섹션 7. TensorFlow 2.0을 이용한 YOLO 논문 구현 3_train.py

train.py - 라이브러리 import & flags 지정

```
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import os
import random
```

```
from loss import yolo_loss
from model import YOLOv1
from dataset import process_each_ground_truth
from utils import draw_bounding_box_and_label_info, generate_color,
find_max_confidence_bounding_box, yolo_format_to_bounding_box_dict
```

```
from absl import flags
from absl import app
```

- absl : 구글에서 만든 파이썬 라이브러리
- flags를 통해 터미널에서 가변적으로 값을 변경할 수 있음. 코드에서 값을 변경할 수도 있지만, 터미널에서 값을 변경하면 코드를 수정하고 저장해야 하는 번거로움을 해소할 수 있음
    - ex) 터미널 상에서 'python 3 trian.py --lr_decay_rate=0.55' 를 입력하면 기존의 lr_decay_rate=0.5에서 0.55로 변경되어 계산됨

```
flags.DEFINE_float('lr_decay_rate', default=0.5, help='decay
rate for the learning rate')
```

```
flags.DEFINE_string('checkpoint_path', default='saved_model',
help='path to a directory to save model checkpoints during training')
flags.DEFINE_integer('save_checkpoint_steps', default=50, help='period
at which checkpoints are saved (defaults to every 50 steps)')
flags.DEFINE_string('tensorboard_log_path', default='tensorboard_log',
help='path to a directory to save tensorboard log')
flags.DEFINE_integer('validation_steps', default=50, help='period at
which test prediction result and save image')
flags.DEFINE_integer('num_epochs', default=135, help='training epochs')
# original paper : 135 epoch
flags.DEFINE_float('init_learning_rate', default=0.0001, help='initial
learning rate') # original paper : 0.001 (1epoch) -> 0.01 (75epoch) ->
0.001 (30epoch) -> 0.0001 (30epoch)
flags.DEFINE_float('lr_decay_rate', default=0.5, help='decay rate for
the learning rate')
flags.DEFINE_integer('lr_decay_steps', default=2000, help='number of
steps after which the learning rate is decayed by decay rate')
flags.DEFINE_integer('num_visualize_image', default=8, help='number of
visualize image for validation')

FLAGS = flags.FLAGS
```

- 변경할 수 있는 flags들

```
# set cat label dictionary
cat_label_dict = {
  0: "cat"
}
cat_class_to_label_dict = {v: k for k, v in cat_label_dict.items()}
```

- object detection task 관련 코드를 구현할 때, 빈번하게 사용되는 패턴
- dictionary 생성 시, key = interger, value = label(string type)
- 딥러닝 모델이 예측하는 값은 string이 아니라, interger, float과 같은 숫자 형이기 때문에 이와 같은 dictionary를 만들 필요가 있음
- Pascal VOC dataset의 label은 20개이지만, 현재는 'cat' label에 대해서만 train을 진행

train.py - YOLO 설정값 & Loss Function coefficient 지정

```
# set configuration value
batch_size = 24 # original paper : 64
input_width = 224 # original paper : 448
input_height = 224 # original paper : 448
cell_size = 7 # original paper : 7
num_classes = 1 # original paper : 20
boxes_per_cell = 2  # original paper : 2
```

```
# set color_list for drawing
color_list = generate_color(num_classes)
```

- label별로 색이 다른 bounding box를 생성하기 위해 class 개수만큼 random color 값을 가져옴

```
# set loss function coefficients
coord_scale = 10 # original paper : 5
class_scale = 0.1  # original paper : 1
object_scale = 1   # original paper : 1
noobject_scale = 0.5  # original paper : 0.5
```

