1. **from \_\_future\_\_ import division, print\_function**
2. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
3. **import math**
4. **train\_file = './data/my\_data/train.txt'**
5. **val\_file = './data/my\_data/val.txt'**
6. **restore\_path = './data/darknet\_weights/yolov3.ckpt'**
7. **save\_dir = './checkpoint/'**
8. **log\_dir = './data/logs/'**
9. **progress\_log\_path = './data/progress.log'**
10. **anchor\_path = './data/yolo\_anchors.txt'**
11. **class\_name\_path = './data/voc\_names.txt'**
12. **batch\_size = 1**
13. **img\_size = [416, 416]**
14. **letterbox\_resize = True**
15. **total\_epoches = 100**
16. **train\_evaluation\_step = 100**
17. **val\_evaluation\_epoch = 2**
18. **save\_epoch = 10**
19. **batch\_norm\_decay = 0.99**
20. **weight\_decay = 5e-4**
21. **global\_step = 0**
22. **num\_threads = 10**
23. **prefetech\_buffer = 5**
24. **optimizer\_name = 'momentum'**
25. **save\_optimizer = True**
26. **learning\_rate\_init = 1e-4**
27. **lr\_type = 'piecewise'**
28. **lr\_decay\_epoch = 5**
29. **lr\_decay\_factor = 0.96**
30. **lr\_lower\_bound = 1e-6**
31. **pw\_boundaries = [30, 50]**
32. **pw\_values = [learning\_rate\_init, 3e-5, 1e-5]**
33. **restore\_include = None**
34. **restore\_exclude = ['yolov3/yolov3\_head/Conv\_14', 'yolov3/yolov3\_head/Conv\_6', 'yolov3/yolov3\_head/Conv\_22']**
35. **update\_part = ['yolov3/yolov3\_head']**
36. **multi\_scale\_train = True**
37. **use\_label\_smooth = True**
38. **use\_focal\_loss = True**
39. **use\_mix\_up = True**
40. **use\_warm\_up = True**
41. **warm\_up\_epoch = 3**
42. **nms\_threshold = 0.45**
43. **score\_threshold = 0.01**
44. **nms\_topk = 150**
45. **eval\_threshold = 0.5**
46. **use\_voc\_07\_metric = False**
47. **anchors = parse\_anchors(anchor\_path)**
48. **classes = read\_class\_names(class\_name\_path)**
49. **class\_num = len(classes)**
50. **train\_img\_cnt = len(open(train\_file, 'r').readlines())**
51. **val\_img\_cnt = len(open(val\_file, 'r').readlines())**
52. **train\_batch\_num = int(math.ceil(float(train\_img\_cnt) / batch\_size))**
53. **lr\_decay\_freq = int(train\_batch\_num \* lr\_decay\_epoch)**
54. **pw\_boundaries = [float(i) \* train\_batch\_num + global\_step for i in pw\_boundaries]**
55. **from \_\_future\_\_ import division, print\_function**
56. **import tensorflow as tf**
57. **import numpy as np**
58. **import argparse**
59. **import cv2**
60. **import time**
61. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
62. **from utils.nms\_utils import gpu\_nms**
63. **from utils.plot\_utils import get\_color\_table, plot\_one\_box**
64. **from utils.data\_aug import letterbox\_resize**
65. **from model import yolov3**
66. **from tensorflow.compat.v1 import ConfigProto**
67. **from tensorflow.compat.v1 import InteractiveSession**
68. **config = ConfigProto()**
69. **config.gpu\_options.allow\_growth = True**
70. **session = InteractiveSession(config=config)**
71. **parser = argparse.ArgumentParser(description="YOLO-V3 video test procedure.")**
72. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
73. **help="The path of the anchor txt file.")**
74. **parser.add\_argument("--new\_size", nargs='\*', type=int, default=[416, 416],**
75. **help="Resize the input image with `new\_size`, size format: [width, height]")**
76. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=True,**
77. **help="Whether to use the letterbox resize.")**
78. **parser.add\_argument("--class\_name\_path", type=str, default="./data/voc\_names.txt",**
79. **help="The path of the class names.")**
80. **parser.add\_argument("--restore\_path", type=str, default="./checkpoint/model-epoch\_90\_step\_276639\_loss\_0.5149\_lr\_1e-05",**
81. **help="The path of the weights to restore.")**
82. **parser.add\_argument("--save\_video", type=lambda x: (str(x).lower() == 'true'), default=False,**
83. **help="Whether to save the video detection results.")**
84. **args = parser.parse\_args()**
85. **args.anchors = parse\_anchors(args.anchor\_path)**
86. **args.classes = read\_class\_names(args.class\_name\_path)**
87. **args.num\_class = len(args.classes)**
88. **color\_table = get\_color\_table(args.num\_class)**
89. **vid = cv2.VideoCapture(0)**
90. **if args.save\_video:**
91. **fourcc = cv2.VideoWriter\_fourcc('m', 'p', '4', 'v')**
92. **videoWriter = cv2.VideoWriter('video\_result.mp4', fourcc, video\_fps, (video\_width, video\_height))**
93. **with tf.Session() as sess:**
94. **input\_data = tf.placeholder(tf.float32, [1, args.new\_size[1], args.new\_size[0], 3], name='input\_data')**
95. **yolo\_model = yolov3(args.num\_class, args.anchors)**
96. **with tf.variable\_scope('yolov3'):**
97. **pred\_feature\_maps = yolo\_model.forward(input\_data, False)**
98. **pred\_boxes, pred\_confs, pred\_probs = yolo\_model.predict(pred\_feature\_maps)**
99. **pred\_scores = pred\_confs \* pred\_probs**
100. **boxes, scores, labels = gpu\_nms(pred\_boxes, pred\_scores, args.num\_class, max\_boxes=200, score\_thresh=0.3, nms\_thresh=0.45)**
101. **saver = tf.train.Saver()**
102. **saver.restore(sess, args.restore\_path)**
103. **while True:**
104. **ret, img\_ori = vid.read()**
105. **if args.letterbox\_resize:**
106. **img, resize\_ratio, dw, dh = letterbox\_resize(img\_ori, args.new\_size[0], args.new\_size[1])**
107. **else:**
108. **height\_ori, width\_ori = img\_ori.shape[:2]**
109. **img = cv2.resize(img\_ori, tuple(args.new\_size))**
110. **img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**
111. **img = np.asarray(img, np.float32)**
112. **img = img[np.newaxis, :] / 255.**
113. **start\_time = time.time()**
114. **boxes\_, scores\_, labels\_ = sess.run([boxes, scores, labels], feed\_dict={input\_data: img})**
115. **end\_time = time.time()**
116. **if args.letterbox\_resize:**
117. **boxes\_[:, [0, 2]] = (boxes\_[:, [0, 2]] - dw) / resize\_ratio**
118. **boxes\_[:, [1, 3]] = (boxes\_[:, [1, 3]] - dh) / resize\_ratio**
119. **else:**
120. **boxes\_[:, [0, 2]] \*= (width\_ori/float(args.new\_size[0]))**
121. **boxes\_[:, [1, 3]] \*= (height\_ori/float(args.new\_size[1]))**
122. **for i in range(len(boxes\_)):**
123. **x0, y0, x1, y1 = boxes\_[i]**
124. **if scores\_[i] < 0.9:**
125. **continue**
126. **plot\_one\_box(img\_ori, [x0, y0, x1, y1], label=args.classes[labels\_[i]] + ', {:.2f}%'.format(scores\_[i] \* 100), color=color\_table[labels\_[i]])**
127. **print(x0, y0, x1, y1, args.classes[labels\_[i]])**
128. **cv2.putText(img\_ori, '{:.2f}ms'.format((end\_time - start\_time) \* 1000), (40, 40), 0,**
129. **fontScale=1, color=(0, 255, 0), thickness=2)**
130. **cv2.imshow('image', img\_ori)**
131. **if args.save\_video:**
132. **videoWriter.write(img\_ori)**
133. **if cv2.waitKey(1) & 0xFF == ord('q'):**
134. **break**
135. **vid.release()**
136. **if args.save\_video:**
137. **videoWriter.release()**
138. **from \_\_future\_\_ import division, print\_function**
139. **import tensorflow as tf**
140. **import numpy as np**
141. **import argparse**
142. **import cv2**
143. **import time**
144. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
145. **from utils.nms\_utils import gpu\_nms**
146. **from utils.plot\_utils import get\_color\_table, plot\_one\_box**
147. **from utils.data\_aug import letterbox\_resize**
148. **from model import yolov3**
149. **from tensorflow.compat.v1 import ConfigProto**
150. **from tensorflow.compat.v1 import InteractiveSession**
151. **config = ConfigProto()**
152. **config.gpu\_options.allow\_growth = True**
153. **session = InteractiveSession(config=config)**
154. **parser = argparse.ArgumentParser(description="YOLO-V3 video test procedure.")**
155. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
156. **help="The path of the anchor txt file.")**
157. **parser.add\_argument("--new\_size", nargs='\*', type=int, default=[416, 416],**
158. **help="Resize the input image with `new\_size`, size format: [width, height]")**
159. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=True,**
160. **help="Whether to use the letterbox resize.")**
161. **parser.add\_argument("--class\_name\_path", type=str, default="./data/coco.names",**
162. **help="The path of the class names.")**
163. **parser.add\_argument("--restore\_path", type=str, default="./data/darknet\_weights/yolov3.ckpt",**
164. **help="The path of the weights to restore.")**
165. **parser.add\_argument("--save\_video", type=lambda x: (str(x).lower() == 'true'), default=False,**
166. **help="Whether to save the video detection results.")**
167. **args = parser.parse\_args()**
168. **args.anchors = parse\_anchors(args.anchor\_path)**
169. **args.classes = read\_class\_names(args.class\_name\_path)**
170. **args.num\_class = len(args.classes)**
171. **color\_table = get\_color\_table(args.num\_class)**
172. **vid = cv2.VideoCapture(0)**
173. **if args.save\_video:**
174. **fourcc = cv2.VideoWriter\_fourcc('m', 'p', '4', 'v')**
175. **videoWriter = cv2.VideoWriter('video\_result.mp4', fourcc, video\_fps, (video\_width, video\_height))**
176. **with tf.Session() as sess:**
177. **input\_data = tf.placeholder(tf.float32, [1, args.new\_size[1], args.new\_size[0], 3], name='input\_data')**
178. **yolo\_model = yolov3(args.num\_class, args.anchors)**
179. **with tf.variable\_scope('yolov3'):**
180. **pred\_feature\_maps = yolo\_model.forward(input\_data, False)**
181. **pred\_boxes, pred\_confs, pred\_probs = yolo\_model.predict(pred\_feature\_maps)**
182. **pred\_scores = pred\_confs \* pred\_probs**
183. **boxes, scores, labels = gpu\_nms(pred\_boxes, pred\_scores, args.num\_class, max\_boxes=200, score\_thresh=0.3, nms\_thresh=0.45)**
184. **saver = tf.train.Saver()**
185. **saver.restore(sess, args.restore\_path)**
186. **while True:**
187. **ret, img\_ori = vid.read()**
188. **if args.letterbox\_resize:**
189. **img, resize\_ratio, dw, dh = letterbox\_resize(img\_ori, args.new\_size[0], args.new\_size[1])**
190. **else:**
191. **height\_ori, width\_ori = img\_ori.shape[:2]**
192. **img = cv2.resize(img\_ori, tuple(args.new\_size))**
193. **img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**
194. **img = np.asarray(img, np.float32)**
195. **img = img[np.newaxis, :] / 255.**
196. **start\_time = time.time()**
197. **boxes\_, scores\_, labels\_ = sess.run([boxes, scores, labels], feed\_dict={input\_data: img})**
198. **end\_time = time.time()**
199. **if args.letterbox\_resize:**
200. **boxes\_[:, [0, 2]] = (boxes\_[:, [0, 2]] - dw) / resize\_ratio**
201. **boxes\_[:, [1, 3]] = (boxes\_[:, [1, 3]] - dh) / resize\_ratio**
202. **else:**
203. **boxes\_[:, [0, 2]] \*= (width\_ori/float(args.new\_size[0]))**
204. **boxes\_[:, [1, 3]] \*= (height\_ori/float(args.new\_size[1]))**
205. **for i in range(len(boxes\_)):**
206. **x0, y0, x1, y1 = boxes\_[i]**
207. **plot\_one\_box(img\_ori, [x0, y0, x1, y1], label=args.classes[labels\_[i]] + ', {:.2f}%'.format(scores\_[i] \* 100), color=color\_table[labels\_[i]])**
208. **print(x0, y0, x1, y1, args.classes[labels\_[i]])**
209. **cv2.putText(img\_ori, '{:.2f}ms'.format((end\_time - start\_time) \* 1000), (40, 40), 0,**
210. **fontScale=1, color=(0, 255, 0), thickness=2)**
211. **cv2.imshow('image', img\_ori)**
212. **if args.save\_video:**
213. **videoWriter.write(img\_ori)**
214. **if cv2.waitKey(1) & 0xFF == ord('q'):**
215. **break**
216. **vid.release()**
217. **if args.save\_video:**
218. **videoWriter.release()**
219. **from \_\_future\_\_ import division, print\_function**
220. **import os**
221. **import sys**
222. **import tensorflow as tf**
223. **import numpy as np**
224. **from model import yolov3**
225. **from utils.misc\_utils import parse\_anchors, load\_weights**
226. **num\_class = 80**
227. **img\_size = 416**
228. **weight\_path = './data/darknet\_weights/yolov3.weights'**
229. **save\_path = './data/darknet\_weights/yolov3.ckpt'**
230. **anchors = parse\_anchors('./data/yolo\_anchors.txt')**
231. **model = yolov3(80, anchors)**
232. **with tf.Session() as sess:**
233. **inputs = tf.placeholder(tf.float32, [1, img\_size, img\_size, 3])**
234. **with tf.variable\_scope('yolov3'):**
235. **feature\_map = model.forward(inputs)**
236. **saver = tf.train.Saver(var\_list=tf.global\_variables(scope='yolov3'))**
237. **load\_ops = load\_weights(tf.global\_variables(scope='yolov3'), weight\_path)**
