***Loss值计算***

|  |
| --- |
| ***import tensorflow as tf***  ***from keras import backend as K***  ***#---------------------------------------------------#***  ***# 将预测值的每个特征层调成真实值***  ***#---------------------------------------------------#***  ***def yolo\_head(feats, anchors, num\_classes, input\_shape):***  ***num\_anchors = len(anchors)***  ***# [1, 1, 1, num\_anchors, 2]***  ***anchors\_tensor = K.reshape(K.constant(anchors), [1, 1, 1, num\_anchors, 2])***  ***# 获得x，y的网格***  ***# (13, 13, 1, 2)***  ***grid\_shape = K.shape(feats)[1:3] # height, width***  ***grid\_y = K.tile(K.reshape(K.arange(0, stop=grid\_shape[0]), [-1, 1, 1, 1]),***  ***[1, grid\_shape[1], 1, 1])***  ***grid\_x = K.tile(K.reshape(K.arange(0, stop=grid\_shape[1]), [1, -1, 1, 1]),***  ***[grid\_shape[0], 1, 1, 1])***  ***grid = K.concatenate([grid\_x, grid\_y])#连接，中心点***  ***grid = K.cast(grid, K.dtype(feats))***  ***# (batch\_size,13,13,3,85)***  ***feats = K.reshape(feats, [-1, grid\_shape[0], grid\_shape[1],***  ***num\_anchors, num\_classes + 5])***  ***# 将预测值调成真实值***  ***box\_xy = (K.sigmoid(feats[..., :2]) + grid) / K.cast(grid\_shape[::-1],***  ***K.dtype(feats))# box\_xy对应框的中心点***  ***box\_wh = K.exp(feats[..., 2:4]) \* anchors\_tensor / K.cast(input\_shape[::-1],***  ***K.dtype(feats))# box\_wh对应框的宽和高***  ***# 在计算loss的时候返回如下参数***  ***return grid, feats, box\_xy, box\_wh***  ***#---------------------------------------------------#***  ***# 用于计算每个预测框与真实框的iou 差距***  ***#---------------------------------------------------#***  ***def box\_iou(b1, b2):***  ***#Tensor("yolo\_loss/while/strided\_slice\_2:0", shape=(?, ?, 3, 4), dtype=float32) 1***  ***# Tensor("yolo\_loss/while/boolean\_mask/GatherV2:0", shape=(?, ?), dtype=float32) 2***  ***# Tensor("yolo\_loss/while\_1/strided\_slice\_2:0", shape=(?, ?, 3, 4), dtype=float32) 1***  ***# Tensor("yolo\_loss/while\_1/boolean\_mask/GatherV2:0", shape=(?, ?), dtype=float32) 2***  ***# Tensor("yolo\_loss/while\_2/strided\_slice\_2:0", shape=(?, ?, 3, 4), dtype=float32) 1***  ***# Tensor("yolo\_loss/while\_2/boolean\_mask/GatherV2:0", shape=(?, ?), dtype=float32) 2***  ***# 13,13,3,1,4***  ***# 计算左上角的坐标和右下角的坐标***  ***b1 = K.expand\_dims(b1, -2)#在-2的位置增加一个维度***  ***#Tensor("yolo\_loss/while/ExpandDims:0", shape=(?, ?, 3, 1, 4), dtype=float32)***  ***# Tensor("yolo\_loss/while\_1/ExpandDims:0", shape=(?, ?, 3, 1, 4), dtype=float32)***  ***# Tensor("yolo\_loss/while\_2/ExpandDims:0", shape=(?, ?, 3, 1, 4), dtype=float32)***  ***b1\_xy = b1[..., :2]***  ***#print(b1\_xy,'1')***  ***b1\_wh = b1[..., 2:4]***  ***#print(b1\_wh,'2')***  ***b1\_wh\_half = b1\_wh/2.