- 현재 코드에서는 label이 'cat'하나여서 class_scale이 중요하지 않기 때문에 0.1의 낮은 값으로 할당

train.py - tensorflow_datasets 라이브러리를 이용해서 Pascal VOC cat dataset 불러오기

```
# load pascal voc2007/voc2012 dataset using tfds
# notice : voc2007 train data(=2,501 images) for test & voc2007 test
data(=4,952 images) for training
voc2007_test_split_data = tfds.load("voc/2007", split=tfds.Split.TEST,
batch_size=1)
voc2012_train_split_data = tfds.load("voc/2012", split=tfds.Split.
TRAIN, batch_size=1)
voc2012_validation_split_data = tfds.load("voc/2012", split=tfds.Split.
VALIDATION, batch_size=1)
train_data = voc2007_test_split_data.concatenate
(voc2012_train_split_data).concatenate(voc2012_validation_split_data)

# set validation data
voc2007_validation_split_data = tfds.load("voc/2007", split=tfds.Split.
VALIDATION, batch_size=1)
validation_data = voc2007_validation_split_data
```

- `voc2007 test data`의 개수가 4,952개이고 `voc2007 train data`의 개수가 2,501개이기 때문에 더 많은 양의 train을 진행하기 위해 `voc2007 train data`를 test로 정함
- original paper의 train dataset : `voc2007 train data` + `voc2007 validation data` + `voc2007 test data` + `voc2012 train data` + `voc2012 validation data`
- 현재 코드의 train dataset : `voc2007 test data` + `voc2012 train data` + `voc2012 validation data`

```
# label 7 : cat
# Reference : https://stackoverflow.com/questions/55731774/filter-
dataset-to-get-just-images-from-specific-class
def predicate(x, allowed_labels=tf.constant([7.0])):
  label = x['objects']['label']
  isallowed = tf.equal(allowed_labels, tf.cast(label, tf.float32))
  reduced = tf.reduce_sum(tf.cast(isallowed, tf.float32))

  return tf.greater(reduced, tf.constant(0.))

train_data = train_data.filter(predicate)
train_data = train_data.padded_batch(batch_size)

validation_data = validation_data.filter(predicate)
validation_data = validation_data.padded_batch(batch_size)
```

- ‘cat’ label인 data만 추출

train.py - train_step 정의 & for_loop을 이용한 gradient descent 수행 & 파라미터 및 텐서보드 로그 저장

```
def reshape_yolo_preds(preds):
  # flatten vector -> cell_size x cell_size x (num_classes + 5 *
boxes_per_cell)
  return tf.reshape(preds, [tf.shape(preds)[0], cell_size, cell_size,
num_classes + 5 * boxes_per_cell])
```

- 길게 flattening되어 있는 벡터를 YOLO에서 사용하기 편하도록 reshape

```
## define loss function
def calculate_loss(model, batch_image, batch_bbox, batch_labels):
  total_loss = 0.0
  coord_loss = 0.0
  object_loss = 0.0
  noobject_loss = 0.0
  class_loss = 0.0
  for batch_index in range(batch_image.shape[0]):
    image, labels, object_num = process_each_ground_truth(batch_image
[batch_index], batch_bbox[batch_index], batch_labels[batch_index],
input_width, input_height)
    image = tf.expand_dims(image, axis=0)

    predict = model(image)
    predict = reshape_yolo_preds(predict)

    for object_num_index in range(object_num):
      each_object_total_loss, each_object_coord_loss,
each_object_object_loss, each_object_noobject_loss,
each_object_class_loss = yolo_loss(predict[0],
                                   labels,
                                   object_num_index,
                                   num_classes,
                                   boxes_per_cell,
                                   cell_size,
                                   input_width,
                                   input_height,
                                   coord_scale,
                                   object_scale,
                                   noobject_scale,
                                   class_scale
                                   )

      total_loss = total_loss + each_object_total_loss
      coord_loss = coord_loss + each_object_coord_loss
      object_loss = object_loss + each_object_object_loss
      noobject_loss = noobject_loss + each_object_noobject_loss
      class_loss = class_loss + each_object_class_loss

  return total_loss, coord_loss, object_loss, noobject_loss, class_loss
```