238. **sess.run(load\_ops)**
239. **saver.save(sess, save\_path=save\_path)**
240. **print('TensorFlow model checkpoint has been saved to {}'.format(save\_path))**
241. **from \_\_future\_\_ import division, print\_function**
242. **import tensorflow as tf**
243. **import numpy as np**
244. **import argparse**
245. **from tqdm import trange**
246. **from utils.data\_utils import get\_batch\_data**
247. **from utils.misc\_utils import parse\_anchors, read\_class\_names, AverageMeter**
248. **from utils.eval\_utils import evaluate\_on\_cpu, evaluate\_on\_gpu, get\_preds\_gpu, voc\_eval, parse\_gt\_rec**
249. **from utils.nms\_utils import gpu\_nms**
250. **from model import yolov3**
251. **parser = argparse.ArgumentParser(description="YOLO-V3 eval procedure.")**
252. **parser.add\_argument("--eval\_file", type=str, default="./data/my\_data/val.txt",**
253. **help="The path of the validation or test txt file.")**
254. **parser.add\_argument("--restore\_path", type=str, default="./data/darknet\_weights/yolov3.ckpt",**
255. **help="The path of the weights to restore.")**
256. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
257. **help="The path of the anchor txt file.")**
258. **parser.add\_argument("--class\_name\_path", type=str, default="./data/coco.names",**
259. **help="The path of the class names.")**
260. **parser.add\_argument("--img\_size", nargs='\*', type=int, default=[416, 416],**
261. **help="Resize the input image to `img\_size`, size format: [width, height]")**
262. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=False,**
263. **help="Whether to use the letterbox resize, i.e., keep the original image aspect ratio.")**
264. **parser.add\_argument("--num\_threads", type=int, default=10,**
265. **help="Number of threads for image processing used in tf.data pipeline.")**
266. **parser.add\_argument("--prefetech\_buffer", type=int, default=5,**
267. **help="Prefetech\_buffer used in tf.data pipeline.")**
268. **parser.add\_argument("--nms\_threshold", type=float, default=0.45,**
269. **help="IOU threshold in nms operation.")**
270. **parser.add\_argument("--score\_threshold", type=float, default=0.01,**
271. **help="Threshold of the probability of the classes in nms operation.")**
272. **parser.add\_argument("--nms\_topk", type=int, default=400,**
273. **help="Keep at most nms\_topk outputs after nms.")**
274. **parser.add\_argument("--use\_voc\_07\_metric", type=lambda x: (str(x).lower() == 'true'), default=False,**
275. **help="Whether to use the voc 2007 mAP metrics.")**
276. **args = parser.parse\_args()**
277. **args.anchors = parse\_anchors(args.anchor\_path)**
278. **args.classes = read\_class\_names(args.class\_name\_path)**
279. **args.class\_num = len(args.classes)**
280. **args.img\_cnt = len(open(args.eval\_file, 'r').readlines())**
281. **is\_training = tf.placeholder(dtype=tf.bool, name="phase\_train")**
282. **handle\_flag = tf.placeholder(tf.string, [], name='iterator\_handle\_flag')**
283. **pred\_boxes\_flag = tf.placeholder(tf.float32, [1, None, None])**
284. **pred\_scores\_flag = tf.placeholder(tf.float32, [1, None, None])**
285. **gpu\_nms\_op = gpu\_nms(pred\_boxes\_flag, pred\_scores\_flag, args.class\_num, args.nms\_topk, args.score\_threshold, args.nms\_threshold)**
286. **val\_dataset = tf.data.TextLineDataset(args.eval\_file)**
287. **val\_dataset = val\_dataset.batch(1)**
288. **val\_dataset = val\_dataset.map(**
289. **lambda x: tf.py\_func(get\_batch\_data, [x, args.class\_num, args.img\_size, args.anchors, 'val', False, False, args.letterbox\_resize], [tf.int64, tf.float32, tf.float32, tf.float32, tf.float32]),**
290. **num\_parallel\_calls=args.num\_threads**
291. **)**
292. **val\_dataset.prefetch(args.prefetech\_buffer)**
293. **iterator = val\_dataset.make\_one\_shot\_iterator()**
294. **image\_ids, image, y\_true\_13, y\_true\_26, y\_true\_52 = iterator.get\_next()**
295. **image\_ids.set\_shape([None])**
296. **y\_true = [y\_true\_13, y\_true\_26, y\_true\_52]**
297. **image.set\_shape([None, args.img\_size[1], args.img\_size[0], 3])**
298. **for y in y\_true:**
299. **y.set\_shape([None, None, None, None, None])**
300. **yolo\_model = yolov3(args.class\_num, args.anchors)**
301. **with tf.variable\_scope('yolov3'):**
302. **pred\_feature\_maps = yolo\_model.forward(image, is\_training=is\_training)**
303. **loss = yolo\_model.compute\_loss(pred\_feature\_maps, y\_true)**
304. **y\_pred = yolo\_model.predict(pred\_feature\_maps)**
305. **saver\_to\_restore = tf.train.Saver()**
306. **with tf.Session() as sess:**
307. **sess.run([tf.global\_variables\_initializer()])**
308. **saver\_to\_restore.restore(sess, args.restore\_path)**
309. **print('\n----------- start to eval -----------\n')**
310. **val\_loss\_total, val\_loss\_xy, val\_loss\_wh, val\_loss\_conf, val\_loss\_class = \**
311. **AverageMeter(), AverageMeter(), AverageMeter(), AverageMeter(), AverageMeter()**
312. **val\_preds = []**
313. **for j in trange(args.img\_cnt):**
314. **\_\_image\_ids, \_\_y\_pred, \_\_loss = sess.run([image\_ids, y\_pred, loss], feed\_dict={is\_training: False})**
315. **pred\_content = get\_preds\_gpu(sess, gpu\_nms\_op, pred\_boxes\_flag, pred\_scores\_flag, \_\_image\_ids, \_\_y\_pred)**
316. **val\_preds.extend(pred\_content)**
317. **val\_loss\_total.update(\_\_loss[0])**
318. **val\_loss\_xy.update(\_\_loss[1])**
319. **val\_loss\_wh.update(\_\_loss[2])**
320. **val\_loss\_conf.update(\_\_loss[3])**
321. **val\_loss\_class.update(\_\_loss[4])**
322. **rec\_total, prec\_total, ap\_total = AverageMeter(), AverageMeter(), AverageMeter()**
323. **gt\_dict = parse\_gt\_rec(args.eval\_file, args.img\_size, args.letterbox\_resize)**
324. **print('mAP eval:')**
325. **for ii in range(args.class\_num):**
326. **npos, nd, rec, prec, ap = voc\_eval(gt\_dict, val\_preds, ii, iou\_thres=0.5, use\_07\_metric=args.use\_voc\_07\_metric)**
327. **rec\_total.update(rec, npos)**
328. **prec\_total.update(prec, nd)**
329. **ap\_total.update(ap, 1)**
330. **print('Class {}: Recall: {:.4f}, Precision: {:.4f}, AP: {:.4f}'.format(ii, rec, prec, ap))**
331. **mAP = ap\_total.average**
332. **print('final mAP: {:.4f}'.format(mAP))**
333. **print("recall: {:.3f}, precision: {:.3f}".format(rec\_total.average, prec\_total.average))**
334. **print("total\_loss: {:.3f}, loss\_xy: {:.3f}, loss\_wh: {:.3f}, loss\_conf: {:.3f}, loss\_class: {:.3f}".format(**
335. **val\_loss\_total.average, val\_loss\_xy.average, val\_loss\_wh.average, val\_loss\_conf.average, val\_loss\_class.average**
336. **))**
337. **from \_\_future\_\_ import division, print\_function**
338. **import numpy as np**
339. **def iou(box, clusters):**
340. **x = np.minimum(clusters[:, 0], box[0])**
341. **y = np.minimum(clusters[:, 1], box[1])**
342. **if np.count\_nonzero(x == 0) > 0 or np.count\_nonzero(y == 0) > 0:**
343. **raise ValueError("Box has no area")**
344. **intersection = x \* y**
345. **box\_area = box[0] \* box[1]**
346. **cluster\_area = clusters[:, 0] \* clusters[:, 1]**
347. **iou\_ = np.true\_divide(intersection, box\_area + cluster\_area - intersection + 1e-10)**
348. **return iou\_**
349. **def avg\_iou(boxes, clusters):**
350. **return np.mean([np.max(iou(boxes[i], clusters)) for i in range(boxes.shape[0])])**
351. **def translate\_boxes(boxes):**
352. **new\_boxes = boxes.copy()**
353. **for row in range(new\_boxes.shape[0]):**
354. **new\_boxes[row][2] = np.abs(new\_boxes[row][2] - new\_boxes[row][0])**
355. **new\_boxes[row][3] = np.abs(new\_boxes[row][3] - new\_boxes[row][1])**
356. **return np.delete(new\_boxes, [0, 1], axis=1)**
357. **def kmeans(boxes, k, dist=np.median):**
358. **rows = boxes.shape[0]**
359. **distances = np.empty((rows, k))**
360. **last\_clusters = np.zeros((rows,))**
361. **np.random.seed()**
362. **clusters = boxes[np.random.choice(rows, k, replace=False)]**
363. **while True:**
364. **for row in range(rows):**
365. **distances[row] = 1 - iou(boxes[row], clusters)**
366. **nearest\_clusters = np.argmin(distances, axis=1)**
367. **if (last\_clusters == nearest\_clusters).all():**
368. **break**
369. **for cluster in range(k):**
370. **clusters[cluster] = dist(boxes[nearest\_clusters == cluster], axis=0)**
371. **last\_clusters = nearest\_clusters**
372. **return clusters**
373. **def parse\_anno(annotation\_path, target\_size=None):**
374. **anno = open(annotation\_path, 'r')**
375. **result = []**
376. **for line in anno:**
377. **s = line.strip().split(' ')**
378. **img\_w = int(s[2])**
379. **img\_h = int(s[3])**
380. **s = s[4:]**
381. **box\_cnt = len(s) // 5**
382. **for i in range(box\_cnt):**
383. **x\_min, y\_min, x\_max, y\_max = float(s[i\*5+1]), float(s[i\*5+2]), float(s[i\*5+3]), float(s[i\*5+4])**
384. **width = x\_max - x\_min**
385. **height = y\_max - y\_min**
386. **assert width > 0**
387. **assert height > 0**
388. **if target\_size is not None:**
389. **resize\_ratio = min(target\_size[0] / img\_w, target\_size[1] / img\_h)**
390. **width \*= resize\_ratio**
391. **height \*= resize\_ratio**
392. **result.append([width, height])**
393. **else:**
394. **result.append([width, height])**
395. **result = np.asarray(result)**
396. **return result**
397. **def get\_kmeans(anno, cluster\_num=9):**
398. **anchors = kmeans(anno, cluster\_num)**
399. **ave\_iou = avg\_iou(anno, anchors)**
400. **anchors = anchors.astype('int').tolist()**
401. **anchors = sorted(anchors, key=lambda x: x[0] \* x[1])**
402. **return anchors, ave\_iou**
403. **if \_\_name\_\_ == '\_\_main\_\_':**
404. **target\_size = [416, 416]**
405. **annotation\_path = "train.txt"**
406. **anno\_result = parse\_anno(annotation\_path, target\_size=target\_size)**
407. **anchors, ave\_iou = get\_kmeans(anno\_result, 9)**
408. **anchor\_string = ''**
409. **for anchor in anchors:**
410. **anchor\_string += '{},{}, '.format(anchor[0], anchor[1])**
411. **anchor\_string = anchor\_string[:-2]**
412. **print('anchors are:')**
413. **print(anchor\_string)**
414. **print('the average iou is:')**
415. **print(ave\_iou)**
416. **from \_\_future\_\_ import division, print\_function**
417. **import tensorflow as tf**
418. **slim = tf.contrib.slim**
419. **from utils.layer\_utils import conv2d, darknet53\_body, yolo\_block, upsample\_layer**
420. **class yolov3(object):**
421. **def \_\_init\_\_(self, class\_num, anchors, use\_label\_smooth=False, use\_focal\_loss=False, batch\_norm\_decay=0.999,**
422. **weight\_decay=5e-4, use\_static\_shape=True):**
423. **self.class\_num = class\_num**
424. **self.anchors = anchors**
425. **self.batch\_norm\_decay = batch\_norm\_decay**
426. **self.use\_label\_smooth = use\_label\_smooth**
427. **self.use\_focal\_loss = use\_focal\_loss**
428. **self.weight\_decay = weight\_decay**
429. **self.use\_static\_shape = use\_static\_shape**
430. **def forward(self, inputs, is\_training=False, reuse=False):**
431. **self.img\_size = tf.shape(inputs)[1:3]**
432. **batch\_norm\_params = {**
433. **'decay': self.batch\_norm\_decay,**
434. **'epsilon': 1e-05,**
435. **'scale': True,**
436. **'is\_training': is\_training,**
437. **'fused': None,**
438. **}**
439. **with slim.arg\_scope([slim.conv2d, slim.batch\_norm], reuse=reuse):**
440. **with slim.arg\_scope([slim.conv2d],**
441. **normalizer\_fn=slim.batch\_norm,**
442. **normalizer\_params=batch\_norm\_params,**
443. **biases\_initializer=None,**
444. **activation\_fn=lambda x: tf.nn.leaky\_relu(x, alpha=0.1),**
445. **weights\_regularizer=slim.l2\_regularizer(self.weight\_decay)):**
446. **with tf.variable\_scope('darknet53\_body'):**
447. **route\_1, route\_2, route\_3 = darknet53\_body(inputs)**
448. **with tf.variable\_scope('yolov3\_head'):**
449. **inter1, net = yolo\_block(route\_3, 512)**
450. **feature\_map\_1 = slim.conv2d(net, 3 \* (5 + self.class\_num), 1,**
451. **stride=1, normalizer\_fn=None,**
452. **activation\_fn=None, biases\_initializer=tf.zeros\_initializer())**
453. **feature\_map\_1 = tf.identity(feature\_map\_1, name='feature\_map\_1')**
454. **inter1 = conv2d(inter1, 256, 1)**
455. **inter1 = upsample\_layer(inter1,**
456. **route\_2.get\_shape().as\_list() if self.use\_static\_shape else tf.shape(**
457. **route\_2))**
458. **concat1 = tf.concat([inter1, route\_2], axis=3)**
