

DataLab Cup 2: CNN for Object Detection

Shan-Hung Wu & DataLab
Fall 2023

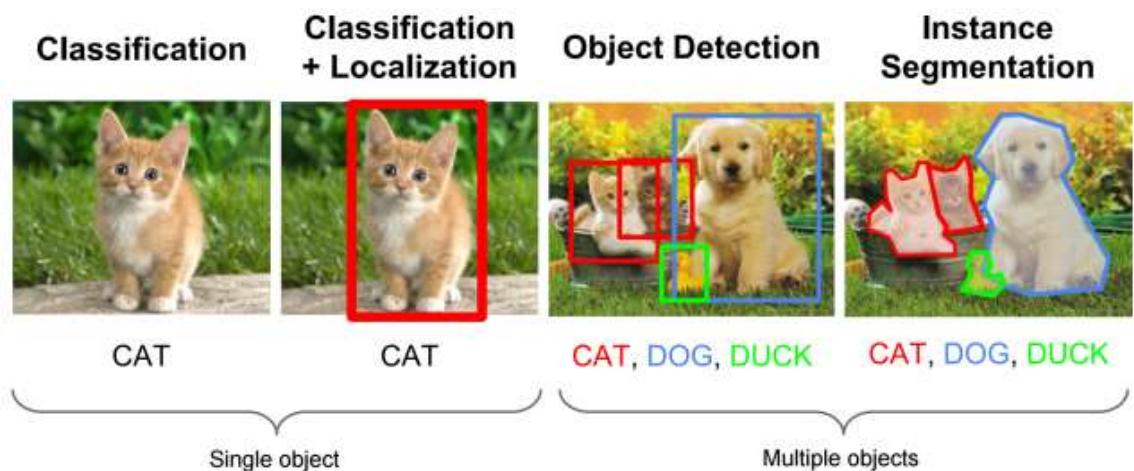
Competition Info

In this competition, you have to train a model that recognizes objects in an image. Your goal is to output bounding boxes for objects.

Platform: [Kaggle](#)

Problem description

Given an image(shape = [undefined, undefined, 3]), you need to output bounding box (x_{min} , y_{min} , x_{max} , y_{max} , class_label, confidence_score) for objects showed in image and its class.
[\(picture source\)](#)



Data provided

Dataset: [PASCAL VOC2007](#)

The dataset contains 20 classes. The train/val data has 5012 images containing 12608 annotated objects. We have preprocessed training dataset(5012 images) and testing dataset(4953 images) for you. You can download them on Kaggle.

```
In [1]: classes_name = ["aeroplane", "bicycle", "bird", "boat", "bottle",
    "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant",
    "sheep", "sofa", "train", "tvmonitor"]
```

Processed data format

The information for each image of training data are recorded in `VOCdevkit_train/VOC2007/Annotations`. However, we have processed those files for you into one record file: `pascal_voc_training_data.txt` in which each line records informations of each training images.

The data format of `pascal_voc_training_data` is:

`[image_name xmini ymini xmaxi ymaxi classi] (repeat number of objects times)`

Elements are separated by space.

```
In [2]: training_data_file = open("./pascal_voc_training_data.txt", "r")
for i, line in enumerate(training_data_file):
    if i >5:
        break
    line = line.strip()
    print(line)

000005.jpg 263 211 324 339 8 165 264 253 372 8 5 244 67 374 8 241 194 295 299 8 27
7 186 312 220 8
000007.jpg 141 50 500 330 6
000009.jpg 69 172 270 330 12 150 141 229 284 14 285 201 327 331 14 258 198 297 329
14
000012.jpg 156 97 351 270 6
000016.jpg 92 72 305 473 1
000017.jpg 185 62 279 199 14 90 78 403 336 12
```

As you can see, one image may have multiple objects. Another thing to note is, the heights and widths of the images in this dataset are different. Therefore, you are suggested to reshape images and ground truth bounding boxes' coordinates into same size.

In this competition, you can implement all kinds of object detection models (R-CNN, Fast-RCNN, Faster-RCNN, YOLOs, SSD,...etc.). Here we provide a simple template based on [YOLO\(You Only Look Once\)](#).

```
In [3]: import tensorflow as tf
import numpy as np
```

```
In [4]: gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # Currently, memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        # Select GPU number 1
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
```

3 Physical GPUs, 1 Logical GPUs

- Hyperparameters

```
In [5]: # common params
IMAGE_SIZE = 448
BATCH_SIZE = 8
NUM_CLASSES = 20
MAX_OBJECTS_PER_IMAGE = 20

# dataset params
DATA_PATH = './pascal_voc_training_data.txt'
IMAGE_DIR = './VOCdevkit_train/VOC2007/JPEGImages/'

# model params
CELL_SIZE = 7
BOXES_PER_CELL = 2
OBJECT_SCALE = 1
NOOBJECT_SCALE = 0.5
CLASS_SCALE = 1
COORD_SCALE = 5

# training params
LEARNING_RATE = 1e-4
EPOCHS = 3
```

Dataset Loader

We define a class especially to process training data, reading the records from `pascal_voc_training_data.txt` and follow the steps below to prepare data for our network:

