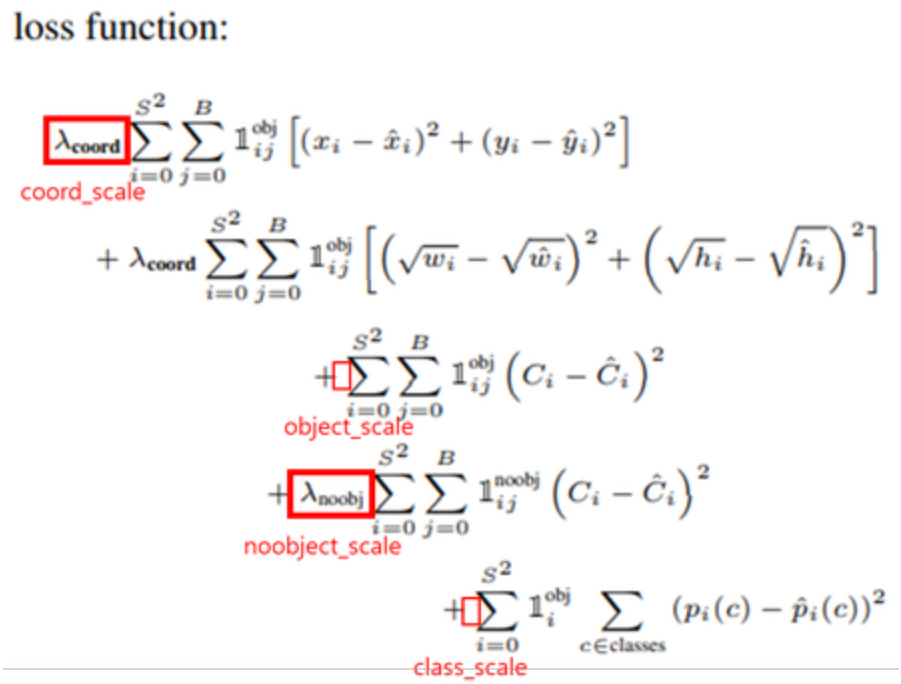
섹션 7. TensorFlow 2.0을 이용한 YOLO 논문 구현 1_Loss

1. 딥러닝 논문 구현방법 개요

딥러닝 논문 구현 프로젝트 파일 구조

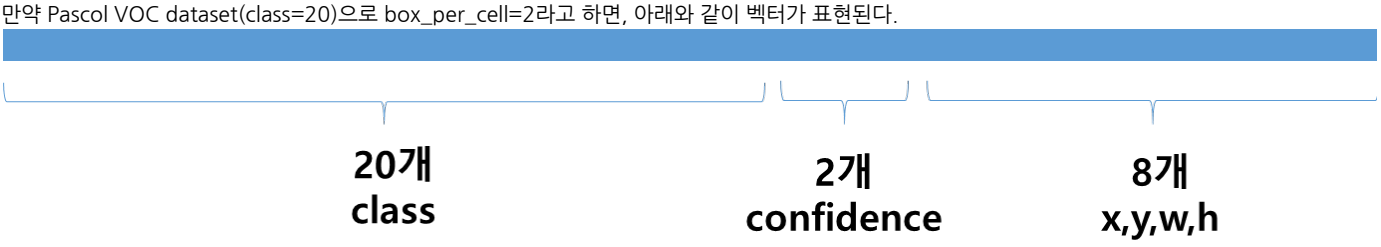
- 특정 기능을 담당하는 파이썬 스크립트를 나눠서 구현
- train.py : 모델 class를 인스턴스로 선언하고 For-loop을 돌면서 gradient descent를 수행하면서 원하는 epoch 횟수만큼 파라미터를 업데이트 하는 실제 training 관련 로직. 추가적으로 checkpoint file로 파라미터를 저장하거나, tensorboad log를 남길 수 있음.
- evaluate.py/test.py : Training된 파라미터를 불러와서 학습 중간 성능을 판단하는 evaluation이나 완전히 학습이 끝났을 때, 새로운 input data에 대해 test/inference를 진행하는 로직. 터미널을 두 개에 train.py와 evaluate.py를 각각 실행하기 때문에 각각 스크립트를 따로 작성하는 것이 좋음.
- model.py : Keras Subclassing 형태의 모델 구조 class 정의
- dataset.py : 데이터 전처리 및 batch 단위로 묶는 로직
- utils.py : 딥러닝 메인 로직 외에 유틸리티성 기능들을 모아 놓은 로직
- loss.py : 모델의 Loss Function이 복잡한 경우, Loss Function을 정의