459. **inter2, net = yolo\_block(concat1, 256)**
460. **feature\_map\_2 = slim.conv2d(net, 3 \* (5 + self.class\_num), 1,**
461. **stride=1, normalizer\_fn=None,**
462. **activation\_fn=None, biases\_initializer=tf.zeros\_initializer())**
463. **feature\_map\_2 = tf.identity(feature\_map\_2, name='feature\_map\_2')**
464. **inter2 = conv2d(inter2, 128, 1)**
465. **inter2 = upsample\_layer(inter2,**
466. **route\_1.get\_shape().as\_list() if self.use\_static\_shape else tf.shape(**
467. **route\_1))**
468. **concat2 = tf.concat([inter2, route\_1], axis=3)**
469. **\_, feature\_map\_3 = yolo\_block(concat2, 128)**
470. **feature\_map\_3 = slim.conv2d(feature\_map\_3, 3 \* (5 + self.class\_num), 1,**
471. **stride=1, normalizer\_fn=None,**
472. **activation\_fn=None, biases\_initializer=tf.zeros\_initializer())**
473. **feature\_map\_3 = tf.identity(feature\_map\_3, name='feature\_map\_3')**
474. **return feature\_map\_1, feature\_map\_2, feature\_map\_3**
475. **def reorg\_layer(self, feature\_map, anchors):**
476. **grid\_size = feature\_map.get\_shape().as\_list()[1:3] if self.use\_static\_shape else tf.shape(feature\_map)[**
477. **1:3]**
478. **ratio = tf.cast(self.img\_size / grid\_size, tf.float32)**
479. **rescaled\_anchors = [(anchor[0] / ratio[1], anchor[1] / ratio[0]) for anchor in anchors]**
480. **feature\_map = tf.reshape(feature\_map, [-1, grid\_size[0], grid\_size[1], 3, 5 + self.class\_num])**
481. **box\_centers, box\_sizes, conf\_logits, prob\_logits = tf.split(feature\_map, [2, 2, 1, self.class\_num], axis=-1)**
482. **box\_centers = tf.nn.sigmoid(box\_centers)**
483. **grid\_x = tf.range(grid\_size[1], dtype=tf.int32)**
484. **grid\_y = tf.range(grid\_size[0], dtype=tf.int32)**
485. **grid\_x, grid\_y = tf.meshgrid(grid\_x, grid\_y)**
486. **x\_offset = tf.reshape(grid\_x, (-1, 1))**
487. **y\_offset = tf.reshape(grid\_y, (-1, 1))**
488. **x\_y\_offset = tf.concat([x\_offset, y\_offset], axis=-1)**
489. **x\_y\_offset = tf.cast(tf.reshape(x\_y\_offset, [grid\_size[0], grid\_size[1], 1, 2]), tf.float32)**
490. **box\_centers = box\_centers + x\_y\_offset**
491. **box\_centers = box\_centers \* ratio[::-1]**
492. **box\_sizes = tf.exp(box\_sizes) \* rescaled\_anchors**
493. **box\_sizes = box\_sizes \* ratio[::-1]**
494. **boxes = tf.concat([box\_centers, box\_sizes], axis=-1)**
495. **return x\_y\_offset, boxes, conf\_logits, prob\_logits**
496. **def predict(self, feature\_maps):**
497. **feature\_map\_1, feature\_map\_2, feature\_map\_3 = feature\_maps**
498. **feature\_map\_anchors = [(feature\_map\_1, self.anchors[6:9]),**
499. **(feature\_map\_2, self.anchors[3:6]),**
500. **(feature\_map\_3, self.anchors[0:3])]**
501. **reorg\_results = [self.reorg\_layer(feature\_map, anchors) for (feature\_map, anchors) in feature\_map\_anchors]**
502. **def \_reshape(result):**
503. **x\_y\_offset, boxes, conf\_logits, prob\_logits = result**
504. **grid\_size = x\_y\_offset.get\_shape().as\_list()[:2] if self.use\_static\_shape else tf.shape(x\_y\_offset)[:2]**
505. **boxes = tf.reshape(boxes, [-1, grid\_size[0] \* grid\_size[1] \* 3, 4])**
506. **conf\_logits = tf.reshape(conf\_logits, [-1, grid\_size[0] \* grid\_size[1] \* 3, 1])**
507. **prob\_logits = tf.reshape(prob\_logits, [-1, grid\_size[0] \* grid\_size[1] \* 3, self.class\_num])**
508. **return boxes, conf\_logits, prob\_logits**
509. **boxes\_list, confs\_list, probs\_list = [], [], []**
510. **for result in reorg\_results:**
511. **boxes, conf\_logits, prob\_logits = \_reshape(result)**
512. **confs = tf.sigmoid(conf\_logits)**
513. **probs = tf.sigmoid(prob\_logits)**
514. **boxes\_list.append(boxes)**
515. **confs\_list.append(confs)**
516. **probs\_list.append(probs)**
517. **boxes = tf.concat(boxes\_list, axis=1)**
518. **confs = tf.concat(confs\_list, axis=1)**
519. **probs = tf.concat(probs\_list, axis=1)**
520. **center\_x, center\_y, width, height = tf.split(boxes, [1, 1, 1, 1], axis=-1)**
521. **x\_min = center\_x - width / 2**
522. **y\_min = center\_y - height / 2**
523. **x\_max = center\_x + width / 2**
524. **y\_max = center\_y + height / 2**
525. **boxes = tf.concat([x\_min, y\_min, x\_max, y\_max], axis=-1)**
526. **return boxes, confs, probs**
527. **def loss\_layer(self, feature\_map\_i, y\_true, anchors):**
528. **grid\_size = tf.shape(feature\_map\_i)[1:3]**
529. **ratio = tf.cast(self.img\_size / grid\_size, tf.float32)**
530. **N = tf.cast(tf.shape(feature\_map\_i)[0], tf.float32)**
531. **x\_y\_offset, pred\_boxes, pred\_conf\_logits, pred\_prob\_logits = self.reorg\_layer(feature\_map\_i, anchors)**
532. **object\_mask = y\_true[..., 4:5]**
533. **ignore\_mask = tf.TensorArray(tf.float32, size=0, dynamic\_size=True)**
534. **def loop\_cond(idx, ignore\_mask):**
535. **return tf.less(idx, tf.cast(N, tf.int32))**
536. **def loop\_body(idx, ignore\_mask):**
537. **valid\_true\_boxes = tf.boolean\_mask(y\_true[idx, ..., 0:4], tf.cast(object\_mask[idx, ..., 0], 'bool'))**
538. **iou = self.box\_iou(pred\_boxes[idx], valid\_true\_boxes)**
539. **best\_iou = tf.reduce\_max(iou, axis=-1)**
540. **ignore\_mask\_tmp = tf.cast(best\_iou < 0.5, tf.float32)**
541. **ignore\_mask = ignore\_mask.write(idx, ignore\_mask\_tmp)**
542. **return idx + 1, ignore\_mask**
543. **\_, ignore\_mask = tf.while\_loop(cond=loop\_cond, body=loop\_body, loop\_vars=[0, ignore\_mask])**
544. **ignore\_mask = ignore\_mask.stack()**
545. **ignore\_mask = tf.expand\_dims(ignore\_mask, -1)**
546. **pred\_box\_xy = pred\_boxes[..., 0:2]**
547. **pred\_box\_wh = pred\_boxes[..., 2:4]**
548. **true\_xy = y\_true[..., 0:2] / ratio[::-1] - x\_y\_offset**
549. **pred\_xy = pred\_box\_xy / ratio[::-1] - x\_y\_offset**
550. **true\_tw\_th = y\_true[..., 2:4] / anchors**
551. **pred\_tw\_th = pred\_box\_wh / anchors**
552. **true\_tw\_th = tf.where(condition=tf.equal(true\_tw\_th, 0),**
553. **x=tf.ones\_like(true\_tw\_th), y=true\_tw\_th)**
554. **pred\_tw\_th = tf.where(condition=tf.equal(pred\_tw\_th, 0),**
555. **x=tf.ones\_like(pred\_tw\_th), y=pred\_tw\_th)**
556. **true\_tw\_th = tf.log(tf.clip\_by\_value(true\_tw\_th, 1e-9, 1e9))**
557. **pred\_tw\_th = tf.log(tf.clip\_by\_value(pred\_tw\_th, 1e-9, 1e9))**
558. **box\_loss\_scale = 2. - (y\_true[..., 2:3] / tf.cast(self.img\_size[1], tf.float32)) \* (**
559. **y\_true[..., 3:4] / tf.cast(self.img\_size[0], tf.float32))**
560. **mix\_w = y\_true[..., -1:]**
561. **xy\_loss = tf.reduce\_sum(tf.square(true\_xy - pred\_xy) \* object\_mask \* box\_loss\_scale \* mix\_w) / N**
562. **wh\_loss = tf.reduce\_sum(tf.square(true\_tw\_th - pred\_tw\_th) \* object\_mask \* box\_loss\_scale \* mix\_w) / N**
563. **conf\_pos\_mask = object\_mask**
564. **conf\_neg\_mask = (1 - object\_mask) \* ignore\_mask**
565. **conf\_loss\_pos = conf\_pos\_mask \* tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=object\_mask,**
566. **logits=pred\_conf\_logits)**
567. **conf\_loss\_neg = conf\_neg\_mask \* tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=object\_mask,**
568. **logits=pred\_conf\_logits)**
569. **conf\_loss = conf\_loss\_pos + conf\_loss\_neg**
570. **if self.use\_focal\_loss:**
571. **alpha = 1.0**
572. **gamma = 2.0**
573. **focal\_mask = alpha \* tf.pow(tf.abs(object\_mask - tf.sigmoid(pred\_conf\_logits)), gamma)**
574. **conf\_loss \*= focal\_mask**
575. **conf\_loss = tf.reduce\_sum(conf\_loss \* mix\_w) / N**
576. **if self.use\_label\_smooth:**
577. **delta = 0.01**
578. **label\_target = (1 - delta) \* y\_true[..., 5:-1] + delta \* 1. / self.class\_num**
579. **else:**
580. **label\_target = y\_true[..., 5:-1]**
581. **class\_loss = object\_mask \* tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=label\_target,**
582. **logits=pred\_prob\_logits) \* mix\_w**
583. **class\_loss = tf.reduce\_sum(class\_loss) / N**
584. **return xy\_loss, wh\_loss, conf\_loss, class\_loss**
585. **def box\_iou(self, pred\_boxes, valid\_true\_boxes):**
586. **pred\_box\_xy = pred\_boxes[..., 0:2]**
587. **pred\_box\_wh = pred\_boxes[..., 2:4]**
588. **pred\_box\_xy = tf.expand\_dims(pred\_box\_xy, -2)**
589. **pred\_box\_wh = tf.expand\_dims(pred\_box\_wh, -2)**
590. **true\_box\_xy = valid\_true\_boxes[:, 0:2]**
591. **true\_box\_wh = valid\_true\_boxes[:, 2:4]**
592. **intersect\_mins = tf.maximum(pred\_box\_xy - pred\_box\_wh / 2.,**
593. **true\_box\_xy - true\_box\_wh / 2.)**
594. **intersect\_maxs = tf.minimum(pred\_box\_xy + pred\_box\_wh / 2.,**
595. **true\_box\_xy + true\_box\_wh / 2.)**
596. **intersect\_wh = tf.maximum(intersect\_maxs - intersect\_mins, 0.)**
597. **intersect\_area = intersect\_wh[..., 0] \* intersect\_wh[..., 1]**
598. **pred\_box\_area = pred\_box\_wh[..., 0] \* pred\_box\_wh[..., 1]**
599. **true\_box\_area = true\_box\_wh[..., 0] \* true\_box\_wh[..., 1]**
600. **true\_box\_area = tf.expand\_dims(true\_box\_area, axis=0)**
601. **iou = intersect\_area / (pred\_box\_area + true\_box\_area - intersect\_area + 1e-10)**
602. **return iou**
603. **def compute\_loss(self, y\_pred, y\_true):**
604. **loss\_xy, loss\_wh, loss\_conf, loss\_class = 0., 0., 0., 0.**
605. **anchor\_group = [self.anchors[6:9], self.anchors[3:6], self.anchors[0:3]]**
606. **for i in range(len(y\_pred)):**
607. **result = self.loss\_layer(y\_pred[i], y\_true[i], anchor\_group[i])**
608. **loss\_xy += result[0]**
609. **loss\_wh += result[1]**
610. **loss\_conf += result[2]**
611. **loss\_class += result[3]**
612. **total\_loss = loss\_xy + loss\_wh + loss\_conf + loss\_class**
613. **return [total\_loss, loss\_xy, loss\_wh, loss\_conf, loss\_class]**
614. **from \_\_future\_\_ import division, print\_function**
615. **import tensorflow as tf**
616. **import numpy as np**
617. **import argparse**
618. **import cv2**
619. **import time**
620. **import socket**
621. **import serial**
622. **import threading**
623. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
624. **from utils.nms\_utils import gpu\_nms**
625. **from utils.plot\_utils import get\_color\_table, plot\_one\_box**
626. **from utils.data\_aug import letterbox\_resize**
627. **from model import yolov3**
628. **from tensorflow.compat.v1 import ConfigProto**
629. **from tensorflow.compat.v1 import InteractiveSession**
630. **SERVER\_IP = "192.168.3.210"**
631. **SERVER\_PORT = 8888**
632. **print("Starting socket: TCP...")**
633. **server\_addr = (SERVER\_IP, SERVER\_PORT)**
634. **socket\_tcp = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**
635. **frame = None**
636. **while True:**
637. **try:**
638. **print("Connecting to server @ %s:%d..." % (SERVER\_IP, SERVER\_PORT))**
639. **socket\_tcp.connect(server\_addr)**
640. **break**
641. **except Exception:**
642. **print("Can't connect to server,try it latter!")**
643. **time.sleep(1)**
644. **continue**
645. **print("Arm init ...")**
646. **init ="init"**
647. **socket\_tcp.send(init.encode("utf-8"))**
648. **while True:**
649. **data = socket\_tcp.recv(512)**
650. **if len(data) > 3:**
651. **print(data)**
652. **break**
653. **arduino = serial.Serial("/dev/ttyACM1",115200,timeout = 0.0001)**
654. **time.sleep(1.5)**
655. **print("arduino connecting")**
656. **def arduinoSerial():**
657. **arduino.write("45".encode('utf-8'))**
658. **time.sleep(0.5)**
659. **def socketsend(string):**
660. **socket\_tcp.send(string.encode("utf-8"))**
661. **def socketrecv():**
662. **while True:**
663. **data = socket\_tcp.recv(512)**
664. **if data == b"finish":**
665. **break**
666. **elif len(data) > 3:**
667. **print(data)**
668. **while True:**
669. **time.sleep(5)**
670. **print("raspberry serial error")**
671. **fruit = ""**
672. **state = 0**
673. **arduino\_state = 1**
674. **print("123333333333333333333333")**
675. **def socketSerial():**
676. **global fruit,state,arduino\_state**
677. **print("123333333333333333333333")**
678. **while True:**
679. **time.sleep(0.01)**
680. **if fruit != "" and state == 0:**
681. **state = 1**
682. **arduino\_state = 1**
683. **socket\_tcp.send(fruit.encode("utf-8"))**
684. **while True:**
685. **data = socket\_tcp.recv(512)**
686. **if data == b"finish":**
687. **fruit = ""**
688. **state = 0**
689. **arduino\_state = 0**
690. **break**
691. **elif len(data) > 3:**
692. **print(data)**
693. **while True:**
694. **time.sleep(5)**