1. Create Dataset using tensorflow data API.
2. In Dataset map function, read images and do preprocessing(ex. resizing , normalization).
3. In Dataset map function, change box information `[xmin, ymin, xmax, ymax]` coordinates into `[xcenter, ycenter, width, height]` attributes, which is easier for YOLO model to use.
4. Batch, Shuffle operations.

```
In [6]: class DatasetGenerator:
    """
    Load pascalVOC 2007 dataset and creates an input pipeline.
    - Reshapes images into 448 x 448
    - converts [0 1] to [-1 1]
    - shuffles the input
    - builds batches
    """

    def __init__(self):
        self.image_names = []
        self.record_list = []
        self.object_num_list = []
        # filling the record_list
        input_file = open(DATA_PATH, 'r')

        for line in input_file:
            line = line.strip()
            ss = line.split(' ')
            self.image_names.append(ss[0])

            self.record_list.append([float(num) for num in ss[1:]])
```

```

        self.object_num_list.append(min(len(self.record_list)[-1])//5,
                                    MAX_OBJECTS_PER_IMAGE))
    if len(self.record_list[-1]) < MAX_OBJECTS_PER_IMAGE*5:
        # if there are objects less than MAX_OBJECTS_PER_IMAGE, pad the list
        self.record_list[-1] = self.record_list[-1] +\
        [0., 0., 0., 0., 0.]*\
        (MAX_OBJECTS_PER_IMAGE-len(self.record_list[-1])//5)

    elif len(self.record_list[-1]) > MAX_OBJECTS_PER_IMAGE*5:
        # if there are objects more than MAX_OBJECTS_PER_IMAGE, crop the list
        self.record_list[-1] = self.record_list[-1][:MAX_OBJECTS_PER_IMAGE*5]

def _data_preprocess(self, image_name, raw_labels, object_num):
    image_file = tf.io.read_file(IMAGE_DIR+image_name)
    image = tf.io.decode_jpeg(image_file, channels=3)

    h = tf.shape(image)[0]
    w = tf.shape(image)[1]

    width_ratio = IMAGE_SIZE * 1.0 / tf.cast(w, tf.float32)
    height_ratio = IMAGE_SIZE * 1.0 / tf.cast(h, tf.float32)

    image = tf.image.resize(image, size=[IMAGE_SIZE, IMAGE_SIZE])
    image = (image/255) * 2 - 1

    raw_labels = tf.cast(tf.reshape(raw_labels, [-1, 5]), tf.float32)

    xmin = raw_labels[:, 0]
    ymin = raw_labels[:, 1]
    xmax = raw_labels[:, 2]
    ymax = raw_labels[:, 3]
    class_num = raw_labels[:, 4]

    xcenter = (xmin + xmax) * 1.0 / 2.0 * width_ratio
    ycenter = (ymin + ymax) * 1.0 / 2.0 * height_ratio

    box_w = (xmax - xmin) * width_ratio
    box_h = (ymax - ymin) * height_ratio

    labels = tf.stack([xcenter, ycenter, box_w, box_h, class_num], axis=1)

    return image, labels, tf.cast(object_num, tf.int32)

def generate(self):
    dataset = tf.data.Dataset.from_tensor_slices((self.image_names,
                                                np.array(self.record_list),
                                                np.array(self.object_num_list)))
    dataset = dataset.shuffle(100000)
    dataset = dataset.map(self._data_preprocess,
                          num_parallel_calls = tf.data.experimental.AUTOTUNE)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(buffer_size=200)

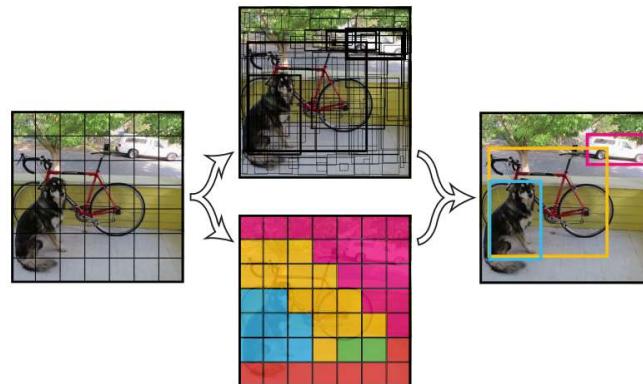
    return dataset

```

Now we can simply new a `DatasetGenerator` which can provide batches of training data for our model.

Object Detection Model (YOLO)

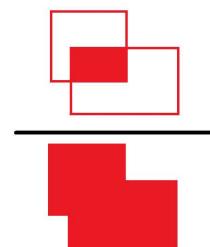
Different from Region Proposal based model, YOLO divide an image into $\text{cell_size} \times \text{cell_size}$ (say 7×7) cells, each has fixed number of output prediction boxes(coordinates, class_number, and confidence score). The final prediction would be the boxes with highest confidence score. The prediction of YOLO can be based on the output features extracted by the convolutional layers on the input image, which is actually "look once" on each image.



Intersection Over Union(IoU)

The loss calculation of **YOLO** includes calculating the intersection over union between the predicted boxes and the ground truth boxes. IoU is a common way to evaluate whether the predicted box coordinate is precise enough or not. The calcuation of iou is

$$\frac{\text{Predicted}_\text{Box} \cap \text{GroundTruth}_\text{Box}}{\text{Predicted}_\text{Box} \cup \text{GroundTruth}_\text{Box}}$$



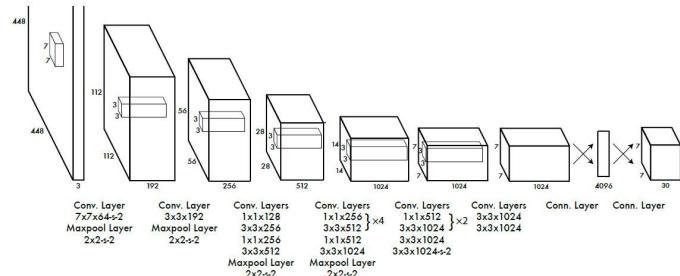
So we would like the IoU of our prediction and the ground the larger the better. In addition, IoU is also used when we evaluate an object detection model is good or not: the prediction is success if the IoU of the predicted box and the ground truth is larger than a threshold.