2. loss.py



```
def yolo_loss(predict,
              labels,
              each_object_num,
              num_classes,
              boxes_per_cell,
              cell_size,
              input_width,
              input_height,
              coord_scale,
              object_scale,
              noobject_scale,
              class_scale
              ):
  '''
  Args:
    predict: 3 - D tensor [cell_size, cell_size, num_classes + 5 *
boxes_per_cell]
    labels: 2-D list [object_num, 5] (xcenter (Absolute coordinate),
ycenter (Absolute coordinate), w (Absolute coordinate), h (Absolute
coordinate), class_num)
    each_object_num: each_object number in image (   object)
    num_classes: number of classes
    #
    boxes_per_cell: number of prediction boxes per each cell
    cell_size: each cell size
    input_width : input width of original image
    input_height : input_height of original image
    # Loss function
    coord_scale : coefficient for coordinate loss (  coord=5)
    object_scale : coefficient for object loss (  object    ,      )
    noobject_scale : coefficient for noobject loss (  noobj=0.5)
    class_scale : coefficient for class loss (   ,      )
  Returns:
    total_loss: coord_loss  + object_loss + noobject_loss + class_loss
    coord_loss
    object_loss
    noobject_loss
    class_loss
  '''
```

만약 Pascol VOC dataset(class=20)으로 box_per_cell=2라고 하면, 아래와 같이 벡터가 표현된다.



```
  # parse only coordinate vector
  predict_boxes = predict[:, :, num_classes + boxes_per_cell:]
  predict_boxes = tf.reshape(predict_boxes, [cell_size, cell_size,
boxes_per_cell, 4])
```

- `predict[:, :, num_classes + boxes_per_cell:]` : x,y,w,h의 position vector값만 추출
- `tf.reshape(predict_boxes, [cell_size, cell_size, boxes_per_cell, 4])` : (cell_size * cell_size * boxes_per_cell * 4)의 형태로 reshape

```
  # prediction : absolute coordinate
  pred_xcenter = predict_boxes[:, :, :, 0]
  pred_ycenter = predict_boxes[:, :, :, 1]
  pred_sqrt_w = tf.sqrt(tf.minimum(input_width * 1.0, tf.maximum(0.0,
predict_boxes[:, :, :, 2])))
  pred_sqrt_h = tf.sqrt(tf.minimum(input_height * 1.0, tf.maximum(0.0,
predict_boxes[:, :, :, 3])))
  pred_sqrt_w = tf.cast(pred_sqrt_w, tf.float32)
  pred_sqrt_h = tf.cast(pred_sqrt_h, tf.float32)
```

- `[:, :, :, 0]` : cell_size * cell_size * boxes_per_cell * 4 의 형태이기 때문
- `tf.sqrt(tf.minimum(input_width * 1.0, tf.maximum(0.0, predict_boxes[:, :, :, 2])))` : width와 hight은 large bounding box와 small bounding box에 대해 같은 error를 부여하여, 이를 해소하기 위해 squard root를 취해준다. 또한 전체 image를 벗어나지 않기 위해 최대최소의 제한을 준다.

loss.py - coord_loss

```
  # parse label
  labels = np.array(labels)
  labels = labels.astype('float32')
  label = labels[each_object_num, :]
  xcenter = label[0]
  ycenter = label[1]
  sqrt_w = tf.sqrt(label[2])
  sqrt_h = tf.sqrt(label[3])
```

- `np.array(labels)` : numpy array 형태로 형변환
- `labels[each_object_num, :]` : yolo의 loss function이 `each_object_num` 별로 하나씩 계산하기 때문에 전체에서 대응되는 object 하나만 가져옴

```
    # calulate iou between ground-truth and predictions
    iou_predict_truth = iou(predict_boxes, label[0:4])

    # find best box mask
    I = iou_predict_truth
    max_I = tf.reduce_max(I, 2, keepdims=True)
    best_box_mask = tf.cast((I >= max_I), tf.float32)
```