- `tf.expand_dims(image, axis=0)` : 이미지가 한 장이 return된 것이어서 앞에 dummy dimension을 만들어줌
  - ex) image shape:[224, 224, 3] => dummy dimension 추가:[1, 224, 224, 3]
- `for object_num_index in range(object_num)` : 실제 object가 존재하는 개수만큼 for-loop

```
def train_step(optimizer, model, batch_image, batch_bbox, batch_labels):
  with tf.GradientTape() as tape:
    total_loss, coord_loss, object_loss, noobject_loss, class_loss =
calculate_loss(model, batch_image, batch_bbox, batch_labels)
  gradients = tape.gradient(total_loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  return total_loss, coord_loss, object_loss, noobject_loss, class_loss
```

- total loss에 기반하여 gradiend descent를 one-step 수행하는 과정

```
if __name__ == '__main__':
  app.run(main)
```

- absl library가 flags를 포함한 상태에서 'main' function을 호출

def main(_)

```
  # set learning rate decay
  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    FLAGS.init_learning_rate,
    decay_steps=FLAGS.lr_decay_steps,
    decay_rate=FLAGS.lr_decay_rate,
    staircase=True)
```

- 초기 learning_rate를 잡고 decay를 하는 반복 step과 step마다 learning_rate을 update해줌
- 현재는 init_learnin_rate=0.0001, lr_decay_steps=2000, lr_decay_rate=0.5 이므로 2000번 step에서는 learning_rate=0.00005, 4000번 step에서는 learning_rate=0.000025

```
  # set optimizer
  optimizer = tf.optimizers.Adam(lr_schedule)  # original paper : SGD
with momentum 0.9, decay 0.0005
```

```
  # check if checkpoint path exists
  if not os.path.exists(FLAGS.checkpoint_path):
    os.mkdir(FLAGS.checkpoint_path)
```

```
  # create YOLO model
  YOLOv1_model = YOLOv1(input_height, input_width, cell_size,
boxes_per_cell, num_classes)
```

```
  # set checkpoint manager
  ckpt = tf.train.Checkpoint(step=tf.Variable(0), model=YOLOv1_model)
  ckpt_manager = tf.train.CheckpointManager(ckpt,
                                            directory=FLAGS.
checkpoint_path,
                                            max_to_keep=None)
  latest_ckpt = tf.train.latest_checkpoint(FLAGS.checkpoint_path)

  # restore latest checkpoint
  if latest_ckpt:
    ckpt.restore(latest_ckpt)
    print('global_step : {}, checkpoint is restored!'.format(int(ckpt.
step)))
```

- `tf.train.latest_checkpoint(FLAGS.checkpoint_path)` : 저장된 파라미터가 있다면 해당 파라미터를 restore()을 이용하여 불러온 후 이어서 진행

```
  # set tensorboard log
  train_summary_writer = tf.summary.create_file_writer(FLAGS.
tensorboard_log_path +  '/train')
  validation_summary_writer = tf.summary.create_file_writer(FLAGS.
tensorboard_log_path +  '/validation')
```

```
  for epoch in range(FLAGS.num_epochs):
    num_batch = len(list(train_data))
    for iter, features in enumerate(train_data):
      batch_image = features['image']
      batch_bbox = features['objects']['bbox']
      batch_labels = features['objects']['label']

      batch_image = tf.squeeze(batch_image, axis=1)
      batch_bbox = tf.squeeze(batch_bbox, axis=1)
      batch_labels = tf.squeeze(batch_labels, axis=1)
```