Model Architecture

24 convolution layers followed by 2 fully connected layers.

Use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$



model

```
In [7]: from tensorflow import keras
from tensorflow.keras import layers
```

```
In [8]: def conv_leaky_relu(inputs, filters, size, stride):
    x = layers.Conv2D(filters, size, stride, padding="same",
                      kernel_initializer=tf.keras.initializers.TruncatedNormal())(i
    x = layers.LeakyReLU(0.1)(x)

    return x
```

```
In [9]: img_inputs = keras.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
x = conv_leaky_relu(img_inputs, 64, 7, 2)
x = layers.MaxPool2D()(x)
x = conv_leaky_relu(x, 192, 3, 1)
x = layers.MaxPool2D()(x)
x = conv_leaky_relu(x, 128, 1, 1)
x = conv_leaky_relu(x, 256, 3, 1)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = layers.MaxPool2D()(x)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = conv_leaky_relu(x, 256, 1, 1)
x = conv_leaky_relu(x, 512, 3, 1)
x = conv_leaky_relu(x, 1024, 3, 1)
x = layers.MaxPool2D()(x)
x = conv_leaky_relu(x, 512, 1, 1)
x = conv_leaky_relu(x, 1024, 3, 1)
x = conv_leaky_relu(x, 512, 1, 1)
x = conv_leaky_relu(x, 1024, 3, 1)
x = conv_leaky_relu(x, 512, 1, 1)
x = conv_leaky_relu(x, 1024, 3, 1)
x = conv_leaky_relu(x, 1024, 3, 2)
x = conv_leaky_relu(x, 1024, 3, 1)
x = conv_leaky_relu(x, 1024, 3, 1)
x = layers.Flatten()(x)
x = layers.Dense(4096,
                 kernel_initializer=tf.keras.initializers.TruncatedNormal(stddev=0.
x = layers.LeakyReLU(0.1)(x)
outputs = layers.Dense(1470,
                      kernel_initializer=tf.keras.initializers.TruncatedNormal(sto

YOLO = keras.Model(inputs=img_inputs, outputs=outputs, name="YOLO")
```

```
In [10]: YOLO.summary()
```

Model: "YOLO"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 448, 448, 3)]	0
conv2d (Conv2D)	(None, 224, 224, 64)	9472
leaky_re_lu (LeakyReLU)	(None, 224, 224, 64)	0
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_1 (Conv2D)	(None, 112, 112, 192)	110784
leaky_re_lu_1 (LeakyReLU)	(None, 112, 112, 192)	0
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 192)	0
conv2d_2 (Conv2D)	(None, 56, 56, 128)	24704
leaky_re_lu_2 (LeakyReLU)	(None, 56, 56, 128)	0
conv2d_3 (Conv2D)	(None, 56, 56, 256)	295168
leaky_re_lu_3 (LeakyReLU)	(None, 56, 56, 256)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	65792
leaky_re_lu_4 (LeakyReLU)	(None, 56, 56, 256)	0
conv2d_5 (Conv2D)	(None, 56, 56, 512)	1180160
leaky_re_lu_5 (LeakyReLU)	(None, 56, 56, 512)	0
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 512)	0
conv2d_6 (Conv2D)	(None, 28, 28, 256)	131328
leaky_re_lu_6 (LeakyReLU)	(None, 28, 28, 256)	0
conv2d_7 (Conv2D)	(None, 28, 28, 512)	1180160
leaky_re_lu_7 (LeakyReLU)	(None, 28, 28, 512)	0
conv2d_8 (Conv2D)	(None, 28, 28, 256)	131328
leaky_re_lu_8 (LeakyReLU)	(None, 28, 28, 256)	0
conv2d_9 (Conv2D)	(None, 28, 28, 512)	1180160
leaky_re_lu_9 (LeakyReLU)	(None, 28, 28, 512)	0
conv2d_10 (Conv2D)	(None, 28, 28, 256)	131328
leaky_re_lu_10 (LeakyReLU)	(None, 28, 28, 256)	0
conv2d_11 (Conv2D)	(None, 28, 28, 512)	1180160
leaky_re_lu_11 (LeakyReLU)	(None, 28, 28, 512)	0
conv2d_12 (Conv2D)	(None, 28, 28, 256)	131328
leaky_re_lu_12 (LeakyReLU)	(None, 28, 28, 256)	0