- `iou(predict_boxes, label[0:4])` : prediction bounding box와 정답 bounding box 의 IoU 계산
- `tf.reduce_max(I, 2, keepdims=True)` : 한 cell에서 가장 IoU가 큰 값을 추출
- `tf.cast((I >= max_I), tf.float32)` : `max_I`보다 크면 1, 작으면 0을 계산하여, 각각의 bounding box 별로 mask map 생성

```
    # find object exists cell mask
    object_exists_cell = np.zeros([cell_size, cell_size, 1])
    object_exists_cell_i, object_exists_cell_j = int(cell_size * ycenter
  / input_height), int(cell_size * xcenter / input_width)
    object_exists_cell[object_exists_cell_i][object_exists_cell_j] = 1
```

- `np.zeros([cell_size, cell_size, 1])` : 초기값을 0으로 설정
- `int(cell_size * ycenter / input_height), int(cell_size * xcenter / input_width)` : 좌표 값의 형태로 실제로 object가 있는 값들의 위치를 반환
- `object_exists_cell[object_exists_cell_i][object_exists_cell_j] = 1` : 실제로 object가 있는 cell은 1로 값을 바꿈

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

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

```
    # set coord_loss
    coord_loss = (tf.nn.l2_loss(object_exists_cell * best_box_mask *
  (pred_xcenter - xcenter) / (input_width / cell_size)) +
                  tf.nn.l2_loss(object_exists_cell * best_box_mask *
  (pred_ycenter - ycenter) / (input_height / cell_size)) +
                  tf.nn.l2_loss(object_exists_cell * best_box_mask *
  (pred_sqrt_w - sqrt_w)) / input_width +
                  tf.nn.l2_loss(object_exists_cell * best_box_mask *
  (pred_sqrt_h - sqrt_h)) / input_height ) \
                 * coord_scale
```

- `(pred_xcenter - xcenter) / (input_width / cell_size)` : loss function에서 xy좌표는 grid cell size 내의 상대 좌표로 계산해야하므로 상대 좌표로 변환
- `(pred_sqrt_w - sqrt_w)) / input_width` : loss function에서 w,h 좌표는 전체 이미지 내의 상대 좌표로 계산해야하므로 상대 좌표로 변환
- `object_exists_cell * best_box_mask` : loss function의 '1 obj ij'을 계산한 값
- `coord_scale` : 논문에서 λnobj=5

loss.py - object_loss

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

```
  # set object_loss information
  C = iou_predict_truth   #  confidence
  pred_C = predict[:, :, num_classes:num_classes + boxes_per_cell]    #
prediction confidence
```

- object가 있는 cell의 정답 confidence으로, 논문에서는 IOU(truth,pred) 으로 계산
- `predict[:, :, num_classes:num_classes + boxes_per_cell]` : confidence 값의 위치만 slicing

```
  # object_loss
  object_loss = tf.nn.l2_loss(object_exists_cell * best_box_mask *
(pred_C - C)) * object_scale
```

- `(pred_C - C)` : 두개의 차이 계산
- `object_exists_cell * best_box_mask` : loss function의 '1 obj ij'을 계산한 값
- `object_scale` : 논문에서는 사용하지 않았지만, 모델의 자유도를 위해 추가해줌.

loss.py - noobject_loss

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

```
  # noobject_loss
  noobject_loss = tf.nn.l2_loss((1 - object_exists_cell) * (pred_C)) *
noobject_scale
```

- `(1 - object_exists_cell)` : object가 존재하지 않는 cell (`object_exists_cell`이 0,1의 binary이기 때문)
- `(pred_C)` : 논문에서 object가 존재하지 않는 cell의 정답 confidence=0으로 지정
- `noobject_scale` : 논문에서 λnoobj=0.5

loss.py - class_loss

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

```
  # set class_loss information
  P = tf.one_hot(tf.cast(label[4], tf.int32), num_classes, dtype=tf.
float32)
  pred_P = predict[:, :, 0:num_classes]
```

- `predict[:, :, 0:num_classes]` : class 개수만큼 slciing
- `tf.one_hot(tf.cast(label[4], tf.int32), num_classes, dtype=tf.float32)` : label이 integer형태이기 때문에 one-hot encoding해줌

```
    # class loss
    class_loss = tf.nn.l2_loss(object_exists_cell * (pred_P - P)) *
  class_scale
```

- `object_exists_cell` : object가 존재하는 cell에서만 계산
- `class_scale` : 논문에서는 존재하지 않지만, 모델의 자유도를 위해 추가해줌