = tf.sigmoid(conv_raw_conf) # object box calculates the predicted confidence
    pred_prob = tf.sigmoid(conv_raw_prob) # calculating the predicted probability category box object

    # calculating the predicted probability category box object
    return tf.concat([pred_xywh, pred_conf, pred_prob], axis=-1)
```

- detection_demo.py
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode() v
      - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
def detect_image(Yolo, image_path, output_path, input_size=416, show=False, CLASSES=YOLO_COCO_CLASSES, score_threshold=0.3, iou_threshold=0.45, rectangle_colors=''):
    original_image      = cv2.imread(image_path)
    original_image      = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
    original_image      = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

    image_data = image_preprocess(np.copy(original_image), [input_size, input_size])
    image_data = image_data[np.newaxis, ...].astype(np.float32)

    if YOLO_FRAMEWORK == "tf":
        pred_bbox = Yolo.predict(image_data)


    pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1])) for x in pred_bbox]
    pred_bbox = tf.concat(pred_bbox, axis=0)

    bboxes = postprocess_boxes(pred_bbox, original_image, input_size, score_threshold)
    bboxes = nms(bboxes, iou_threshold, method='nms')

    image = draw_bbox(original_image, bboxes, CLASSES=CLASSES, rectangle_colors=rectangle_colors)
    # CreateXMLfile("XML_Detections", str(int(time.time())), original_image, bboxes, read_class_names(CLASSES))

    if output_path != '': cv2.imwrite(output_path, image)
    if show:
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return image
```

- detection_demo.py
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
      - ⇒ load_yolo_weights()
  - ⇒ detect_image() v
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms()
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

TensorFlow_2.x_YOLOv3

```python
def nms(bboxes, iou_threshold, sigma=0.3, method='nms'):

    classes_in_img = list(set(bboxes[:, 5]))
    best_bboxes = []

    for cls in classes_in_img:
        cls_mask = (bboxes[:, 5] == cls)
        cls_bboxes = bboxes[cls_mask]
        # Process 1: Determine whether the number of bounding boxes is greater than 0
        while len(cls_bboxes) > 0:
            # Process 2: Select the bounding box with the highest score according to socre order A
            max_ind = np.argmax(cls_bboxes[:, 4])
            best_bbox = cls_bboxes[max_ind]
            best_bboxes.append(best_bbox)
            cls_bboxes = np.concatenate([cls_bboxes[: max_ind], cls_bboxes[max_ind + 1:]])
            # Process 3: Calculate this bounding box A and
            # Remain all iou of the bounding box and remove those bounding boxes whose iou value is higher than the threshold
            iou = bboxes_iou(best_bbox[np.newaxis, :4], cls_bboxes[:, :4])
            weight = np.ones((len(iou),), dtype=np.float32)

            assert method in ['nms', 'soft-nms']

            if method == 'nms':
                iou_mask = iou > iou_threshold
                weight[iou_mask] = 0.0

            if method == 'soft-nms':
                weight = np.exp(-(1.0 * iou ** 2 / sigma))

            cls_bboxes[:, 4] = cls_bboxes[:, 4] * weight
            score_mask = cls_bboxes[:, 4] > 0.
            cls_bboxes = cls_bboxes[score_mask]

    return best_bboxes
```

- detection_demo.py
  - ⇒ Load_Yolo_model()
    - ⇒ Create_Yolo()
      - ⇒ YOLOv3()
        - ⇒ darknet53()
          - ⇒ convolutional()
          - ⇒ residual_block()
        - ⇒ decode()
      - ⇒ load_yolo_weights()
  - ⇒ detect_image()
    - ⇒ Image_preprocess()
    - ⇒ postprocess_boxes()
    - ⇒ nms() v
      - ⇒ YOLOv3()
    - ⇒ draw_bbox()

감사합니다 :)

2023.02.01