conv2d_13 (Conv2D)	(None, 28, 28, 512)	1180160
leaky_re_lu_13 (LeakyReLU)	(None, 28, 28, 512)	0
conv2d_14 (Conv2D)	(None, 28, 28, 512)	262656
leaky_re_lu_14 (LeakyReLU)	(None, 28, 28, 512)	0
conv2d_15 (Conv2D)	(None, 28, 28, 1024)	4719616
leaky_re_lu_15 (LeakyReLU)	(None, 28, 28, 1024)	0
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 1024)	0
conv2d_16 (Conv2D)	(None, 14, 14, 512)	524800
leaky_re_lu_16 (LeakyReLU)	(None, 14, 14, 512)	0
conv2d_17 (Conv2D)	(None, 14, 14, 1024)	4719616
leaky_re_lu_17 (LeakyReLU)	(None, 14, 14, 1024)	0
conv2d_18 (Conv2D)	(None, 14, 14, 512)	524800
leaky_re_lu_18 (LeakyReLU)	(None, 14, 14, 512)	0
conv2d_19 (Conv2D)	(None, 14, 14, 1024)	4719616
leaky_re_lu_19 (LeakyReLU)	(None, 14, 14, 1024)	0
conv2d_20 (Conv2D)	(None, 14, 14, 1024)	9438208
leaky_re_lu_20 (LeakyReLU)	(None, 14, 14, 1024)	0
conv2d_21 (Conv2D)	(None, 7, 7, 1024)	9438208
leaky_re_lu_21 (LeakyReLU)	(None, 7, 7, 1024)	0
conv2d_22 (Conv2D)	(None, 7, 7, 1024)	9438208
leaky_re_lu_22 (LeakyReLU)	(None, 7, 7, 1024)	0
conv2d_23 (Conv2D)	(None, 7, 7, 1024)	9438208
leaky_re_lu_23 (LeakyReLU)	(None, 7, 7, 1024)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 4096)	205524992
leaky_re_lu_24 (LeakyReLU)	(None, 4096)	0
dense_1 (Dense)	(None, 1470)	6022590
<hr/>		
Total params: 271,703,550		
Trainable params: 271,703,550		
Non-trainable params: 0		

Define loss

```
In [11]: # base boxes (for loss calculation)
base_boxes = np.zeros([CELL_SIZE, CELL_SIZE, 4])
```

```
# initialization for each cell
for y in range(CELL_SIZE):
    for x in range(CELL_SIZE):
        base_boxes[y, x, :] = [IMAGE_SIZE / CELL_SIZE * x,
                               IMAGE_SIZE / CELL_SIZE * y, 0, 0]

base_boxes = np.resize(base_boxes, [CELL_SIZE, CELL_SIZE, 1, 4])
base_boxes = np.tile(base_boxes, [1, 1, BOXES_PER_CELL, 1])
```

In [12]:

```
def yolo_loss(predicts, labels, objects_num):
    """
    Add Loss to all the trainable variables
    Args:
        predicts: 4-D tensor [batch_size, cell_size, cell_size, num_classes + 5 * b]
        =====> (num_classes, boxes_per_cell, 4 * boxes_per_cell)
        labels : 3-D tensor of [batch_size, max_objects, 5]
        objects_num: 1-D tensor [batch_size]
    """
    loss = 0.

    #you can parallel the code with tf.map_fn or tf.vectorized_map (big performance
    for i in tf.range(BATCH_SIZE):
        predict = predicts[i, :, :, :]
        label = labels[i, :, :]
        object_num = objects_num[i]

        for j in tf.range(object_num):
            results = losses_calculation(predict, label[j:j+1, :])
            loss = loss + results

    return loss/BATCH_SIZE
```

In [13]:

```
def iou(boxes1, boxes2):
    """calculate ious
    Args:
        boxes1: 4-D tensor [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL, 4] =====> (x_center
        boxes2: 1-D tensor [4] =====> (x_center, y_center, w, h)

    Return:
        iou: 3-D tensor [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
        =====> iou score for each cell
    """

    #boxes1 : [4(xmin, ymin, xmax, ymax), cell_size, cell_size, boxes_per_cell]
    boxes1 = tf.stack([boxes1[:, :, :, 0] - boxes1[:, :, :, 2] / 2, boxes1[:, :, :, 0],
                      boxes1[:, :, :, 0] + boxes1[:, :, :, 2] / 2, boxes1[:, :, :, 0],])

    #boxes1 : [cell_size, cell_size, boxes_per_cell, 4(xmin, ymin, xmax, ymax)]
    boxes1 = tf.transpose(boxes1, [1, 2, 3, 0])

    boxes2 = tf.stack([boxes2[0] - boxes2[2] / 2, boxes2[1] - boxes2[3] / 2,
                      boxes2[0] + boxes2[2] / 2, boxes2[1] + boxes2[3] / 2])

    #calculate the left up point of boxes' overlap area
    lu = tf.maximum(boxes1[:, :, :, 0:2], boxes2[0:2])
    #calculate the right down point of boxes overlap area
    rd = tf.minimum(boxes1[:, :, :, 2:], boxes2[2:])

    #intersection
    intersection = rd - lu
```

```

#the size of the intersection area
inter_square = intersection[:, :, :, 0] * intersection[:, :, :, 1]

mask = tf.cast(intersection[:, :, :, 0] > 0, tf.float32) * tf.cast(intersection[:, :, :, 1], tf.float32)

#if intersection is negative, then the boxes don't overlap
inter_square = mask * inter_square

#calculate the boxes1 square and boxes2 square
square1 = (boxes1[:, :, :, 2] - boxes1[:, :, :, 0]) * (boxes1[:, :, :, 3] - boxes1[:, :, :, 1])
square2 = (boxes2[2] - boxes2[0]) * (boxes2[3] - boxes2[1])

return inter_square/(square1 + square2 - inter_square + 1e-6)

def losses_calculation(predict, label):
    """
    calculate loss
    Args:
        predict: 3-D tensor [cell_size, cell_size, num_classes + 5 * boxes_per_cell]
        label : [1, 5] (x_center, y_center, w, h, class)
    """
    label = tf.reshape(label, [-1])

    #Step A. calculate objects tensor [CELL_SIZE, CELL_SIZE]
    #turn pixel position into cell position (corner)
    min_x = (label[0] - label[2] / 2) / (IMAGE_SIZE / CELL_SIZE)
    max_x = (label[0] + label[2] / 2) / (IMAGE_SIZE / CELL_SIZE)

    min_y = (label[1] - label[3] / 2) / (IMAGE_SIZE / CELL_SIZE)
    max_y = (label[1] + label[3] / 2) / (IMAGE_SIZE / CELL_SIZE)

    min_x = tf.floor(min_x)
    min_y = tf.floor(min_y)

    max_x = tf.minimum(tf.math.ceil(max_x), CELL_SIZE)
    max_y = tf.minimum(tf.math.ceil(max_y), CELL_SIZE)

    #calculate mask of object with cells
    onset = tf.cast(tf.stack([max_y - min_y, max_x - min_x]), dtype=tf.int32)
    object_mask = tf.ones(onset, tf.float32)

    offset = tf.cast(tf.stack([min_y, CELL_SIZE - max_y, min_x, CELL_SIZE - max_x]), tf.int32)
    offset = tf.reshape(offset, (2, 2))
    object_mask = tf.pad(object_mask, offset, "CONSTANT")

    #Step B. calculate the coordination of object center and the corresponding mask
    #turn pixel position into cell position (center)
    center_x = label[0] / (IMAGE_SIZE / CELL_SIZE)
    center_x = tf.floor(center_x)

    center_y = label[1] / (IMAGE_SIZE / CELL_SIZE)
    center_y = tf.floor(center_y)

    response = tf.ones([1, 1], tf.float32)

    #calculate the coordination of object center with cells
    objects_center_coord = tf.cast(tf.stack([center_y, CELL_SIZE - center_y - 1,
                                              center_x, CELL_SIZE - center_x - 1]), tf.int32)
    objects_center_coord = tf.reshape(objects_center_coord, (2, 2))

    #make mask
    response = tf.pad(response, objects_center_coord, "CONSTANT")