- 지정 epoch 횟수만큼 for loop을 돌면서 전체 dataset에 있는 개별 batch단위의 데이터로 다시 for loop을 돌면서 가져옴
- `tf.squeeze` : tf.expand_dims 과 반대되는 api로, dummy dimension을 없애주는 api

```
      # run optimization and calculate loss
      total_loss, coord_loss, object_loss, noobject_loss, class_loss =
train_step(optimizer, YOLOv1_model, batch_image, batch_bbox,
batch_labels)

      # print log
      print("Epoch: %d, Iter: %d/%d, Loss: %f" % ((epoch+1), (iter+1),
num_batch, total_loss.numpy()))

      # save tensorboard log
      with train_summary_writer.as_default():
        tf.summary.scalar('learning_rate ', optimizer.lr(ckpt.step).
numpy(), step=int(ckpt.step))
        tf.summary.scalar('total_loss', total_loss, step=int(ckpt.step))
        tf.summary.scalar('coord_loss', coord_loss, step=int(ckpt.step))
        tf.summary.scalar('object_loss ', object_loss, step=int(ckpt.
step))
        tf.summary.scalar('noobject_loss ', noobject_loss, step=int
(ckpt.step))
        tf.summary.scalar('class_loss ', class_loss, step=int(ckpt.
step))

      # save checkpoint
      if ckpt.step % FLAGS.save_checkpoint_steps == 0:
        # save checkpoint
        ckpt_manager.save(checkpoint_number=ckpt.step)
        print('global_step : {}, checkpoint is saved!'.format(int(ckpt.
step)))

      ckpt.step.assign_add(1)
```

```
# occasionally check validation data and save tensorboard log
      if iter % FLAGS.validation_steps == 0:
        save_validation_result(YOLOv1_model, ckpt,
validation_summary_writer, FLAGS.num_visualize_image)
```

- `save_validation_result` : 현재의 yolo parameter에 기반한 validation 진행

train.py - save_validation_result 함수를 이용한 주기적인 val

```
def save_validation_result(model, ckpt, validation_summary_writer,
num_visualize_image):
  total_validation_total_loss = 0.0
  total_validation_coord_loss = 0.0
  total_validation_object_loss = 0.0
  total_validation_noobject_loss = 0.0
  total_validation_class_loss = 0.0
  for iter, features in enumerate(validation_data):
    batch_validation_image = features['image']
    batch_validation_bbox = features['objects']['bbox']
    batch_validation_labels = features['objects']['label']

    batch_validation_image = tf.squeeze(batch_validation_image, axis=1)
    batch_validation_bbox = tf.squeeze(batch_validation_bbox, axis=1)
    batch_validation_labels = tf.squeeze(batch_validation_labels,
axis=1)

    validation_total_loss, validation_coord_loss,
validation_object_loss, validation_noobject_loss, validation_class_loss
= calculate_loss(model, batch_validation_image, batch_validation_bbox,
batch_validation_labels)

    total_validation_total_loss = total_validation_total_loss +
validation_total_loss
    total_validation_coord_loss = total_validation_coord_loss +
validation_coord_loss
    total_validation_object_loss = total_validation_object_loss +
validation_object_loss
    total_validation_noobject_loss = total_validation_noobject_loss +
validation_noobject_loss
    total_validation_class_loss = total_validation_class_loss +
validation_class_loss
```

- 전체 validation data를 순회하면서 validation data와 yolo가 예측한 validation error값을 개별적으로 측정
- `total_validation_total_loss` : validation loss를 측정하면서 모델의 성능을 객관적으로 판단

```
  # save validation tensorboard log
  with validation_summary_writer.as_default():
    tf.summary.scalar('total_validation_total_loss',
total_validation_total_loss, step=int(ckpt.step))
    tf.summary.scalar('total_validation_coord_loss',
total_validation_coord_loss, step=int(ckpt.step))
    tf.summary.scalar('total_validation_object_loss ',
total_validation_object_loss, step=int(ckpt.step))
    tf.summary.scalar('total_validation_noobject_loss ',
total_validation_noobject_loss, step=int(ckpt.step))
    tf.summary.scalar('total_validation_class_loss ',
total_validation_class_loss, step=int(ckpt.step))
```

```
    # save tensorboard log
    with validation_summary_writer.as_default():
      tf.summary.image('validation_image_'+str(validation_image_index),
drawing_image, step=int(ckpt.step))
```

- 왼쪽에는 ground truth bounding box가 포함된 이미지, 오른쪽에는 yolo가 예측한 것들 중에서 가장 confidence가 큰 bounding box 하나만 뽑아서 drawing