695. **print("raspberry serial error")**
696. **t = threading.Thread(target=socketSerial, args=())**
697. **t.start()**
698. **def ArduinoSerial():**
699. **while True:**
700. **time.sleep(0.5)**
701. **if arduino\_state == 0 :**
702. **arduinoSerial()**
703. **t1 = threading.Thread(target=ArduinoSerial, args=())**
704. **t1.start()**
705. **config = ConfigProto()**
706. **config.gpu\_options.allow\_growth = True**
707. **session = InteractiveSession(config=config)**
708. **parser = argparse.ArgumentParser(description="YOLO-V3 video test procedure.")**
709. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
710. **help="The path of the anchor txt file.")**
711. **parser.add\_argument("--new\_size", nargs='\*', type=int, default=[416, 416],**
712. **help="Resize the input image with `new\_size`, size format: [width, height]")**
713. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=True,**
714. **help="Whether to use the letterbox resize.")**
715. **parser.add\_argument("--class\_name\_path", type=str, default="./data/voc\_names.txt",**
716. **help="The path of the class names.")**
717. **parser.add\_argument("--restore\_path", type=str, default="./checkpoint/model-epoch\_90\_step\_276639\_loss\_0.5149\_lr\_1e-05",**
718. **help="The path of the weights to restore.")**
719. **parser.add\_argument("--save\_video", type=lambda x: (str(x).lower() == 'true'), default=False,**
720. **help="Whether to save the video detection results.")**
721. **args = parser.parse\_args()**
722. **args.anchors = parse\_anchors(args.anchor\_path)**
723. **args.classes = read\_class\_names(args.class\_name\_path)**
724. **args.num\_class = len(args.classes)**
725. **color\_table = get\_color\_table(args.num\_class)**
726. **vid = cv2.VideoCapture(2)**
727. **if args.save\_video:**
728. **fourcc = cv2.VideoWriter\_fourcc('m', 'p', '4', 'v')**
729. **videoWriter = cv2.VideoWriter('video\_result.mp4', fourcc, video\_fps, (video\_width, video\_height))**
730. **with tf.Session() as sess:**
731. **input\_data = tf.placeholder(tf.float32, [1, args.new\_size[1], args.new\_size[0], 3], name='input\_data')**
732. **yolo\_model = yolov3(args.num\_class, args.anchors)**
733. **with tf.variable\_scope('yolov3'):**
734. **pred\_feature\_maps = yolo\_model.forward(input\_data, False)**
735. **pred\_boxes, pred\_confs, pred\_probs = yolo\_model.predict(pred\_feature\_maps)**
736. **pred\_scores = pred\_confs \* pred\_probs**
737. **boxes, scores, labels = gpu\_nms(pred\_boxes, pred\_scores, args.num\_class, max\_boxes=200, score\_thresh=0.3, nms\_thresh=0.45)**
738. **saver = tf.train.Saver()**
739. **saver.restore(sess, args.restore\_path)**
740. **arduino\_state = 0**
741. **while True:**
742. **ret, img\_ori = vid.read()**
743. **if args.letterbox\_resize:**
744. **img, resize\_ratio, dw, dh = letterbox\_resize(img\_ori, args.new\_size[0], args.new\_size[1])**
745. **else:**
746. **height\_ori, width\_ori = img\_ori.shape[:2]**
747. **img = cv2.resize(img\_ori, tuple(args.new\_size))**
748. **img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**
749. **img = np.asarray(img, np.float32)**
750. **img = img[np.newaxis, :] / 255.**
751. **start\_time = time.time()**
752. **boxes\_, scores\_, labels\_ = sess.run([boxes, scores, labels], feed\_dict={input\_data: img})**
753. **end\_time = time.time()**
754. **if args.letterbox\_resize:**
755. **boxes\_[:, [0, 2]] = (boxes\_[:, [0, 2]] - dw) / resize\_ratio**
756. **boxes\_[:, [1, 3]] = (boxes\_[:, [1, 3]] - dh) / resize\_ratio**
757. **else:**
758. **boxes\_[:, [0, 2]] \*= (width\_ori/float(args.new\_size[0]))**
759. **boxes\_[:, [1, 3]] \*= (height\_ori/float(args.new\_size[1]))**
760. **for i in range(len(boxes\_)):**
761. **x0, y0, x1, y1 = boxes\_[i]**
762. **if scores\_[i] < 0.6:**
763. **continue**
764. **c00 = (x0 + x1) / 2**
765. **c01 = (y0 + y1) / 2**
766. **if c00 > 430 and c00 < 540 and c01 > 220 and c01 < 310:**
767. **print(x0, y0, x1, y1, args.classes[labels\_[i]])**
768. **if state == 0:**
769. **fruit =str(args.classes[labels\_[i]])**
770. **print(fruit,state)**
771. **arduino\_state = 1**
772. **plot\_one\_box(img\_ori, [x0, y0, x1, y1], label=args.classes[labels\_[i]] + ', {:.2f}%'.format(scores\_[i] \* 100), color=color\_table[labels\_[i]])**
773. **cv2.putText(img\_ori, '{:.2f}ms'.format((end\_time - start\_time) \* 1000), (40, 40), 0,**
774. **fontScale=1, color=(0, 255, 0), thickness=2)**
775. **cv2.imshow('image', img\_ori)**
776. **if args.save\_video:**
777. **videoWriter.write(img\_ori)**
778. **if cv2.waitKey(1) & 0xFF == ord('q'):**
779. **break**
780. **vid.release()**
781. **if args.save\_video:**
782. **videoWriter.release()**
783. **from \_\_future\_\_ import division, print\_function**
784. **import tensorflow as tf**
785. **import numpy as np**
786. **import argparse**
787. **import cv2**
788. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
789. **from utils.nms\_utils import gpu\_nms**
790. **from utils.plot\_utils import get\_color\_table, plot\_one\_box**
791. **from utils.data\_aug import letterbox\_resize**
792. **from model import yolov3**
793. **parser = argparse.ArgumentParser(description="YOLO-V3 test single image test procedure.")**
794. **parser.add\_argument("input\_image", type=str,**
795. **help="The path of the input image.")**
796. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
797. **help="The path of the anchor txt file.")**
798. **parser.add\_argument("--new\_size", nargs='\*', type=int, default=[416, 416],**
799. **help="Resize the input image with `new\_size`, size format: [width, height]")**
800. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=True,**
801. **help="Whether to use the letterbox resize.")**
802. **parser.add\_argument("--class\_name\_path", type=str, default="./data/voc\_names.txt",**
803. **help="The path of the class names.")**
804. **parser.add\_argument("--restore\_path", type=str, default="./checkpoint/best\_model\_Epoch\_8\_step\_45620\_mAP\_0.9996\_loss\_0.1040\_lr\_0.0001",**
805. **help="The path of the weights to restore.")**
806. **args = parser.parse\_args()**
807. **args.anchors = parse\_anchors(args.anchor\_path)**
808. **args.classes = read\_class\_names(args.class\_name\_path)**
809. **args.num\_class = len(args.classes)**
810. **color\_table = get\_color\_table(args.num\_class)**
811. **img\_ori = cv2.imread(args.input\_image)**
812. **if args.letterbox\_resize:**
813. **img, resize\_ratio, dw, dh = letterbox\_resize(img\_ori, args.new\_size[0], args.new\_size[1])**
814. **else:**
815. **height\_ori, width\_ori = img\_ori.shape[:2]**
816. **img = cv2.resize(img\_ori, tuple(args.new\_size))**
817. **img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**
818. **img = np.asarray(img, np.float32)**
819. **img = img[np.newaxis, :] / 255.**
820. **with tf.Session() as sess:**
821. **input\_data = tf.placeholder(tf.float32, [1, args.new\_size[1], args.new\_size[0], 3], name='input\_data')**
822. **yolo\_model = yolov3(args.num\_class, args.anchors)**
823. **with tf.variable\_scope('yolov3'):**
824. **pred\_feature\_maps = yolo\_model.forward(input\_data, False)**
825. **pred\_boxes, pred\_confs, pred\_probs = yolo\_model.predict(pred\_feature\_maps)**
826. **pred\_scores = pred\_confs \* pred\_probs**
827. **boxes, scores, labels = gpu\_nms(pred\_boxes, pred\_scores, args.num\_class, max\_boxes=200, score\_thresh=0.3, nms\_thresh=0.45)**
828. **saver = tf.train.Saver()**
829. **saver.restore(sess, args.restore\_path)**
830. **boxes\_, scores\_, labels\_ = sess.run([boxes, scores, labels], feed\_dict={input\_data: img})**
831. **if args.letterbox\_resize:**
832. **boxes\_[:, [0, 2]] = (boxes\_[:, [0, 2]] - dw) / resize\_ratio**
833. **boxes\_[:, [1, 3]] = (boxes\_[:, [1, 3]] - dh) / resize\_ratio**
834. **else:**
835. **boxes\_[:, [0, 2]] \*= (width\_ori/float(args.new\_size[0]))**
836. **boxes\_[:, [1, 3]] \*= (height\_ori/float(args.new\_size[1]))**
837. **print("box coords:")**
838. **print(boxes\_)**
839. **print('\*' \* 30)**
840. **print("scores:")**
841. **print(scores\_)**
842. **print('\*' \* 30)**
843. **print("labels:")**
844. **print(labels\_)**
845. **for i in range(len(boxes\_)):**
846. **x0, y0, x1, y1 = boxes\_[i]**
847. **plot\_one\_box(img\_ori, [x0, y0, x1, y1], label=args.classes[labels\_[i]] + ', {:.2f}%'.format(scores\_[i] \* 100), color=color\_table[labels\_[i]])**
848. **cv2.imshow('Detection result', img\_ori)**
849. **cv2.imwrite('detection\_result.jpg', img\_ori)**
850. **cv2.waitKey(0)**
851. **from \_\_future\_\_ import division, print\_function**
852. **import tensorflow as tf**
853. **import numpy as np**
854. **import logging**
855. **from tqdm import trange**
856. **import args**
857. **from utils.data\_utils import get\_batch\_data**
858. **from utils.misc\_utils import shuffle\_and\_overwrite, make\_summary, config\_learning\_rate, config\_optimizer, AverageMeter**
859. **from utils.eval\_utils import evaluate\_on\_cpu, evaluate\_on\_gpu, get\_preds\_gpu, voc\_eval, parse\_gt\_rec**
860. **from utils.nms\_utils import gpu\_nms**
861. **from model import yolov3**
862. **from tensorflow.compat.v1 import ConfigProto**
863. **from tensorflow.compat.v1 import InteractiveSession**
864. **config = ConfigProto()**
865. **config.gpu\_options.allow\_growth = True**
866. **session = InteractiveSession(config=config)**
867. **logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(levelname)s %(message)s',**
868. **datefmt='%a, %d %b %Y %H:%M:%S', filename=args.progress\_log\_path, filemode='w')**
869. **is\_training = tf.placeholder(tf.bool, name="phase\_train")**
870. **handle\_flag = tf.placeholder(tf.string, [], name='iterator\_handle\_flag')**
871. **pred\_boxes\_flag = tf.placeholder(tf.float32, [1, None, None])**
872. **pred\_scores\_flag = tf.placeholder(tf.float32, [1, None, None])**
873. **gpu\_nms\_op = gpu\_nms(pred\_boxes\_flag, pred\_scores\_flag, args.class\_num, args.nms\_topk, args.score\_threshold, args.nms\_threshold)**
874. **train\_dataset = tf.data.TextLineDataset(args.train\_file)**
875. **train\_dataset = train\_dataset.shuffle(args.train\_img\_cnt)**
876. **train\_dataset = train\_dataset.batch(args.batch\_size)**
877. **train\_dataset = train\_dataset.map(**
878. **lambda x: tf.py\_func(get\_batch\_data,**
879. **inp=[x, args.class\_num, args.img\_size, args.anchors, 'train', args.multi\_scale\_train, args.use\_mix\_up, args.letterbox\_resize],**
880. **Tout=[tf.int64, tf.float32, tf.float32, tf.float32, tf.float32]),**
881. **num\_parallel\_calls=args.num\_threads**
882. **)**
883. **train\_dataset = train\_dataset.prefetch(args.prefetech\_buffer)**
884. **val\_dataset = tf.data.TextLineDataset(args.val\_file)**
885. **val\_dataset = val\_dataset.batch(1)**
886. **val\_dataset = val\_dataset.map(**
887. **lambda x: tf.py\_func(get\_batch\_data,**
888. **inp=[x, args.class\_num, args.img\_size, args.anchors, 'val', False, False, args.letterbox\_resize],**
889. **Tout=[tf.int64, tf.float32, tf.float32, tf.float32, tf.float32]),**
890. **num\_parallel\_calls=args.num\_threads**
891. **)**
892. **val\_dataset.prefetch(args.prefetech\_buffer)**
893. **iterator = tf.data.Iterator.from\_structure(train\_dataset.output\_types, train\_dataset.output\_shapes)**
894. **train\_init\_op = iterator.make\_initializer(train\_dataset)**
895. **val\_init\_op = iterator.make\_initializer(val\_dataset)**
896. **image\_ids, image, y\_true\_13, y\_true\_26, y\_true\_52 = iterator.get\_next()**
897. **y\_true = [y\_true\_13, y\_true\_26, y\_true\_52]**
898. **image\_ids.set\_shape([None])**
899. **image.set\_shape([None, None, None, 3])**
900. **for y in y\_true:**
901. **y.set\_shape([None, None, None, None, None])**
902. **yolo\_model = yolov3(args.class\_num, args.anchors, args.use\_label\_smooth, args.use\_focal\_loss, args.batch\_norm\_decay, args.weight\_decay, use\_static\_shape=False)**
903. **with tf.variable\_scope('yolov3'):**
904. **pred\_feature\_maps = yolo\_model.forward(image, is\_training=is\_training)**
905. **loss = yolo\_model.compute\_loss(pred\_feature\_maps, y\_true)**
906. **y\_pred = yolo\_model.predict(pred\_feature\_maps)**
907. **l2\_loss = tf.losses.get\_regularization\_loss()**