***  ***b1\_mins = b1\_xy - b1\_wh\_half***  ***#Tensor("yolo\_loss/while/sub:0", shape=(?, ?, 3, 1, 2), dtype=float32)***  ***#print(b1\_mins)***  ***b1\_maxes = b1\_xy + b1\_wh\_half***  ***# 1,n,4***  ***# 计算左上角和右下角的坐标***  ***b2 = K.expand\_dims(b2, 0)***  ***b2\_xy = b2[..., :2]#中心点***  ***b2\_wh = b2[..., 2:4]#高宽***  ***b2\_wh\_half = b2\_wh/2.#高宽的一半***  ***b2\_mins = b2\_xy - b2\_wh\_half***  ***b2\_maxes = b2\_xy + b2\_wh\_half***  ***# 计算重合面积***  ***intersect\_mins = K.maximum(b1\_mins, b2\_mins)#最小值最的最大差***  ***intersect\_maxes = K.minimum(b1\_maxes, b2\_maxes)#***  ***intersect\_wh = K.maximum(intersect\_maxes - intersect\_mins, 0.)#***  ***#print(intersect\_wh,'1')***  ***#Tensor("yolo\_loss/while/Maximum\_1:0", shape=(?, ?, 3, ?, 2), dtype=float32) 1***  ***# Tensor("yolo\_loss/while\_1/Maximum\_1:0", shape=(?, ?, 3, ?, 2), dtype=float32) 1***  ***# Tensor("yolo\_loss/while\_2/Maximum\_1:0", shape=(?, ?, 3, ?, 2), dtype=float32) 1***  ***intersect\_area = intersect\_wh[..., 0] \* intersect\_wh[..., 1]#面积***  ***b1\_area = b1\_wh[..., 0] \* b1\_wh[..., 1]#高和宽求出面积***  ***b2\_area = b2\_wh[..., 0] \* b2\_wh[..., 1]#高和宽求出面积***  ***iou = intersect\_area / (b1\_area + b2\_area - intersect\_area)***  ***return iou***  ***#---------------------------------------------------#***  ***# loss值计算***  ***#---------------------------------------------------#***  ***def yolo\_loss(args, anchors, num\_classes, ignore\_thresh=.5):***  ***#print(args)***  ***# 一共有三层***  ***num\_layers = len(anchors)//3***  ***# 将预测结果和实际ground truth分开，args是[\*model\_body.output, \*y\_true]***  ***# y\_true是一个列表，包含三个特征层，shape分别为(m,13,13,3,85),(m,26,26,3,85),(m,52,52,3,85)。***  ***y\_true = args[num\_layers:]#真实值***  ***yolo\_outputs = args[:num\_layers]#预测值***  ***# 先验框***  ***# 678为116,90, 156,198, 373,3260***  ***# 345为30,61, 62,45, 59,119***  ***# 012为10,13, 16,30, 33,23,***  ***anchor\_mask = [[6,7,8], [3,4,5], [0,1,2]] if num\_layers==3 else [[3,4,5], [1,2,3]]***  ***# 得到input\_shpae为416,416***  ***input\_shape = K.cast(K.shape(yolo\_outputs[0])[1:3] \* 32, K.dtype(y\_true[0]))***  ***#print(input\_shape) Tensor("yolo\_loss/Cast:0", shape=(2,), dtype=float32)***  ***# 得到网格的shape为13,13;26,26;52,52***  ***grid\_shapes = [K.cast(K.shape(yolo\_outputs[l])[1:3], K.dtype(y\_true[0])) for l in range(num\_layers)]***  ***#print(grid\_shapes)***  ***#[<tf.Tensor 'yolo\_loss/Cast\_1:0' shape=(2,) dtype=float32>,***  ***# <tf.Tensor 'yolo\_loss/Cast\_2:0' shape=(2,) dtype=float32>,***  ***# <tf.Tensor 'yolo\_loss/Cast\_3:0' shape=(2,) dtype=float32>]***  ***loss = 0***  ***# 取出每一张图片***  ***# m的值就是batch\_size***  ***m = K.shape(yolo\_outputs[0])[0]***  ***mf = K.cast(m, K.dtype(yolo\_outputs[0]))***  ***# y\_true是一个列表，包含三个特征层，shape分别为(m,13,13,3,85),(m,26,26,3,85),(m,52,52,3,85)。***  ***for l in range(num\_layers):#循环三遍***  ***# 以第一个特征层(m,13,13,3,85)为例子***  ***# 取出该特征层中存在目标的点的位置。