```

```

#Step C. calculate iou_predict_truth [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
predict_boxes = predict[:, :, NUM_CLASSES + BOXES_PER_CELL:]

predict_boxes = tf.reshape(predict_boxes, [CELL_SIZE,
                                           CELL_SIZE,
                                           BOXES_PER_CELL, 4])
#cell position to pixel position
predict_boxes = predict_boxes * [IMAGE_SIZE / CELL_SIZE,
                                 IMAGE_SIZE / CELL_SIZE,
                                 IMAGE_SIZE, IMAGE_SIZE]

#if there's no predict_box in that cell, then the base_boxes will be calculated
predict_boxes = base_boxes + predict_boxes

iou_predict_truth = iou(predict_boxes, label[0:4])

#calculate C tensor [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
C = iou_predict_truth * tf.reshape(response, [CELL_SIZE, CELL_SIZE, 1])

#calculate I tensor [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
I = iou_predict_truth * tf.reshape(response, [CELL_SIZE, CELL_SIZE, 1])

max_I = tf.reduce_max(I, 2, keepdims=True)

#replace Large iou scores with response (object center) value
I = tf.cast((I >= max_I), tf.float32) * tf.reshape(response, (CELL_SIZE, CELL_SIZE, 1))

#calculate no_I tensor [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
no_I = tf.ones_like(I, dtype=tf.float32) - I

p_C = predict[:, :, NUM_CLASSES:NUM_CLASSES + BOXES_PER_CELL]

#calculate truth x, y, sqrt_w, sqrt_h 0-D
x = label[0]
y = label[1]

sqrt_w = tf.sqrt(tf.abs(label[2]))
sqrt_h = tf.sqrt(tf.abs(label[3]))

#calculate predict p_x, p_y, p_sqrt_w, p_sqrt_h 3-D [CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
p_x = predict_boxes[:, :, :, 0]
p_y = predict_boxes[:, :, :, 1]

p_sqrt_w = tf.sqrt(tf.minimum(IMAGE_SIZE * 1.0, tf.maximum(0.0, predict_boxes[:, :, :, 2])))
p_sqrt_h = tf.sqrt(tf.minimum(IMAGE_SIZE * 1.0, tf.maximum(0.0, predict_boxes[:, :, :, 3])))

#calculate ground truth p 1-D tensor [NUM_CLASSES]
P = tf.one_hot(tf.cast(label[4], tf.int32), NUM_CLASSES, dtype=tf.float32)

#calculate predicted p_P 3-D tensor [CELL_SIZE, CELL_SIZE, NUM_CLASSES]
p_P = predict[:, :, 0:NUM_CLASSES]

#class_loss
class_loss = tf.nn.l2_loss(tf.reshape(object_mask, (CELL_SIZE, CELL_SIZE, 1)) * p_C)

#object_loss
object_loss = tf.nn.l2_loss(I * (p_C - C)) * OBJECT_SCALE

#noobject_loss
noobject_loss = tf.nn.l2_loss(no_I * (p_C)) * NOOBJECT_SCALE

#coord_loss
coord_loss = (tf.nn.l2_loss(I * (p_x - x)/(IMAGE_SIZE/CELL_SIZE)) +
              tf.nn.l2_loss(I * (p_y - y)/(IMAGE_SIZE/CELL_SIZE)) +

```

```
tf.nn.l2_loss(I * (p_sqrt_w - sqrt_w))/IMAGE_SIZE +
tf.nn.l2_loss(I * (p_sqrt_h - sqrt_h))/IMAGE_SIZE) * COORD_SCALE
```

```
return class_loss + object_loss + noobject_loss + coord_loss
```