908. **saver\_to\_restore = tf.train.Saver(var\_list=tf.contrib.framework.get\_variables\_to\_restore(include=args.restore\_include, exclude=args.restore\_exclude))**
909. **update\_vars = tf.contrib.framework.get\_variables\_to\_restore(include=args.update\_part)**
910. **tf.summary.scalar('train\_batch\_statistics/total\_loss', loss[0])**
911. **tf.summary.scalar('train\_batch\_statistics/loss\_xy', loss[1])**
912. **tf.summary.scalar('train\_batch\_statistics/loss\_wh', loss[2])**
913. **tf.summary.scalar('train\_batch\_statistics/loss\_conf', loss[3])**
914. **tf.summary.scalar('train\_batch\_statistics/loss\_class', loss[4])**
915. **tf.summary.scalar('train\_batch\_statistics/loss\_l2', l2\_loss)**
916. **tf.summary.scalar('train\_batch\_statistics/loss\_ratio', l2\_loss / loss[0])**
917. **global\_step = tf.Variable(float(args.global\_step), trainable=False, collections=[tf.GraphKeys.LOCAL\_VARIABLES])**
918. **if args.use\_warm\_up:**
919. **learning\_rate = tf.cond(tf.less(global\_step, args.train\_batch\_num \* args.warm\_up\_epoch),**
920. **lambda: args.learning\_rate\_init \* global\_step / (args.train\_batch\_num \* args.warm\_up\_epoch),**
921. **lambda: config\_learning\_rate(args, global\_step - args.train\_batch\_num \* args.warm\_up\_epoch))**
922. **else:**
923. **learning\_rate = config\_learning\_rate(args, global\_step)**
924. **tf.summary.scalar('learning\_rate', learning\_rate)**
925. **if not args.save\_optimizer:**
926. **saver\_to\_save = tf.train.Saver()**
927. **saver\_best = tf.train.Saver()**
928. **optimizer = config\_optimizer(args.optimizer\_name, learning\_rate)**
929. **update\_ops = tf.get\_collection(tf.GraphKeys.UPDATE\_OPS)**
930. **with tf.control\_dependencies(update\_ops):**
931. **gvs = optimizer.compute\_gradients(loss[0] + l2\_loss, var\_list=update\_vars)**
932. **clip\_grad\_var = [gv if gv[0] is None else [**
933. **tf.clip\_by\_norm(gv[0], 100.), gv[1]] for gv in gvs]**
934. **train\_op = optimizer.apply\_gradients(clip\_grad\_var, global\_step=global\_step)**
935. **if args.save\_optimizer:**
936. **print('Saving optimizer parameters to checkpoint! Remember to restore the global\_step in the fine-tuning afterwards.')**
937. **saver\_to\_save = tf.train.Saver()**
938. **saver\_best = tf.train.Saver()**
939. **with tf.Session() as sess:**
940. **sess.run([tf.global\_variables\_initializer(), tf.local\_variables\_initializer()])**
941. **saver\_to\_restore.restore(sess, args.restore\_path)**
942. **merged = tf.summary.merge\_all()**
943. **writer = tf.summary.FileWriter(args.log\_dir, sess.graph)**
944. **print('\n----------- start to train -----------\n')**
945. **best\_mAP = -np.Inf**
946. **for epoch in range(args.total\_epoches):**
947. **sess.run(train\_init\_op)**
948. **loss\_total, loss\_xy, loss\_wh, loss\_conf, loss\_class = AverageMeter(), AverageMeter(), AverageMeter(), AverageMeter(), AverageMeter()**
949. **for i in trange(args.train\_batch\_num):**
950. **\_, summary, \_\_y\_pred, \_\_y\_true, \_\_loss, \_\_global\_step, \_\_lr = sess.run(**
951. **[train\_op, merged, y\_pred, y\_true, loss, global\_step, learning\_rate],**
952. **feed\_dict={is\_training: True})**
953. **writer.add\_summary(summary, global\_step=\_\_global\_step)**
954. **loss\_total.update(\_\_loss[0], len(\_\_y\_pred[0]))**
955. **loss\_xy.update(\_\_loss[1], len(\_\_y\_pred[0]))**
956. **loss\_wh.update(\_\_loss[2], len(\_\_y\_pred[0]))**
957. **loss\_conf.update(\_\_loss[3], len(\_\_y\_pred[0]))**
958. **loss\_class.update(\_\_loss[4], len(\_\_y\_pred[0]))**
959. **if \_\_global\_step % args.train\_evaluation\_step == 0 and \_\_global\_step > 0:**
960. **recall, precision = evaluate\_on\_gpu(sess, gpu\_nms\_op, pred\_boxes\_flag, pred\_scores\_flag, \_\_y\_pred, \_\_y\_true, args.class\_num, args.nms\_threshold)**
961. **info = "Epoch: {}, global\_step: {} | loss: total: {:.2f}, xy: {:.2f}, wh: {:.2f}, conf: {:.2f}, class: {:.2f} | ".format(**
962. **epoch, int(\_\_global\_step), loss\_total.average, loss\_xy.average, loss\_wh.average, loss\_conf.average, loss\_class.average)**
963. **info += 'Last batch: rec: {:.3f}, prec: {:.3f} | lr: {:.5g}'.format(recall, precision, \_\_lr)**
964. **print(info)**
965. **logging.info(info)**
966. **writer.add\_summary(make\_summary('evaluation/train\_batch\_recall', recall), global\_step=\_\_global\_step)**
967. **writer.add\_summary(make\_summary('evaluation/train\_batch\_precision', precision), global\_step=\_\_global\_step)**
968. **if np.isnan(loss\_total.average):**
969. **print('\*\*\*\*' \* 10)**
970. **raise ArithmeticError(**
971. **'Gradient exploded! Please train again and you may need modify some parameters.')**
972. **if epoch % args.save\_epoch == 0 and epoch > 0:**
973. **if loss\_total.average <= 2.:**
974. **saver\_to\_save.save(sess, args.save\_dir + 'model-epoch\_{}\_step\_{}\_loss\_{:.4f}\_lr\_{:.5g}'.format(epoch, int(\_\_global\_step), loss\_total.average, \_\_lr))**
975. **if epoch % args.val\_evaluation\_epoch == 0 and epoch >= args.warm\_up\_epoch:**
976. **sess.run(val\_init\_op)**
977. **val\_loss\_total, val\_loss\_xy, val\_loss\_wh, val\_loss\_conf, val\_loss\_class = \**
978. **AverageMeter(), AverageMeter(), AverageMeter(), AverageMeter(), AverageMeter()**
979. **val\_preds = []**
980. **for j in trange(args.val\_img\_cnt):**
981. **\_\_image\_ids, \_\_y\_pred, \_\_loss = sess.run([image\_ids, y\_pred, loss],**
982. **feed\_dict={is\_training: False})**
983. **pred\_content = get\_preds\_gpu(sess, gpu\_nms\_op, pred\_boxes\_flag, pred\_scores\_flag, \_\_image\_ids, \_\_y\_pred)**
984. **val\_preds.extend(pred\_content)**
985. **val\_loss\_total.update(\_\_loss[0])**
986. **val\_loss\_xy.update(\_\_loss[1])**
987. **val\_loss\_wh.update(\_\_loss[2])**
988. **val\_loss\_conf.update(\_\_loss[3])**
989. **val\_loss\_class.update(\_\_loss[4])**
990. **rec\_total, prec\_total, ap\_total = AverageMeter(), AverageMeter(), AverageMeter()**
991. **gt\_dict = parse\_gt\_rec(args.val\_file, args.img\_size, args.letterbox\_resize)**
992. **info = '======> Epoch: {}, global\_step: {}, lr: {:.6g} <======\n'.format(epoch, \_\_global\_step, \_\_lr)**
993. **for ii in range(args.class\_num):**
994. **npos, nd, rec, prec, ap = voc\_eval(gt\_dict, val\_preds, ii, iou\_thres=args.eval\_threshold, use\_07\_metric=args.use\_voc\_07\_metric)**
995. **info += 'EVAL: Class {}: Recall: {:.4f}, Precision: {:.4f}, AP: {:.4f}\n'.format(ii, rec, prec, ap)**
996. **rec\_total.update(rec, npos)**
997. **prec\_total.update(prec, nd)**
998. **ap\_total.update(ap, 1)**
999. **mAP = ap\_total.average**
1000. **info += 'EVAL: Recall: {:.4f}, Precison: {:.4f}, mAP: {:.4f}\n'.format(rec\_total.average, prec\_total.average, mAP)**
1001. **info += 'EVAL: loss: total: {:.2f}, xy: {:.2f}, wh: {:.2f}, conf: {:.2f}, class: {:.2f}\n'.format(**
1002. **val\_loss\_total.average, val\_loss\_xy.average, val\_loss\_wh.average, val\_loss\_conf.average, val\_loss\_class.average)**
1003. **print(info)**
1004. **logging.info(info)**
1005. **if mAP > best\_mAP:**
1006. **best\_mAP = mAP**
1007. **saver\_best.save(sess, args.save\_dir + 'best\_model\_Epoch\_{}\_step\_{}\_mAP\_{:.4f}\_loss\_{:.4f}\_lr\_{:.7g}'.format(**
1008. **epoch, int(\_\_global\_step), best\_mAP, val\_loss\_total.average, \_\_lr))**
1009. **writer.add\_summary(make\_summary('evaluation/val\_mAP', mAP), global\_step=epoch)**
1010. **writer.add\_summary(make\_summary('evaluation/val\_recall', rec\_total.average), global\_step=epoch)**
1011. **writer.add\_summary(make\_summary('evaluation/val\_precision', prec\_total.average), global\_step=epoch)**
1012. **writer.add\_summary(make\_summary('validation\_statistics/total\_loss', val\_loss\_total.average), global\_step=epoch)**
1013. **writer.add\_summary(make\_summary('validation\_statistics/loss\_xy', val\_loss\_xy.average), global\_step=epoch)**
1014. **writer.add\_summary(make\_summary('validation\_statistics/loss\_wh', val\_loss\_wh.average), global\_step=epoch)**
1015. **writer.add\_summary(make\_summary('validation\_statistics/loss\_conf', val\_loss\_conf.average), global\_step=epoch)**
1016. **writer.add\_summary(make\_summary('validation\_statistics/loss\_class', val\_loss\_class.average), global\_step=epoch)**
1017. **from \_\_future\_\_ import division, print\_function**
1018. **import tensorflow as tf**
1019. **import numpy as np**
1020. **import argparse**
1021. **import cv2**
1022. **import time**
1023. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
1024. **from utils.nms\_utils import gpu\_nms**
1025. **from utils.plot\_utils import get\_color\_table, plot\_one\_box**
1026. **from utils.data\_aug import letterbox\_resize**
1027. **from model import yolov3**
1028. **from tensorflow.compat.v1 import ConfigProto**
1029. **from tensorflow.compat.v1 import InteractiveSession**
1030. **config = ConfigProto()**
1031. **config.gpu\_options.allow\_growth = True**
1032. **session = InteractiveSession(config=config)**
1033. **parser = argparse.ArgumentParser(description="YOLO-V3 video test procedure.")**
1034. **parser.add\_argument("input\_video", type=str,**
1035. **help="The path of the input video.")**
1036. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
1037. **help="The path of the anchor txt file.")**
1038. **parser.add\_argument("--new\_size", nargs='\*', type=int, default=[416, 416],**
1039. **help="Resize the input image with `new\_size`, size format: [width, height]")**
1040. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=True,**
1041. **help="Whether to use the letterbox resize.")**
1042. **parser.add\_argument("--class\_name\_path", type=str, default="./data/voc\_names.txt",**
1043. **help="The path of the class names.")**
1044. **parser.add\_argument("--restore\_path", type=str, default="./checkpoint/model-epoch\_90\_step\_86449\_loss\_1.0549\_lr\_1e-05",**
1045. **help="The path of the weights to restore.")**
1046. **parser.add\_argument("--save\_video", type=lambda x: (str(x).lower() == 'true'), default=False,**
1047. **help="Whether to save the video detection results.")**
1048. **args = parser.parse\_args()**
1049. **args.anchors = parse\_anchors(args.anchor\_path)**
1050. **args.classes = read\_class\_names(args.class\_name\_path)**
1051. **args.num\_class = len(args.classes)**
1052. **color\_table = get\_color\_table(args.num\_class)**
1053. **vid = cv2.VideoCapture(args.input\_video)**
1054. **video\_frame\_cnt = int(vid.get(7))**
1055. **video\_width = int(vid.get(3))**
1056. **video\_height = int(vid.get(4))**
1057. **video\_fps = int(vid.get(5))**
1058. **if args.save\_video:**
1059. **fourcc = cv2.VideoWriter\_fourcc('m', 'p', '4', 'v')**
1060. **videoWriter = cv2.VideoWriter('video\_result.mp4', fourcc, video\_fps, (video\_width, video\_height))**
1061. **with tf.Session() as sess:**
1062. **input\_data = tf.placeholder(tf.float32, [1, args.new\_size[1], args.new\_size[0], 3], name='input\_data')**
1063. **yolo\_model = yolov3(args.num\_class, args.anchors)**
1064. **with tf.variable\_scope('yolov3'):**
1065. **pred\_feature\_maps = yolo\_model.forward(input\_data, False)**
1066. **pred\_boxes, pred\_confs, pred\_probs = yolo\_model.predict(pred\_feature\_maps)**
1067. **pred\_scores = pred\_confs \* pred\_probs**
1068. **boxes, scores, labels = gpu\_nms(pred\_boxes, pred\_scores, args.num\_class, max\_boxes=200, score\_thresh=0.3, nms\_thresh=0.45)**
1069. **saver = tf.train.Saver()**
1070. **saver.restore(sess, args.restore\_path)**
1071. **for i in range(video\_frame\_cnt):**
1072. **ret, img\_ori = vid.read()**
1073. **if args.letterbox\_resize:**
1074. **img, resize\_ratio, dw, dh = letterbox\_resize(img\_ori, args.new\_size[0], args.new\_size[1])**
1075. **else:**
1076. **height\_ori, width\_ori = img\_ori.shape[:2]**
1077. **img = cv2.resize(img\_ori, tuple(args.new\_size))**
1078. **img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**
1079. **img = np.asarray(img, np.float32)**
1080. **img = img[np.newaxis, :] / 255.**
1081. **start\_time = time.time()**
1082. **boxes\_, scores\_, labels\_ = sess.run([boxes, scores, labels], feed\_dict={input\_data: img})**
1083. **end\_time = time.time()**
1084. **if args.letterbox\_resize:**
1085. **boxes\_[:, [0, 2]] = (boxes\_[:, [0, 2]] - dw) / resize\_ratio**