(m,13,13,3,1)***  ***object\_mask = y\_true[l][..., 4:5]***  ***# 取出其对应的种类(m,13,13,3,80)***  ***true\_class\_probs = y\_true[l][..., 5:]***  ***# 将yolo\_outputs的特征层输出进行处理***  ***# grid为网格结构(13,13,1,2)，raw\_pred为尚未处理的预测结果(m,13,13,3,85)***  ***# 还有解码后的xy，wh，(m,13,13,3,2)***  ***grid, raw\_pred, pred\_xy, pred\_wh = yolo\_head(yolo\_outputs[l],***  ***anchors[anchor\_mask[l]], num\_classes, input\_shape)***  ***# 这个是解码后的预测的box的位置***  ***# (m,13,13,3,4)***  ***pred\_box = K.concatenate([pred\_xy, pred\_wh])#连接***  ***# 找到负样本群组，第一步是创建一个数组，[]***  ***ignore\_mask = tf.TensorArray(K.dtype(y\_true[0]), size=1, dynamic\_size=True)***  ***#print('ignore\_mask',ignore\_mask)***  ***object\_mask\_bool = K.cast(object\_mask, 'bool')***  ***# 对每一张图片计算ignore\_mask***  ***def loop\_body(b, ignore\_mask):***  ***# 取出第b副图内，真实存在的所有的box的参数***  ***# n,4***  ***true\_box = tf.boolean\_mask(y\_true[l][b,...,0:4], object\_mask\_bool[b,...,0])***  ***# 计算预测结果与真实情况的iou***  ***# pred\_box为13,13,3,4***  ***# 计算的结果是每个pred\_box和其它所有真实框的iou***  ***# 13,13,3,n***  ***iou = box\_iou(pred\_box[b], true\_box)***  ***# 13,13,3,1***  ***best\_iou = K.max(iou, axis=-1)***  ***# 判断预测框的iou小于ignore\_thresh则认为该预测框没有与之对应的真实框***  ***# 则被认为是这幅图的负样本***  ***ignore\_mask = ignore\_mask.write(b, K.cast(best\_iou<ignore\_thresh, K.dtype(true\_box)))***  ***return b+1, ignore\_mask***  ***# 遍历所有的图片***  ***\_, ignore\_mask = K.control\_flow\_ops.while\_loop(lambda b,***  ***\*args: b<m, loop\_body, [0, ignore\_mask])***  ***# 将每幅图的内容压缩，进行处理***  ***ignore\_mask = ignore\_mask.stack()***  ***#(m,13,13,3,1,1)***  ***ignore\_mask = K.expand\_dims(ignore\_mask, -1)***  ***# 将真实框进行编码，使其格式与预测的相同，后面用于计算loss***  ***raw\_true\_xy = y\_true[l][..., :2]\*grid\_shapes[l][:] - grid#生成真正的中心点***  ***raw\_true\_wh = K.log(y\_true[l][..., 2:4] / anchors[anchor\_mask[l]] \* input\_shape[::-1]) # object\_mask如果真实存在目标则保存其wh值***  ***# switch接口，就是一个if/else条件判断语句***  ***raw\_true\_wh = K.switch(object\_mask, raw\_true\_wh, K.zeros\_like(raw\_true\_wh))***  ***box\_loss\_scale = 2 - y\_true[l][...,2:3]\*y\_true[l][...,3:4]***  ***xy\_loss = object\_mask\*box\_loss\_scale\* K.binary\_crossentropy(raw\_true\_xy,***  ***raw\_pred[...,0:2], from\_logits=True)***  ***wh\_loss = object\_mask \* box\_loss\_scale \* 0.5 \* K.square(raw\_true\_wh-raw\_pred[...,2:4])***  ***# 如果该位置本来有框，那么计算1与置信度的交叉熵***  ***# 如果该位置本来没有框，而且满足best\_iou<ignore\_thresh，则被认定为负样本***  ***# best\_iou<ignore\_thresh用于限制负样本数量***  ***confidence\_loss=object\_mask\*K.binary\_crossentropy(object\_mask,***  ***raw\_pred[...,4:5], from\_logits=True)+(object\_mask) \* K.binary\_crossentropy(object\_mask,***  ***(1-raw\_pred[...,4:5], from\_logits=True) \* ignore\_mask***  ***class\_loss = object\_mask \* K.binary\_crossentropy(true\_class\_probs,***  ***raw\_pred[...,5:], from\_logits=True)***  ***xy\_loss = K.sum(xy\_loss) / mf#中心点***  ***wh\_loss = K.sum(wh\_loss) / mf#高宽***  ***confidence\_loss = K.sum(confidence\_loss) / mf#置信度***  ***class\_loss = K.sum(class\_loss) / mf#类偏差***  ***loss += xy\_loss + wh\_loss + confidence\_loss + class\_loss#***  ***return loss*** |