Start Training

Now we can start training our YOLO model:

```
In [14]: dataset = DatasetGenerator().generate()
```

```
In [15]: optimizer = tf.keras.optimizers.Adam(LEARNING_RATE)
train_loss_metric = tf.keras.metrics.Mean(name='loss')
```

```
In [16]: ckpt = tf.train.Checkpoint(epoch=tf.Variable(0), net=YOLO)

manager = tf.train.CheckpointManager(ckpt, './ckpts/YOLO', max_to_keep=3,
                                      checkpoint_name='yolo')
```

```
In [17]: @tf.function
def train_step(image, labels, objects_num):
    with tf.GradientTape() as tape:
        outputs = YOLO(image)
        class_end = CELL_SIZE * CELL_SIZE * NUM_CLASSES
        conf_end = class_end + CELL_SIZE * CELL_SIZE * BOXES_PER_CELL
        class_probs = tf.reshape(outputs[:, 0:class_end], (-1, 7, 7, 20))
        confs = tf.reshape(outputs[:, class_end:conf_end], (-1, 7, 7, 2))
        boxes = tf.reshape(outputs[:, conf_end:], (-1, 7, 7, 2*4))
        predicts = tf.concat([class_probs, confs, boxes], 3)

        loss = yolo_loss(predicts, labels, objects_num)
        train_loss_metric(loss)

        grads = tape.gradient(loss, YOLO.trainable_weights)
        optimizer.apply_gradients(zip(grads, YOLO.trainable_weights))
```

```
In [18]: from datetime import datetime
```

```
In [ ]: print("{}， start training.".format(datetime.now()))
for i in range(EPOCHS):
    train_loss_metric.reset_states()
    ckpt.epoch.assign_add(1)

    for idx, (image, labels, objects_num) in enumerate(dataset):
        train_step(image, labels, objects_num)

    print("{}， Epoch {}: loss {:.2f}".format(datetime.now(), i+1, train_loss_metric))

    save_path = manager.save()
    print("Saved checkpoint for epoch {}: {}".format(int(ckpt.epoch), save_path))
```

Predict Test data

After training, we should run testing on the test data images. Since we should output a txt file in similar format as `pascal_voc_training_data.txt`, we should change the YOLO output box [xcenter, ycenter, width, height] format back to [xmin, ymin, xmax, ymax].

Process YOLO's predictions

Below is the function process the output of the YOLO network and return the most confident box and its corresponding class and confidence score.

```
In [19]: def process_outputs(outputs):
    """
    Process YOLO outputs into bounding boxes
    """

    class_end = CELL_SIZE * CELL_SIZE * NUM_CLASSES
    conf_end = class_end + CELL_SIZE * CELL_SIZE * BOXES_PER_CELL
    class_probs = np.reshape(outputs[:, 0:class_end], (-1, 7, 7, 20))
    confs = np.reshape(outputs[:, class_end:conf_end], (-1, 7, 7, 2))
    boxes = np.reshape(outputs[:, conf_end:], (-1, 7, 7, 2*4))
    predicts = np.concatenate([class_probs, confs, boxes], 3)

    p_classes = predicts[0, :, :, 0:20]
    C = predicts[0, :, :, 20:22]
    coordinate = predicts[0, :, :, 22:]

    p_classes = np.reshape(p_classes, (CELL_SIZE, CELL_SIZE, 1, 20))
    C = np.reshape(C, (CELL_SIZE, CELL_SIZE, BOXES_PER_CELL, 1))

    P = C * p_classes
    #P's shape [7, 7, 2, 20]

    #choose the most confidence one
    max_conf = np.max(P)
    index = np.argmax(P)

    index = np.unravel_index(index, P.shape)

    class_num = index[3]

    coordinate = np.reshape(coordinate,
                           (CELL_SIZE,
                            CELL_SIZE,
                            BOXES_PER_CELL,
                            4))

    max_coordinate = coordinate[index[0], index[1], index[2], :]

    xcenter = max_coordinate[0]
    ycenter = max_coordinate[1]
    w = max_coordinate[2]
    h = max_coordinate[3]

    xcenter = (index[1] + xcenter) * (IMAGE_SIZE/float(CELL_SIZE))
    ycenter = (index[0] + ycenter) * (IMAGE_SIZE/float(CELL_SIZE))

    w = w * IMAGE_SIZE
    h = h * IMAGE_SIZE

    xmin = xcenter - w/2.0
    ymin = ycenter - h/2.0

    xmax = xmin + w
    ymax = ymin + h

    return xmin, ymin, xmax, ymax, class_num, max_conf
```

Build Test dataset Iterator

```
In [20]: test_img_files = open('./pascal_voc_testing_data.txt')
test_img_dir = './VOCdevkit_test/VOC2007/JPEGImages/'
test_images = []

for line in test_img_files:
    line = line.strip()
    ss = line.split(' ')
    test_images.append(ss[0])

test_dataset = tf.data.Dataset.from_tensor_slices(test_images)

def load_img_data(image_name):
    image_file = tf.io.read_file(test_img_dir+image_name)
    image = tf.image.decode_jpeg(image_file, channels=3)

    h = tf.shape(image)[0]
    w = tf.shape(image)[1]

    image = tf.image.resize(image, size=[IMAGE_SIZE, IMAGE_SIZE])
    image = (image/255) * 2 - 1

    return image_name, image, h, w

test_dataset = test_dataset.map(load_img_data, num_parallel_calls = tf.data.experimental.AUTOTUNE)
test_dataset = test_dataset.batch(32)
```

```
In [22]: ckpt = tf.train.Checkpoint(net=Yolo)
ckpt.restore('./ckpts/Yolo/yolo-3')
```

```
Out[22]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fbc9c2e29d0>
```

```
In [23]: @tf.function
def prediction_step(img):
    return Yolo(img, training=False)
```

Make Prediction and Output to txt file

To run the evaluation program we provide, you should output your prediction with this format(similar but different with `pascal_voc_training_data.txt`)

```
image_name {xmin_i  ymin_i  xmax_i  ymax_i  class_i
confidence_score} (repeat number of objects times)
```

for each line in the txt file.