1086. **boxes\_[:, [1, 3]] = (boxes\_[:, [1, 3]] - dh) / resize\_ratio**
1087. **else:**
1088. **boxes\_[:, [0, 2]] \*= (width\_ori/float(args.new\_size[0]))**
1089. **boxes\_[:, [1, 3]] \*= (height\_ori/float(args.new\_size[1]))**
1090. **for i in range(len(boxes\_)):**
1091. **x0, y0, x1, y1 = boxes\_[i]**
1092. **if scores\_[i] < 0.8:**
1093. **continue**
1094. **plot\_one\_box(img\_ori, [x0, y0, x1, y1], label=args.classes[labels\_[i]] + ', {:.2f}%'.format(scores\_[i] \* 100), color=color\_table[labels\_[i]])**
1095. **cv2.putText(img\_ori, '{:.2f}ms'.format((end\_time - start\_time) \* 1000), (40, 40), 0,**
1096. **fontScale=1, color=(0, 255, 0), thickness=2)**
1097. **cv2.imshow('image', img\_ori)**
1098. **if args.save\_video:**
1099. **videoWriter.write(img\_ori)**
1100. **if cv2.waitKey(1) & 0xFF == ord('q'):**
1101. **break**
1102. **vid.release()**
1103. **if args.save\_video:**
1104. **videoWriter.release()**
1105. **from \_\_future\_\_ import division, print\_function**
1106. **import tensorflow as tf**
1107. **import numpy as np**
1108. **import argparse**
1109. **import cv2**
1110. **import time**
1111. **from utils.misc\_utils import parse\_anchors, read\_class\_names**
1112. **from utils.nms\_utils import gpu\_nms**
1113. **from utils.plot\_utils import get\_color\_table, plot\_one\_box**
1114. **from utils.data\_aug import letterbox\_resize**
1115. **from model import yolov3**
1116. **from tensorflow.compat.v1 import ConfigProto**
1117. **from tensorflow.compat.v1 import InteractiveSession**
1118. **config = ConfigProto()**
1119. **config.gpu\_options.allow\_growth = True**
1120. **session = InteractiveSession(config=config)**
1121. **parser = argparse.ArgumentParser(description="YOLO-V3 video test procedure.")**
1122. **parser.add\_argument("input\_video", type=str,**
1123. **help="The path of the input video.")**
1124. **parser.add\_argument("--anchor\_path", type=str, default="./data/yolo\_anchors.txt",**
1125. **help="The path of the anchor txt file.")**
1126. **parser.add\_argument("--new\_size", nargs='\*', type=int, default=[416, 416],**
1127. **help="Resize the input image with `new\_size`, size format: [width, height]")**
1128. **parser.add\_argument("--letterbox\_resize", type=lambda x: (str(x).lower() == 'true'), default=True,**
1129. **help="Whether to use the letterbox resize.")**
1130. **parser.add\_argument("--class\_name\_path", type=str, default="./data/coco.names",**
1131. **help="The path of the class names.")**
1132. **parser.add\_argument("--restore\_path", type=str, default="./data/darknet\_weights/yolov3.ckpt",**
1133. **help="The path of the weights to restore.")**
1134. **parser.add\_argument("--save\_video", type=lambda x: (str(x).lower() == 'true'), default=False,**
1135. **help="Whether to save the video detection results.")**
1136. **args = parser.parse\_args()**
1137. **args.anchors = parse\_anchors(args.anchor\_path)**
1138. **args.classes = read\_class\_names(args.class\_name\_path)**
1139. **args.num\_class = len(args.classes)**
1140. **color\_table = get\_color\_table(args.num\_class)**
1141. **vid = cv2.VideoCapture(args.input\_video)**
1142. **video\_frame\_cnt = int(vid.get(7))**
1143. **video\_width = int(vid.get(3))**
1144. **video\_height = int(vid.get(4))**
1145. **video\_fps = int(vid.get(5))**
1146. **if args.save\_video:**
1147. **fourcc = cv2.VideoWriter\_fourcc('m', 'p', '4', 'v')**
1148. **videoWriter = cv2.VideoWriter('video\_result.mp4', fourcc, video\_fps, (video\_width, video\_height))**
1149. **with tf.Session() as sess:**
1150. **input\_data = tf.placeholder(tf.float32, [1, args.new\_size[1], args.new\_size[0], 3], name='input\_data')**
1151. **yolo\_model = yolov3(args.num\_class, args.anchors)**
1152. **with tf.variable\_scope('yolov3'):**
1153. **pred\_feature\_maps = yolo\_model.forward(input\_data, False)**
1154. **pred\_boxes, pred\_confs, pred\_probs = yolo\_model.predict(pred\_feature\_maps)**
1155. **pred\_scores = pred\_confs \* pred\_probs**
1156. **boxes, scores, labels = gpu\_nms(pred\_boxes, pred\_scores, args.num\_class, max\_boxes=200, score\_thresh=0.3, nms\_thresh=0.45)**
1157. **saver = tf.train.Saver()**
1158. **saver.restore(sess, args.restore\_path)**
1159. **for i in range(video\_frame\_cnt):**
1160. **ret, img\_ori = vid.read()**
1161. **if args.letterbox\_resize:**
1162. **img, resize\_ratio, dw, dh = letterbox\_resize(img\_ori, args.new\_size[0], args.new\_size[1])**
1163. **else:**
1164. **height\_ori, width\_ori = img\_ori.shape[:2]**
1165. **img = cv2.resize(img\_ori, tuple(args.new\_size))**
1166. **img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**
1167. **img = np.asarray(img, np.float32)**
1168. **img = img[np.newaxis, :] / 255.**
1169. **start\_time = time.time()**
1170. **boxes\_, scores\_, labels\_ = sess.run([boxes, scores, labels], feed\_dict={input\_data: img})**
1171. **end\_time = time.time()**
1172. **if args.letterbox\_resize:**
1173. **boxes\_[:, [0, 2]] = (boxes\_[:, [0, 2]] - dw) / resize\_ratio**
1174. **boxes\_[:, [1, 3]] = (boxes\_[:, [1, 3]] - dh) / resize\_ratio**
1175. **else:**
1176. **boxes\_[:, [0, 2]] \*= (width\_ori/float(args.new\_size[0]))**
1177. **boxes\_[:, [1, 3]] \*= (height\_ori/float(args.new\_size[1]))**
1178. **for i in range(len(boxes\_)):**
1179. **x0, y0, x1, y1 = boxes\_[i]**
1180. **plot\_one\_box(img\_ori, [x0, y0, x1, y1], label=args.classes[labels\_[i]] + ', {:.2f}%'.format(scores\_[i] \* 100), color=color\_table[labels\_[i]])**
1181. **cv2.putText(img\_ori, '{:.2f}ms'.format((end\_time - start\_time) \* 1000), (40, 40), 0,**
1182. **fontScale=1, color=(0, 255, 0), thickness=2)**
1183. **cv2.imshow('image', img\_ori)**
1184. **if args.save\_video:**
1185. **videoWriter.write(img\_ori)**
1186. **if cv2.waitKey(1) & 0xFF == ord('q'):**
1187. **break**
1188. **vid.release()**
1189. **if args.save\_video:**
1190. **videoWriter.release()**
1191. **import numpy as np**
1192. **import tensorflow as tf**
1193. **import random**
1194. **from tensorflow.core.framework import summary\_pb2**
1195. **def make\_summary(name, val):**
1196. **return summary\_pb2.Summary(value=[summary\_pb2.Summary.Value(tag=name, simple\_value=val)])**
1197. **class AverageMeter(object):**
1198. **def \_\_init\_\_(self):**
1199. **self.reset()**
1200. **def reset(self):**
1201. **self.val = 0**
1202. **self.average = 0**
1203. **self.sum = 0**
1204. **self.count = 0**
1205. **def update(self, val, n=1):**
1206. **self.val = val**
1207. **self.sum += val \* n**
1208. **self.count += n**
1209. **self.average = self.sum / float(self.count)**
1210. **def parse\_anchors(anchor\_path):**
1211. **anchors = np.reshape(np.asarray(open(anchor\_path, 'r').read().split(','), np.float32), [-1, 2])**
1212. **return anchors**
1213. **def read\_class\_names(class\_name\_path):**
1214. **names = {}**
1215. **with open(class\_name\_path, 'r') as data:**
1216. **for ID, name in enumerate(data):**
1217. **names[ID] = name.strip('\n')**
1218. **return names**
1219. **def shuffle\_and\_overwrite(file\_name):**
1220. **content = open(file\_name, 'r').readlines()**
1221. **random.shuffle(content)**
1222. **with open(file\_name, 'w') as f:**
1223. **for line in content:**
1224. **f.write(line)**
1225. **def update\_dict(ori\_dict, new\_dict):**
1226. **if not ori\_dict:**
1227. **return new\_dict**
1228. **for key in ori\_dict:**
1229. **ori\_dict[key] += new\_dict[key]**
1230. **return ori\_dict**
1231. **def list\_add(ori\_list, new\_list):**
1232. **for i in range(len(ori\_list)):**
1233. **ori\_list[i] += new\_list[i]**
1234. **return ori\_list**
1235. **def load\_weights(var\_list, weights\_file):**
1236. **with open(weights\_file, "rb") as fp:**
1237. **np.fromfile(fp, dtype=np.int32, count=5)**
1238. **weights = np.fromfile(fp, dtype=np.float32)**
1239. **ptr = 0**
1240. **i = 0**
1241. **assign\_ops = []**
1242. **while i < len(var\_list) - 1:**
1243. **var1 = var\_list[i]**
1244. **var2 = var\_list[i + 1]**
1245. **if 'Conv' in var1.name.split('/')[-2]:**
1246. **if 'BatchNorm' in var2.name.split('/')[-2]:**
1247. **gamma, beta, mean, var = var\_list[i + 1:i + 5]**
1248. **batch\_norm\_vars = [beta, gamma, mean, var]**
1249. **for var in batch\_norm\_vars:**
1250. **shape = var.shape.as\_list()**
1251. **num\_params = np.prod(shape)**
1252. **var\_weights = weights[ptr:ptr + num\_params].reshape(shape)**
1253. **ptr += num\_params**
1254. **assign\_ops.append(tf.assign(var, var\_weights, validate\_shape=True))**
1255. **i += 4**
1256. **elif 'Conv' in var2.name.split('/')[-2]:**
1257. **bias = var2**
1258. **bias\_shape = bias.shape.as\_list()**
1259. **bias\_params = np.prod(bias\_shape)**
1260. **bias\_weights = weights[ptr:ptr +**
1261. **bias\_params].reshape(bias\_shape)**
1262. **ptr += bias\_params**
1263. **assign\_ops.append(tf.assign(bias, bias\_weights, validate\_shape=True))**
1264. **i += 1**
1265. **shape = var1.shape.as\_list()**
1266. **num\_params = np.prod(shape)**
1267. **var\_weights = weights[ptr:ptr + num\_params].reshape(**
1268. **(shape[3], shape[2], shape[0], shape[1]))**
1269. **var\_weights = np.transpose(var\_weights, (2, 3, 1, 0))**
1270. **ptr += num\_params**
1271. **assign\_ops.append(**
1272. **tf.assign(var1, var\_weights, validate\_shape=True))**
1273. **i += 1**
1274. **return assign\_ops**
1275. **def config\_learning\_rate(args, global\_step):**
1276. **if args.lr\_type == 'exponential':**
1277. **lr\_tmp = tf.train.exponential\_decay(args.learning\_rate\_init, global\_step, args.lr\_decay\_freq,**
1278. **args.lr\_decay\_factor, staircase=True, name='exponential\_learning\_rate')**
1279. **return tf.maximum(lr\_tmp, args.lr\_lower\_bound)**
1280. **elif args.lr\_type == 'cosine\_decay':**
1281. **train\_steps = (args.total\_epoches - float(args.use\_warm\_up) \* args.warm\_up\_epoch) \* args.train\_batch\_num**
1282. **return args.lr\_lower\_bound + 0.5 \* (args.learning\_rate\_init - args.lr\_lower\_bound) \* \**
1283. **(1 + tf.cos(global\_step / train\_steps \* np.pi))**
1284. **elif args.lr\_type == 'cosine\_decay\_restart':**
1285. **return tf.train.cosine\_decay\_restarts(args.learning\_rate\_init, global\_step,**
1286. **args.lr\_decay\_freq, t\_mul=2.0, m\_mul=1.0,**
1287. **name='cosine\_decay\_learning\_rate\_restart')**
1288. **elif args.lr\_type == 'fixed':**
1289. **return tf.convert\_to\_tensor(args.learning\_rate\_init, name='fixed\_learning\_rate')**
1290. **elif args.lr\_type == 'piecewise':**
1291. **return tf.train.piecewise\_constant(global\_step, boundaries=args.pw\_boundaries, values=args.pw\_values,**
1292. **name='piecewise\_learning\_rate')**
1293. **else:**
1294. **raise ValueError('Unsupported learning rate type!')**
1295. **def config\_optimizer(optimizer\_name, learning\_rate, decay=0.9, momentum=0.9):**
1296. **if optimizer\_name == 'momentum':**
1297. **return tf.train.MomentumOptimizer(learning\_rate, momentum=momentum)**
1298. **elif optimizer\_name == 'rmsprop':**
1299. **return tf.train.RMSPropOptimizer(learning\_rate, decay=decay, momentum=momentum)**
1300. **elif optimizer\_name == 'adam':**
1301. **return tf.train.AdamOptimizer(learning\_rate)**
1302. **elif optimizer\_name == 'sgd':**
1303. **return tf.train.GradientDescentOptimizer(learning\_rate)**
1304. **else:**
1305. **raise ValueError('Unsupported optimizer type!')**
1306. **from typing import (**
1307. **Any, Callable, Dict, Generator, Iterable, List, IO, NoReturn, Optional,**
1308. **Pattern, Sequence, Tuple, Type, Union, TypeVar, overload**
1309. **)**
1310. **import sys**
1311. **\_T = TypeVar('\_T')**
1312. **\_ActionT = TypeVar('\_ActionT', bound='Action')**
1313. **if sys.version\_info >= (3,):**
1314. **\_Text = str**
1315. **else:**
1316. **\_Text = Union[str, unicode]**
1317. **ONE\_OR\_MORE: str**
1318. **OPTIONAL: str**
1319. **PARSER: str**
1320. **REMAINDER: str**
1321. **SUPPRESS: str**
1322. **ZERO\_OR\_MORE: str**
1323. **\_UNRECOGNIZED\_ARGS\_ATTR: str**
1324. **class ArgumentError(Exception): ...**
1325. **class \_AttributeHolder:**
1326. **def \_get\_kwargs(self) -> List[Tuple[str, Any]]: ...**
1327. **def \_get\_args(self) -> List[Any]: ...**
1328. **class \_ActionsContainer:**
1329. **description: Optional[\_Text]**
1330. **prefix\_chars: \_Text**