***Yolo\_body***

|  |
| --- |
| ***from functools import wraps#修饰符***  ***from keras.layers import Conv2D, Add, ZeroPadding2D, UpSampling2D, Concatenate#卷积层***  ***from keras.layers.advanced\_activations import LeakyReLU***  ***from keras.layers.normalization import BatchNormalization***  ***from keras.models import Model***  ***from keras.regularizers import l2***  ***from functools import reduce***  ***def compose(\*funcs):***  ***if funcs:***  ***return reduce(lambda f, g: lambda \*a, \*\*kw: g(f(\*a, \*\*kw)), funcs)***  ***else:***  ***raise ValueError('Composition of empty sequence not supported.')***  ***#---------------------------------------------------#***  ***# darknet53 的主体部分***  ***#---------------------------------------------------#***  ***def darknet\_body(image\_input):# darknet主体***  ***#image\_input=（416，616，3）***  ***x = DarknetConv2D\_BN\_Leaky(32, (3,3))(image\_input)#将图片形状输入到特殊卷积块，长，宽不变，通道数调整为32***  ***#（208，208，64）***  ***x = resblock\_body(x, 64, 1)#经过特殊卷积块输入的特征层，输出的通道数，循环的次数***  ***#（104，104，128，）***  ***x = resblock\_body(x, 128, 2)#经过上面的卷积层，高宽/2，通道数x2***  ***#（52，52，256）***  ***x = resblock\_body(x, 256, 8)#x=形状，8=次数***  ***feat1 = x#提取第一个特征层，传入yolo的解码预测值处理的网络中***  ***# （26，26，512）***  ***x = resblock\_body(x, 512, 8)***  ***feat2 = x#提取第二个特征层，传入yolo的解码预测值处理的网络中***  ***#（13，13，1024）***  ***x = resblock\_body(x, 1024, 4)***  ***feat3 = x#提取第三个特征层，传入yolo的解码预测值处理的网络中***  ***return feat1,feat2,feat3#返回三次的数据，将三个提取出的特征层传入主体部分***  ***#--------------------------------------------------#***  ***# 单次卷积***  ***#--------------------------------------------------#***  ***@wraps(Conv2D)***  ***def DarknetConv2D(\*args, \*\*kwargs):***  ***darknet\_conv\_kwargs = {'kernel\_regularizer': l2(5e-4)}***  ***darknet\_conv\_kwargs['padding'] = 'valid' if kwargs.get('strides')==(2,2) else 'same'***  ***darknet\_conv\_kwargs.update(kwargs)***  ***return Conv2D(\*args, \*\*darknet\_conv\_kwargs)***  ***#---------------------------------------------------#***  ***# 卷积块***  ***# DarknetConv2D + BatchNormalization + LeakyReLU***  ***#---------------------------------------------------#***  ***def DarknetConv2D\_BN\_Leaky(\*args, \*\*kwargs):***  ***no\_bias\_kwargs = {'use\_bias': False}***  ***no\_bias\_kwargs.update(kwargs)***  ***return compose(***  ***DarknetConv2D(\*args, \*\*no\_bias\_kwargs),***  ***BatchNormalization(),***  ***LeakyReLU(alpha=0.1))***  ***#---------------------------------------------------#***  ***# 卷积块***  ***# DarknetConv2D + BatchNormalization + LeakyReLU***  ***#---------------------------------------------------#***  ***def resblock\_body(x, num\_filters, num\_blocks):#卷积块***  ***x = ZeroPadding2D(((1,0),(1,0)))(x)***  ***#步长为2，长和宽变为原来的2/1***  ***x = DarknetConv2D\_BN\_Leaky(num\_filters, (3,3), strides=(2,2))(x)#残差网络的输入***  ***for i in range(num\_blocks):#残差网络***  ***y = DarknetConv2D\_BN\_Leaky(num\_filters//2, (1,1))(x)#1x1卷积，把通道数压缩，***  ***y = DarknetConv2D\_BN\_Leaky(num\_filters, (3,3))(y)#3x3，把通道数扩展回原来***  ***x = Add()([x,y])***  ***return x***  ***#---------------------------------------------------#***  ***# 特征层->最后的输出***  ***#---------------------------------------------------#***  ***def make\_last\_layers(x, num\_filters, out\_filters):***  ***# 五次卷积***  ***x = DarknetConv2D\_BN\_Leaky(num\_filters, (1,1))(x)***  ***x = DarknetConv2D\_BN\_Leaky(num\_filters\*2, (3,3))(x)***  ***x = DarknetConv2D\_BN\_Leaky(num\_filters, (1,1))(x)***  ***x = DarknetConv2D\_BN\_Leaky(num\_filters\*2, (3,3))(x)***  ***x = DarknetConv2D\_BN\_Leaky(num\_filters, (1,1))(x)#进行5次卷积后有5个去向，***  ***#第一个去向***  ***#第2个去向***  ***# 将最后的通道数调整为outfilter***  ***y = DarknetConv2D\_BN\_Leaky(num\_filters\*2, (3,3))(x)***  ***y = DarknetConv2D(out\_filters, (1,1))(y)***  ***return x, y***  ***#---------------------------------------------------#***  ***# 特征层->最后的输出***  ***#---------------------------------------------------#***  ***def yolo\_body(image\_input, num\_anchors, num\_classes):***  ***# 生成darknet53的主干模型***  ***# （52，52，256）***  ***# （26，26，512）***  ***# （13，13，1024）***  ***feat1,feat2,feat3 = darknet\_body(image\_input)#获取图片输出的三个特征层***  ***darknet = Model(image\_input, feat3)***  ***# 第一个特征层***  ***# y1=(batch\_size,13,13,3,85)***  ***x, y1 = make\_last\_layers(darknet.output, 512, num\_anchors\*(num\_classes+5))***  ***#y1是第一个特征层处理完的结果，x是五次卷积后的结果***  ***#x=Tensor("leaky\_re\_lu\_57/LeakyRelu:0", shape=(?, ?, ?, 512), dtype=float32)***  ***x = compose(DarknetConv2D\_BN\_Leaky(256, (1,1)),UpSampling2D(2))(x)#进行上采样，***  ***x = Concatenate()([x,feat2])#对26x26的特征层进行结合（堆叠）***  ***# 第二个特征层***  ***# y2=(batch\_size,26,26,3,85)***  ***x, y2 = make\_last\_layers(x, 256, num\_anchors\*(num\_classes+5))#进行上采样，***  ***x = compose(DarknetConv2D\_BN\_Leaky(128, (1,1)),UpSampling2D(2))(x)***  ***x = Concatenate()([x,feat1])#对52x52的特征层进行结合（堆叠）***  ***# 第三个特征层***  ***# y3=(batch\_size,52,52,3,85)***  ***x, y3 = make\_last\_layers(x, 128, num\_anchors\*(num\_classes+5))#进行上采样，***  ***return Model(image\_input, [y1,y2,y3])#52，52，256）***  ***# （26，26，512）***  ***# （13，13，1024 三个和形状*** |

***Tensorbord***

|  |
| --- |
| ***from keras.callbacks import TensorBoard***  ***tensorbord = TensorBoard(log\_dir='./logs', histogram\_freq=1, write\_grads=True)***  ***# mode.fit(callable=[tensorbord])*** |