Note: it is also acceptable if there are multiple lines with same image name(different box predictions).

```
In [ ]: output_file = open('./test_prediction.txt', 'w')

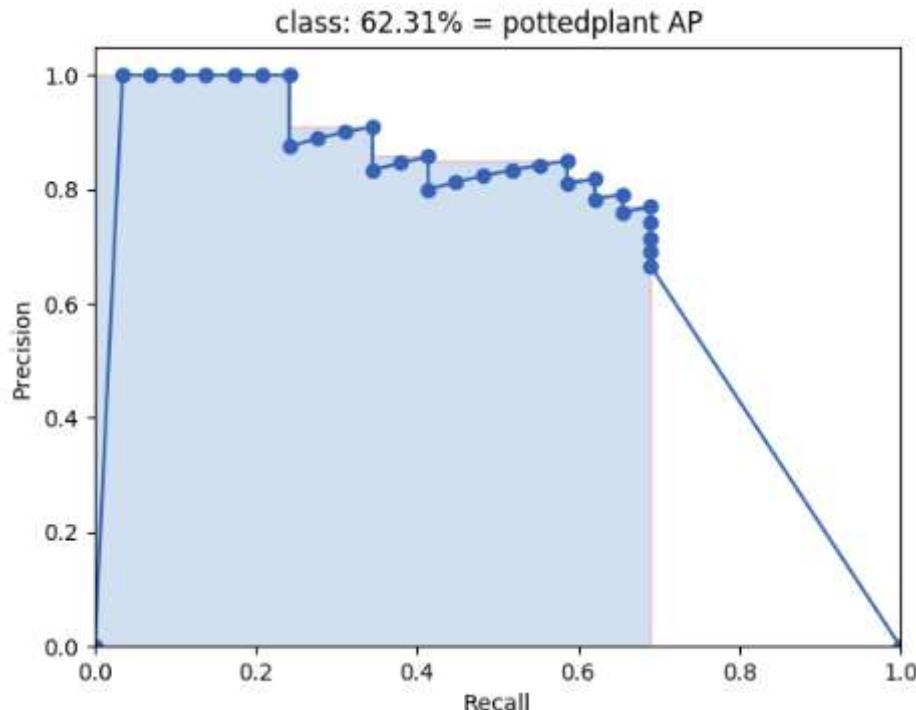
for img_name, test_img, img_h, img_w in test_dataset:
    batch_num = img_name.shape[0]
    for i in range(batch_num):
        xmin, ymin, xmax, ymax, class_num, conf = process_outputs(prediction_step(test_img[i]))
        xmin, ymin, xmax, ymax = xmin*(img_w[i:i+1]/IMAGE_SIZE), ymin*(img_h[i:i+1]
```

```
#img filename, xmin, ymin, xmax, ymax, class, confidence
output_file.write(img_name[i:i+1].numpy()[0].decode('ascii')+" %d %d %d %d\n"
output_file.close()
```

Run Evaluation Metric

Finally, you can use following example code to run the evaluation program we provide and output the csv file. Please submit the csv file onto Kaggle.

The evaluation program calculates [mean Average Precision\(mAP\)](#) of your output boxes. It will first sort your prediction by your confidence score, and get the Precision/Recall curve:



(img source:[github](https://github.com/Cartucho/mAP))

Then the area under this curve is the mAP of this class.

We separated test data into 10 groups, and calculates the mAP of each class for each group. Your goal is to maximize the total mAP score.

```
In [ ]: import sys
sys.path.insert(0, './evaluate')
```

```
In [ ]: import evaluate
#evaluate.evaluate("input prediction file name", "desire output csv file name")
evaluate.evaluate('./test_prediction.txt', './output_file.csv')
```

Visualization

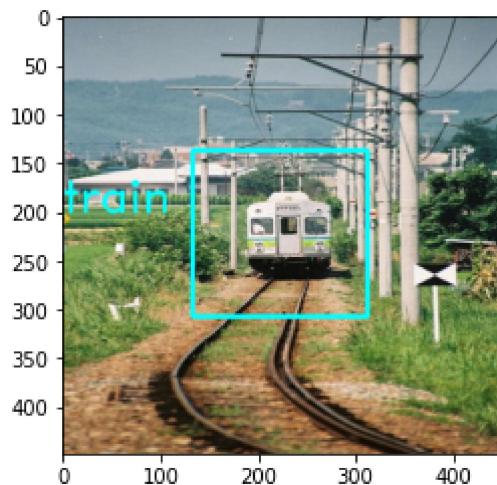
Here we provide a simple code to draw the predicted bounding box and class onto the image and visualize using matplotlib.

```
In [24]: %matplotlib inline
import matplotlib.pyplot as plt
import cv2
```