1331. **argument\_default: Optional[\_Text]**
1332. **conflict\_handler: \_Text**
1333. **\_registries: Dict[\_Text, Dict[Any, Any]]**
1334. **\_actions: List[Action]**
1335. **\_option\_string\_actions: Dict[\_Text, Action]**
1336. **\_action\_groups: List[\_ArgumentGroup]**
1337. **\_mutually\_exclusive\_groups: List[\_MutuallyExclusiveGroup]**
1338. **\_defaults: Dict[str, Any]**
1339. **\_negative\_number\_matcher: Pattern[str]**
1340. **\_has\_negative\_number\_optionals: List[bool]**
1341. **def \_\_init\_\_(self, description: Optional[\_Text], prefix\_chars: \_Text,**
1342. **argument\_default: Optional[\_Text], conflict\_handler: \_Text) -> None: ...**
1343. **def register(self, registry\_name: \_Text, value: Any, object: Any) -> None: ...**
1344. **def \_registry\_get(self, registry\_name: \_Text, value: Any, default: Any = ...) -> Any: ...**
1345. **def set\_defaults(self, \*\*kwargs: Any) -> None: ...**
1346. **def get\_default(self, dest: \_Text) -> Any: ...**
1347. **def add\_argument(self,**
1348. **\*name\_or\_flags: \_Text,**
1349. **action: Union[\_Text, Type[Action]] = ...,**
1350. **nargs: Union[int, \_Text] = ...,**
1351. **const: Any = ...,**
1352. **default: Any = ...,**
1353. **type: Union[Callable[[str], \_T], FileType] = ...,**
1354. **choices: Iterable[\_T] = ...,**
1355. **required: bool = ...,**
1356. **help: Optional[\_Text] = ...,**
1357. **metavar: Union[\_Text, Tuple[\_Text, ...]] = ...,**
1358. **dest: Optional[\_Text] = ...,**
1359. **version: \_Text = ...,**
1360. **\*\*kwargs: Any) -> Action: ...**
1361. **def add\_argument\_group(self, \*args: Any, \*\*kwargs: Any) -> \_ArgumentGroup: ...**
1362. **def add\_mutually\_exclusive\_group(self, \*\*kwargs: Any) -> \_MutuallyExclusiveGroup: ...**
1363. **def \_add\_action(self, action: \_ActionT) -> \_ActionT: ...**
1364. **def \_remove\_action(self, action: Action) -> None: ...**
1365. **def \_add\_container\_actions(self, container: \_ActionsContainer) -> None: ...**
1366. **def \_get\_positional\_kwargs(self, dest: \_Text, \*\*kwargs: Any) -> Dict[str, Any]: ...**
1367. **def \_get\_optional\_kwargs(self, \*args: Any, \*\*kwargs: Any) -> Dict[str, Any]: ...**
1368. **def \_pop\_action\_class(self, kwargs: Any, default: Optional[Type[Action]] = ...) -> Type[Action]: ...**
1369. **def \_get\_handler(self) -> Callable[[Action, Iterable[Tuple[\_Text, Action]]], Any]: ...**
1370. **def \_check\_conflict(self, action: Action) -> None: ...**
1371. **def \_handle\_conflict\_error(self, action: Action, conflicting\_actions: Iterable[Tuple[\_Text, Action]]) -> NoReturn: ...**
1372. **def \_handle\_conflict\_resolve(self, action: Action, conflicting\_actions: Iterable[Tuple[\_Text, Action]]) -> None: ...**
1373. **class ArgumentParser(\_AttributeHolder, \_ActionsContainer):**
1374. **prog: \_Text**
1375. **usage: Optional[\_Text]**
1376. **epilog: Optional[\_Text]**
1377. **formatter\_class: Type[HelpFormatter]**
1378. **fromfile\_prefix\_chars: Optional[\_Text]**
1379. **add\_help: bool**
1380. **if sys.version\_info >= (3, 5):**
1381. **allow\_abbrev: bool**
1382. **\_positionals: \_ArgumentGroup**
1383. **\_optionals: \_ArgumentGroup**
1384. **\_subparsers: Optional[\_ArgumentGroup]**
1385. **if sys.version\_info >= (3, 5):**
1386. **def \_\_init\_\_(self,**
1387. **prog: Optional[str] = ...,**
1388. **usage: Optional[str] = ...,**
1389. **description: Optional[str] = ...,**
1390. **epilog: Optional[str] = ...,**
1391. **parents: Sequence[ArgumentParser] = ...,**
1392. **formatter\_class: Type[HelpFormatter] = ...,**
1393. **prefix\_chars: \_Text = ...,**
1394. **fromfile\_prefix\_chars: Optional[str] = ...,**
1395. **argument\_default: Optional[str] = ...,**
1396. **conflict\_handler: \_Text = ...,**
1397. **add\_help: bool = ...,**
1398. **allow\_abbrev: bool = ...) -> None: ...**
1399. **else:**
1400. **def \_\_init\_\_(self,**
1401. **prog: Optional[\_Text] = ...,**
1402. **usage: Optional[\_Text] = ...,**
1403. **description: Optional[\_Text] = ...,**
1404. **epilog: Optional[\_Text] = ...,**
1405. **parents: Sequence[ArgumentParser] = ...,**
1406. **formatter\_class: Type[HelpFormatter] = ...,**
1407. **prefix\_chars: \_Text = ...,**
1408. **fromfile\_prefix\_chars: Optional[\_Text] = ...,**
1409. **argument\_default: Optional[\_Text] = ...,**
1410. **conflict\_handler: \_Text = ...,**
1411. **add\_help: bool = ...) -> None: ...**
1412. **def parse\_args(self, args: Optional[Sequence[\_Text]] = ...,**
1413. **namespace: Optional[Namespace] = ...) -> Namespace: ...**
1414. **if sys.version\_info >= (3, 7):**
1415. **def add\_subparsers(self, title: \_Text = ...,**
1416. **description: Optional[\_Text] = ...,**
1417. **prog: \_Text = ...,**
1418. **parser\_class: Type[ArgumentParser] = ...,**
1419. **action: Type[Action] = ...,**
1420. **option\_string: \_Text = ...,**
1421. **dest: Optional[\_Text] = ...,**
1422. **required: bool = ...,**
1423. **help: Optional[\_Text] = ...,**
1424. **metavar: Optional[\_Text] = ...) -> \_SubParsersAction: ...**
1425. **else:**
1426. **def add\_subparsers(self, title: \_Text = ...,**
1427. **description: Optional[\_Text] = ...,**
1428. **prog: \_Text = ...,**
1429. **parser\_class: Type[ArgumentParser] = ...,**
1430. **action: Type[Action] = ...,**
1431. **option\_string: \_Text = ...,**
1432. **dest: Optional[\_Text] = ...,**
1433. **help: Optional[\_Text] = ...,**
1434. **metavar: Optional[\_Text] = ...) -> \_SubParsersAction: ...**
1435. **def print\_usage(self, file: Optional[IO[str]] = ...) -> None: ...**
1436. **def print\_help(self, file: Optional[IO[str]] = ...) -> None: ...**
1437. **def format\_usage(self) -> str: ...**
1438. **def format\_help(self) -> str: ...**
1439. **def parse\_known\_args(self, args: Optional[Sequence[\_Text]] = ...,**
1440. **namespace: Optional[Namespace] = ...) -> Tuple[Namespace, List[str]]: ...**
1441. **def convert\_arg\_line\_to\_args(self, arg\_line: \_Text) -> List[str]: ...**
1442. **def exit(self, status: int = ..., message: Optional[\_Text] = ...) -> NoReturn: ...**
1443. **def error(self, message: \_Text) -> NoReturn: ...**
1444. **if sys.version\_info >= (3, 7):**
1445. **def parse\_intermixed\_args(self, args: Optional[Sequence[\_Text]] = ...,**
1446. **namespace: Optional[Namespace] = ...) -> Namespace: ...**
1447. **def parse\_known\_intermixed\_args(self,**
1448. **args: Optional[Sequence[\_Text]] = ...,**
1449. **namespace: Optional[Namespace] = ...) -> Tuple[Namespace, List[str]]: ...**
1450. **def \_get\_optional\_actions(self) -> List[Action]: ...**
1451. **def \_get\_positional\_actions(self) -> List[Action]: ...**
1452. **def \_parse\_known\_args(self, arg\_strings: List[\_Text], namespace: Namespace) -> Tuple[Namespace, List[str]]: ...**
1453. **def \_read\_args\_from\_files(self, arg\_strings: List[\_Text]) -> List[\_Text]: ...**
1454. **def \_match\_argument(self, action: Action, arg\_strings\_pattern: \_Text) -> int: ...**
1455. **def \_match\_arguments\_partial(self, actions: Sequence[Action], arg\_strings\_pattern: \_Text) -> List[int]: ...**
1456. **def \_parse\_optional(self, arg\_string: \_Text) -> Optional[Tuple[Optional[Action], \_Text, Optional[\_Text]]]: ...**
1457. **def \_get\_option\_tuples(self, option\_string: \_Text) -> List[Tuple[Action, \_Text, Optional[\_Text]]]: ...**
1458. **def \_get\_nargs\_pattern(self, action: Action) -> \_Text: ...**
1459. **def \_get\_values(self, action: Action, arg\_strings: List[\_Text]) -> Any: ...**
1460. **def \_get\_value(self, action: Action, arg\_string: \_Text) -> Any: ...**
1461. **def \_check\_value(self, action: Action, value: Any) -> None: ...**
1462. **def \_get\_formatter(self) -> HelpFormatter: ...**
1463. **def \_print\_message(self, message: str, file: Optional[IO[str]] = ...) -> None: ...**
1464. **class HelpFormatter:**
1465. **\_prog: \_Text**
1466. **\_indent\_increment: int**
1467. **\_max\_help\_position: int**
1468. **\_width: int**
1469. **\_current\_indent: int**
1470. **\_level: int**
1471. **\_action\_max\_length: int**
1472. **\_root\_section: Any**
1473. **\_current\_section: Any**
1474. **\_whitespace\_matcher: Pattern[str]**
1475. **\_long\_break\_matcher: Pattern[str]**
1476. **\_Section: Type[Any]**
1477. **def \_\_init\_\_(self, prog: \_Text, indent\_increment: int = ...,**
1478. **max\_help\_position: int = ...,**
1479. **width: Optional[int] = ...) -> None: ...**
1480. **def \_indent(self) -> None: ...**
1481. **def \_dedent(self) -> None: ...**
1482. **def \_add\_item(self, func: Callable[..., \_Text], args: Iterable[Any]) -> None: ...**
1483. **def start\_section(self, heading: Optional[\_Text]) -> None: ...**
1484. **def end\_section(self) -> None: ...**
1485. **def add\_text(self, text: Optional[\_Text]) -> None: ...**
1486. **def add\_usage(self, usage: \_Text, actions: Iterable[Action], groups: Iterable[\_ArgumentGroup], prefix: Optional[\_Text] = ...) -> None: ...**
1487. **def add\_argument(self, action: Action) -> None: ...**
1488. **def add\_arguments(self, actions: Iterable[Action]) -> None: ...**
1489. **def format\_help(self) -> \_Text: ...**
1490. **def \_join\_parts(self, part\_strings: Iterable[\_Text]) -> \_Text: ...**
1491. **def \_format\_usage(self, usage: \_Text, actions: Iterable[Action], groups: Iterable[\_ArgumentGroup], prefix: Optional[\_Text]) -> \_Text: ...**
1492. **def \_format\_actions\_usage(self, actions: Iterable[Action], groups: Iterable[\_ArgumentGroup]) -> \_Text: ...**
1493. **def \_format\_text(self, text: \_Text) -> \_Text: ...**
1494. **def \_format\_action(self, action: Action) -> \_Text: ...**
1495. **def \_format\_action\_invocation(self, action: Action) -> \_Text: ...**
1496. **def \_metavar\_formatter(self, action: Action, default\_metavar: \_Text) -> Callable[[int], Tuple[\_Text, ...]]: ...**
1497. **def \_format\_args(self, action: Action, default\_metavar: \_Text) -> \_Text: ...**
1498. **def \_expand\_help(self, action: Action) -> \_Text: ...**
1499. **def \_iter\_indented\_subactions(self, action: Action) -> Generator[Action, None, None]: ...**
1500. **def \_split\_lines(self, text: \_Text, width: int) -> List[\_Text]: ...**
1501. **def \_fill\_text(self, text: \_Text, width: int, indent: int) -> \_Text: ...**
1502. **def \_get\_help\_string(self, action: Action) -> Optional[\_Text]: ...**
1503. **def \_get\_default\_metavar\_for\_optional(self, action: Action) -> \_Text: ...**
1504. **def \_get\_default\_metavar\_for\_positional(self, action: Action) -> \_Text: ...**
1505. **class RawDescriptionHelpFormatter(HelpFormatter): ...**
1506. **class RawTextHelpFormatter(HelpFormatter): ...**
1507. **class ArgumentDefaultsHelpFormatter(HelpFormatter): ...**
1508. **if sys.version\_info >= (3,):**
1509. **class MetavarTypeHelpFormatter(HelpFormatter): ...**
1510. **class Action(\_AttributeHolder):**
1511. **option\_strings: Sequence[\_Text]**
1512. **dest: \_Text**
1513. **nargs: Optional[Union[int, \_Text]]**
1514. **const: Any**
1515. **default: Any**
1516. **type: Union[Callable[[str], Any], FileType, None]**
1517. **choices: Optional[Iterable[Any]]**
1518. **required: bool**
1519. **help: Optional[\_Text]**
1520. **metavar: Union[\_Text, Tuple[\_Text, ...]]**
1521. **def \_\_init\_\_(self,**
1522. **option\_strings: Sequence[\_Text],**
1523. **dest: \_Text,**
1524. **nargs: Optional[Union[int, \_Text]] = ...,**
1525. **const: Any = ...,**
1526. **default: Any = ...,**
1527. **type: Optional[Union[Callable[[str], \_T], FileType]] = ...,**
1528. **choices: Optional[Iterable[\_T]] = ...,**
1529. **required: bool = ...,**
1530. **help: Optional[\_Text] = ...,**
1531. **metavar: Optional[Union[\_Text, Tuple[\_Text, ...]]] = ...) -> None: ...**
1532. **def \_\_call\_\_(self, parser: ArgumentParser, namespace: Namespace,**
1533. **values: Union[\_Text, Sequence[Any], None],**
1534. **option\_string: Optional[\_Text] = ...) -> None: ...**
1535. **class Namespace(\_AttributeHolder):**
1536. **def \_\_init\_\_(self, \*\*kwargs: Any) -> None: ...**
1537. **def \_\_getattr\_\_(self, name: \_Text) -> Any: ...**
1538. **def \_\_setattr\_\_(self, name: \_Text, value: Any) -> None: ...**
1539. **def \_\_contains\_\_(self, key: str) -> bool: ...**
1540. **class FileType:**
1541. **\_mode: \_Text**