***随机数据增强***

|  |
| --- |
| ***"""各种各样的效用函数."""***  ***from PIL import Image***  ***import numpy as np***  ***from matplotlib.colors import rgb\_to\_hsv, hsv\_to\_rgb***  ***def rand(a=0, b=1):***  ***return np.random.rand() \* (b - a) + a***  ***#返回浮点16位0到1的数***  ***def get\_random\_data(annotation\_line, input\_shape, random=True, max\_boxes=20, jitter=0.3, hue=.1, sat=1.5, val=1.5,***  ***proc\_img=True):***  ***'''r实时数据增强的随机预处理'''#输入数据annotation\_line，输入形状***  ***line = annotation\_line.split()***  ***image = Image.open(line[0])#取第一个路径***  ***iw, ih = image.size#print('iw, ih',iw, ih)iw, ih 500 375 iw, ih 400 300图像的高和宽***  ***h, w = input\_shape#h, w 416 416***  ***box = np.array([np.array(list(map(int, box.split(',')))) for box in line[1:]])***  ***#[[ 32 111 466 254 0][]]将后面的真实框提取出来放入list转numpy***  ***# 图片拉伸***  ***import random***  ***nw=int(random.uniform(350,480))***  ***nh=int(random.uniform(350,480))***  ***image = image.resize((nw,nh),Image.BICUBIC)***  ***# place image***  ***dx = int(rand(0, w - nw))***  ***dy = int(rand(0, h - nh))***  ***new\_image = Image.new('RGB', (w, h), (128, 128, 128))#创建一个新图的通道，高宽，背景颜色为灰色***  ***#print(np.array(image).shape)***  ***new\_image.paste(image, (dx, dy))***  ***#print(np.array(new\_image).shape)***  ***image = new\_image***  ***# flip image or not***  ***flip = rand() < .5***  ***if flip: image = image.transpose(Image.FLIP\_LEFT\_RIGHT)***  ***# distort image***  ***hue = rand(-hue, hue)#0.1***  ***sat = rand(1, sat) if rand() < .5 else 1 / rand(1, sat)#1.5***  ***val = rand(1, val) if rand() < .5 else 1 / rand(1, val)***  ***# import matplotlib.pyplot as plt#查看图片***  ***# plt.imshow(np.array(image))***  ***# plt.show()***  ***#print(np.array(image).shape)***  ***x = rgb\_to\_hsv(np.array(image) / 255.)#***  ***x[..., 0] += hue***  ***x[..., 0][x[..., 0] > 1] -= 1***  ***x[..., 0][x[..., 0] < 0] += 1***  ***x[..., 1] \*= sat***  ***x[..., 2] \*= val***  ***#print(x)***  ***x[x > 1] = 1***  ***x[x < 0] = 0***  ***image\_data = hsv\_to\_rgb(x) # numpy array, 0 to 1***  ***#print(np.array(image).shape)***  ***# correct boxes***  ***box\_data = np.zeros((max\_boxes, 5))***  ***if len(box) > 0:***  ***np.random.shuffle(box)#打乱***  ***box[:, [0, 2]] = box[:, [0, 2]] \* nw / iw + dx***  ***box[:, [1, 3]] = box[:, [1, 3]] \* nh / ih + dy***  ***if flip: box[:, [0, 2]] = w - box[:, [2, 0]]***  ***box[:, 0:2][box[:, 0:2] < 0] = 0***  ***box[:, 2][box[:, 2] > w] = w***  ***box[:, 3][box[:, 3] > h] = h***  ***box\_w = box[:, 2] - box[:, 0]***  ***box\_h = box[:, 3] - box[:, 1]***  ***box = box[np.logical\_and(box\_w > 1, box\_h > 1)] # discard invalid box***  ***if len(box) > max\_boxes: box = box[:max\_boxes]***  ***box\_data[:len(box)] = box***  ***return image\_data, box\_data*** |