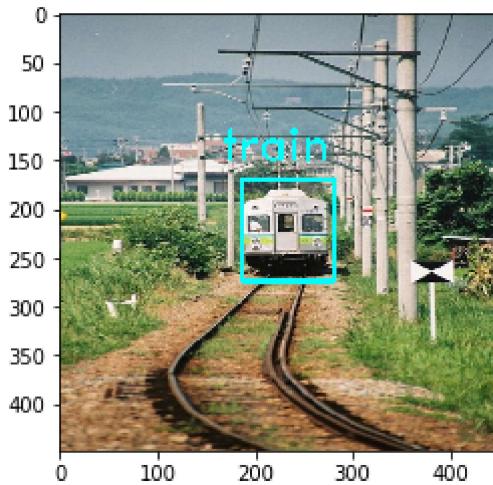
```
In [25]: np_img = cv2.imread('./VOCdevkit_test/VOC2007/JPEGImages/000002.jpg')
resized_img = cv2.resize(np_img, (IMAGE_SIZE, IMAGE_SIZE))
np_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB)
resized_img = np_img
np_img = np_img.astype(np.float32)
np_img = np_img / 255.0 * 2 - 1
np_img = np.reshape(np_img, (1, IMAGE_SIZE, IMAGE_SIZE, 3))

y_pred = YOLO(np_img, training=False)
xmin, ymin, xmax, ymax, class_num, conf = process_outputs(y_pred)
class_name = classes_name[class_num]
cv2.rectangle(resized_img, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (0, 255, 255), 2)
cv2.putText(resized_img, class_name, (0, 200), 2, 1.5, (0, 255, 255), 2)

plt.imshow(resized_img)
plt.show()
```



As you can see, the result of current model needs some improvements. After training, your model shall have at least the capability to output prediction like this:



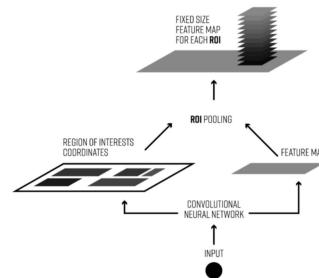
Other Models

[Fast-RCNN](#)

[Roi pooling \(api source\)](#)

Region of interest pooling (RoI pooling) is an operation widely used in object detection tasks using convolutional neural networks. It was proposed by Ross Girshick ([paper](#)) and it achieves a significant speedup of both training and testing. It also maintains a high detection accuracy. The layer takes two inputs:

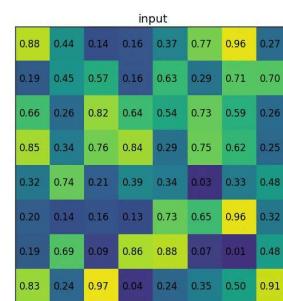
1. A fixed-size feature map obtained from a deep convolutional network with many convolutions and max pooling layers.
2. An N-by-5 matrix of representing a list of regions, where N is a number of RoIs. The first columns represents the image index and the remaining four are the coordinates of the top left and bottom right corners of the region.



What does the RoI pooling actually do? For every region of interest from the input list, it takes a section of the input feature map that corresponds to it and scales it to some pre-defined size (e.g., 7×7). The scaling is done by:

1. Dividing the region proposal into equal-sized sections (the number of which is the same as the dimension of the output)
2. Finding the largest value in each section
3. Copying these max values to the output buffer

The result is that from a list of rectangles with different sizes we can quickly get a list of corresponding feature maps with a fixed size.

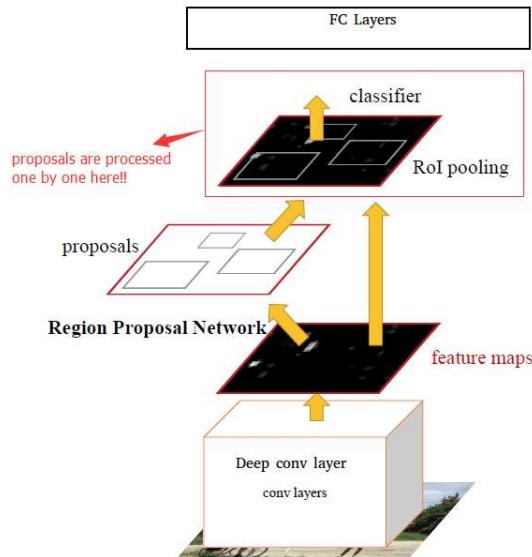


(source: [deepsense.ai](<https://deepsense.ai/region-of-interest-pooling-explained/>))

Faster-RCNN

The main idea is use the last conv layers to infer region proposals. Faster-RCNN consists of two modules.

- Region Proposal Network (RPN): Gives a set of rectangles based on deep convolution layer.
- Fast-RCNN RoI Pooling layer: Classify each proposal, and refining proposal location.



SSD

Single-Shot Multi Box Detector is a model based on YOLO, but it has better ability to detect diverse scale objects.

- [reference](#)

Precautions

Scoring and Report

Your score will be part of the final private result on Kaggle and part of your report.

- Your report(`.ipynb` file) should have:
 - Your code
 - What kind of models you have tried and how did they work.
 - Anything you've done and want to tell us.
 - What problems occurred and how did you solve them.
- Also, please upload the `.py` file exported from your `.ipynb` file.

What you can do

- Implement other models **by yourself**.
- Load pretrained models trained on ImageNet. e.g. vgg19, resnet, etc. (ex. `tf.keras.applications`)
- Data augmentation.

What you should NOT do

- Load pretrained object detection model weights directly from other sources.
- Clone other's project from github or other sites. (You should implement by yourself).

- Plagiarize codes from other teams.
- Pretrain your network on other dataset (you can only use the pascal data we provided on Kaggle).
- Use the groundtruth to generate output.

Competition timeline

- 2023/11/09 (Thu) competition announced
- 2023/11/23 (Thu) 23:59 competition deadline
- 2023/11/26 (Sun) 23:59 report deadline
- 2023/11/30 (Thu) top 3 team sharing