1542. **\_bufsize: int**
1543. **if sys.version\_info >= (3, 4):**
1544. **\_encoding: Optional[\_Text]**
1545. **\_errors: Optional[\_Text]**
1546. **if sys.version\_info >= (3, 4):**
1547. **def \_\_init\_\_(self, mode: \_Text = ..., bufsize: int = ...,**
1548. **encoding: Optional[\_Text] = ...,**
1549. **errors: Optional[\_Text] = ...) -> None: ...**
1550. **elif sys.version\_info >= (3,):**
1551. **def \_\_init\_\_(self,**
1552. **mode: \_Text = ..., bufsize: int = ...) -> None: ...**
1553. **else:**
1554. **def \_\_init\_\_(self,**
1555. **mode: \_Text = ..., bufsize: Optional[int] = ...) -> None: ...**
1556. **def \_\_call\_\_(self, string: \_Text) -> IO[Any]: ...**
1557. **class \_ArgumentGroup(\_ActionsContainer):**
1558. **title: Optional[\_Text]**
1559. **\_group\_actions: List[Action]**
1560. **def \_\_init\_\_(self, container: \_ActionsContainer,**
1561. **title: Optional[\_Text] = ...,**
1562. **description: Optional[\_Text] = ..., \*\*kwargs: Any) -> None: ...**
1563. **class \_MutuallyExclusiveGroup(\_ArgumentGroup):**
1564. **required: bool**
1565. **\_container: \_ActionsContainer**
1566. **def \_\_init\_\_(self, container: \_ActionsContainer, required: bool = ...) -> None: ...**
1567. **class \_StoreAction(Action): ...**
1568. **class \_StoreConstAction(Action):**
1569. **def \_\_init\_\_(self,**
1570. **option\_strings: Sequence[\_Text],**
1571. **dest: \_Text,**
1572. **const: Any,**
1573. **default: Any = ...,**
1574. **required: bool = ...,**
1575. **help: Optional[\_Text] = ...,**
1576. **metavar: Optional[Union[\_Text, Tuple[\_Text, ...]]] = ...) -> None: ...**
1577. **class \_StoreTrueAction(\_StoreConstAction):**
1578. **def \_\_init\_\_(self,**
1579. **option\_strings: Sequence[\_Text],**
1580. **dest: \_Text,**
1581. **default: bool = ...,**
1582. **required: bool = ...,**
1583. **help: Optional[\_Text] = ...) -> None: ...**
1584. **class \_StoreFalseAction(\_StoreConstAction):**
1585. **def \_\_init\_\_(self,**
1586. **option\_strings: Sequence[\_Text],**
1587. **dest: \_Text,**
1588. **default: bool = ...,**
1589. **required: bool = ...,**
1590. **help: Optional[\_Text] = ...) -> None: ...**
1591. **class \_AppendAction(Action): ...**
1592. **class \_AppendConstAction(Action):**
1593. **def \_\_init\_\_(self,**
1594. **option\_strings: Sequence[\_Text],**
1595. **dest: \_Text,**
1596. **const: Any,**
1597. **default: Any = ...,**
1598. **required: bool = ...,**
1599. **help: Optional[\_Text] = ...,**
1600. **metavar: Optional[Union[\_Text, Tuple[\_Text, ...]]] = ...) -> None: ...**
1601. **class \_CountAction(Action):**
1602. **def \_\_init\_\_(self,**
1603. **option\_strings: Sequence[\_Text],**
1604. **dest: \_Text,**
1605. **default: Any = ...,**
1606. **required: bool = ...,**
1607. **help: Optional[\_Text] = ...) -> None: ...**
1608. **class \_HelpAction(Action):**
1609. **def \_\_init\_\_(self,**
1610. **option\_strings: Sequence[\_Text],**
1611. **dest: \_Text = ...,**
1612. **default: \_Text = ...,**
1613. **help: Optional[\_Text] = ...) -> None: ...**
1614. **class \_VersionAction(Action):**
1615. **version: Optional[\_Text]**
1616. **def \_\_init\_\_(self,**
1617. **option\_strings: Sequence[\_Text],**
1618. **version: Optional[\_Text] = ...,**
1619. **dest: \_Text = ...,**
1620. **default: \_Text = ...,**
1621. **help: \_Text = ...) -> None: ...**
1622. **class \_SubParsersAction(Action):**
1623. **\_ChoicesPseudoAction: Type[Any]**
1624. **\_prog\_prefix: \_Text**
1625. **\_parser\_class: Type[ArgumentParser]**
1626. **\_name\_parser\_map: Dict[\_Text, ArgumentParser]**
1627. **\_choices\_actions: List[Action]**
1628. **def \_\_init\_\_(self,**
1629. **option\_strings: Sequence[\_Text],**
1630. **prog: \_Text,**
1631. **parser\_class: Type[ArgumentParser],**
1632. **dest: \_Text = ...,**
1633. **required: bool = ...,**
1634. **help: Optional[\_Text] = ...,**
1635. **metavar: Optional[Union[\_Text, Tuple[\_Text, ...]]] = ...) -> None: ...**
1636. **def add\_parser(self, name: \_Text, \*\*kwargs: Any) -> ArgumentParser: ...**
1637. **def \_get\_subactions(self) -> List[Action]: ...**
1638. **class ArgumentTypeError(Exception): ...**
1639. **if sys.version\_info < (3, 7):**
1640. **def \_ensure\_value(namespace: Namespace, name: \_Text, value: Any) -> Any: ...**
1641. **def \_get\_action\_name(argument: Optional[Action]) -> Optional[str]: ...**
1642. **from \_\_future\_\_ import division, print\_function**
1643. **import numpy as np**
1644. **import tensorflow as tf**
1645. **def gpu\_nms(boxes, scores, num\_classes, max\_boxes=50, score\_thresh=0.5, nms\_thresh=0.5):**
1646. **boxes\_list, label\_list, score\_list = [], [], []**
1647. **max\_boxes = tf.constant(max\_boxes, dtype='int32')**
1648. **boxes = tf.reshape(boxes, [-1, 4])**
1649. **score = tf.reshape(scores, [-1, num\_classes])**
1650. **mask = tf.greater\_equal(score, tf.constant(score\_thresh))**
1651. **for i in range(num\_classes):**
1652. **filter\_boxes = tf.boolean\_mask(boxes, mask[:,i])**
1653. **filter\_score = tf.boolean\_mask(score[:,i], mask[:,i])**
1654. **nms\_indices = tf.image.non\_max\_suppression(boxes=filter\_boxes,**
1655. **scores=filter\_score,**
1656. **max\_output\_size=max\_boxes,**
1657. **iou\_threshold=nms\_thresh, name='nms\_indices')**
1658. **label\_list.append(tf.ones\_like(tf.gather(filter\_score, nms\_indices), 'int32')\*i)**
1659. **boxes\_list.append(tf.gather(filter\_boxes, nms\_indices))**
1660. **score\_list.append(tf.gather(filter\_score, nms\_indices))**
1661. **boxes = tf.concat(boxes\_list, axis=0)**
1662. **score = tf.concat(score\_list, axis=0)**
1663. **label = tf.concat(label\_list, axis=0)**
1664. **return boxes, score, label**
1665. **def py\_nms(boxes, scores, max\_boxes=50, iou\_thresh=0.5):**
1666. **assert boxes.shape[1] == 4 and len(scores.shape) == 1**
1667. **x1 = boxes[:, 0]**
1668. **y1 = boxes[:, 1]**
1669. **x2 = boxes[:, 2]**
1670. **y2 = boxes[:, 3]**
1671. **areas = (x2 - x1) \* (y2 - y1)**
1672. **order = scores.argsort()[::-1]**
1673. **keep = []**
1674. **while order.size > 0:**
1675. **i = order[0]**
1676. **keep.append(i)**
1677. **xx1 = np.maximum(x1[i], x1[order[1:]])**
1678. **yy1 = np.maximum(y1[i], y1[order[1:]])**
1679. **xx2 = np.minimum(x2[i], x2[order[1:]])**
1680. **yy2 = np.minimum(y2[i], y2[order[1:]])**
1681. **w = np.maximum(0.0, xx2 - xx1 + 1)**
1682. **h = np.maximum(0.0, yy2 - yy1 + 1)**
1683. **inter = w \* h**
1684. **ovr = inter / (areas[i] + areas[order[1:]] - inter)**
1685. **inds = np.where(ovr <= iou\_thresh)[0]**
1686. **order = order[inds + 1]**
1687. **return keep[:max\_boxes]**
1688. **def cpu\_nms(boxes, scores, num\_classes, max\_boxes=50, score\_thresh=0.5, iou\_thresh=0.5):**
1689. **boxes = boxes.reshape(-1, 4)**
1690. **scores = scores.reshape(-1, num\_classes)**
1691. **picked\_boxes, picked\_score, picked\_label = [], [], []**
1692. **for i in range(num\_classes):**
1693. **indices = np.where(scores[:,i] >= score\_thresh)**
1694. **filter\_boxes = boxes[indices]**
1695. **filter\_scores = scores[:,i][indices]**
1696. **if len(filter\_boxes) == 0:**
1697. **continue**
1698. **indices = py\_nms(filter\_boxes, filter\_scores,**
1699. **max\_boxes=max\_boxes, iou\_thresh=iou\_thresh)**
1700. **picked\_boxes.append(filter\_boxes[indices])**
1701. **picked\_score.append(filter\_scores[indices])**
1702. **picked\_label.append(np.ones(len(indices), dtype='int32')\*i)**
1703. **if len(picked\_boxes) == 0:**
1704. **return None, None, None**
1705. **boxes = np.concatenate(picked\_boxes, axis=0)**
1706. **score = np.concatenate(picked\_score, axis=0)**
1707. **label = np.concatenate(picked\_label, axis=0)**
1708. **return boxes, score, label**
1709. **from \_\_future\_\_ import division, print\_function**
1710. **import random**
1711. **import numpy as np**
1712. **import cv2**
1713. **def mix\_up(img1, img2, bbox1, bbox2):**
1714. **height = max(img1.shape[0], img2.shape[0])**
1715. **width = max(img1.shape[1], img2.shape[1])**
1716. **mix\_img = np.zeros(shape=(height, width, 3), dtype='float32')**
1717. **rand\_num = np.random.beta(1.5, 1.5)**
1718. **rand\_num = max(0, min(1, rand\_num))**
1719. **mix\_img[:img1.shape[0], :img1.shape[1], :] = img1.astype('float32') \* rand\_num**
1720. **mix\_img[:img2.shape[0], :img2.shape[1], :] += img2.astype('float32') \* (1. - rand\_num)**
1721. **mix\_img = mix\_img.astype('uint8')**
1722. **bbox1 = np.concatenate((bbox1, np.full(shape=(bbox1.shape[0], 1), fill\_value=rand\_num)), axis=-1)**
1723. **bbox2 = np.concatenate((bbox2, np.full(shape=(bbox2.shape[0], 1), fill\_value=1. - rand\_num)), axis=-1)**
1724. **mix\_bbox = np.concatenate((bbox1, bbox2), axis=0)**
1725. **return mix\_img, mix\_bbox**
1726. **def bbox\_crop(bbox, crop\_box=None, allow\_outside\_center=True):**
1727. **bbox = bbox.copy()**
1728. **if crop\_box is None:**
1729. **return bbox**
1730. **if not len(crop\_box) == 4:**
1731. **raise ValueError(**
1732. **"Invalid crop\_box parameter, requires length 4, given {}".format(str(crop\_box)))**
1733. **if sum([int(c is None) for c in crop\_box]) == 4:**
1734. **return bbox**
1735. **l, t, w, h = crop\_box**
1736. **left = l if l else 0**
1737. **top = t if t else 0**
1738. **right = left + (w if w else np.inf)**
1739. **bottom = top + (h if h else np.inf)**
1740. **crop\_bbox = np.array((left, top, right, bottom))**
1741. **if allow\_outside\_center:**
1742. **mask = np.ones(bbox.shape[0], dtype=bool)**
1743. **else:**
1744. **centers = (bbox[:, :2] + bbox[:, 2:4]) / 2**
1745. **mask = np.logical\_and(crop\_bbox[:2] <= centers, centers < crop\_bbox[2:]).all(axis=1)**
1746. **bbox[:, :2] = np.maximum(bbox[:, :2], crop\_bbox[:2])**
1747. **bbox[:, 2:4] = np.minimum(bbox[:, 2:4], crop\_bbox[2:4])**
1748. **bbox[:, :2] -= crop\_bbox[:2]**
1749. **bbox[:, 2:4] -= crop\_bbox[:2]**
1750. **mask = np.logical\_and(mask, (bbox[:, :2] < bbox[:, 2:4]).all(axis=1))**
1751. **bbox = bbox[mask]**
1752. **return bbox**
1753. **def bbox\_iou(bbox\_a, bbox\_b, offset=0):**
1754. **if bbox\_a.shape[1] < 4 or bbox\_b.shape[1] < 4:**
1755. **raise IndexError("Bounding boxes axis 1 must have at least length 4")**
1756. **tl = np.maximum(bbox\_a[:, None, :2], bbox\_b[:, :2])**
1757. **br = np.minimum(bbox\_a[:, None, 2:4], bbox\_b[:, 2:4])**
1758. **area\_i = np.prod(br - tl + offset, axis=2) \* (tl < br).all(axis=2)**
1759. **area\_a = np.prod(bbox\_a[:, 2:4] - bbox\_a[:, :2] + offset, axis=1)**
1760. **area\_b = np.prod(bbox\_b[:, 2:4] - bbox\_b[:, :2] + offset, axis=1)**
1761. **return area\_i / (area\_a[:, None] + area\_b - area\_i)**
1762. **def random\_crop\_with\_constraints(bbox, size, min\_scale=0.3, max\_scale=1,**
1763. **max\_aspect\_ratio=2, constraints=None,**
1764. **max\_trial=50):**
1765. **if constraints is None:**
1766. **constraints = (**
1767. **(0.1, None),**
1768. **(0.3, None),**
1769. **(0.5, None),**
1770. **(0.7, None),**
1771. **(0.9, None),**
1772. **(None, 1),**
1773. **)**
1774. **w, h = size**
1775. **candidates = [(0, 0, w, h)]**
1776. **for min\_iou, max\_iou in constraints:**
1777. **min\_iou = -np.inf if min\_iou is None else min\_iou**
1778. **max\_iou = np.inf if max\_iou is None else max\_iou**
1779. **for \_ in range(max\_trial):**
1780. **scale = random.uniform(min\_scale, max\_scale)**
1781. **aspect\_ratio = random.uniform(**
1782. **max(1 / max\_aspect\_ratio, scale \* scale),**
1783. **min(max\_aspect\_ratio, 1 / (scale \* scale)))**
1784. **crop\_h = int(h \* scale / np.sqrt(aspect\_ratio))**
1785. **crop\_w = int(w \* scale \* np.sqrt(aspect\_ratio))**
1786. **crop\_t = random.randrange(h - crop\_h)**
1787. **crop\_l = random.randrange(w - crop\_w)**
1788. **crop\_bb = np.array((crop\_l, crop\_t, crop\_l + crop\_w, crop\_t + crop\_h))**
1789. **if len(bbox) == 0:**
1790. **top, bottom = crop\_t, crop\_t + crop\_h**
1791. **left, right = crop\_l, crop\_l + crop\_w**
1792. **return bbox, (left, top, right-left, bottom-top)**
1793. **iou = bbox\_iou(bbox, crop\_bb[np.newaxis])**
1794. **if min\_iou <= iou.min() and iou.max() <= max\_iou:**
1795. **top, bottom = crop\_t, crop\_t + crop\_h**
1796. **left, right = crop\_l, crop\_l + crop\_w**
1797. **candidates.append((left, top, right-left, bottom-top))**
1798. **break**
1799. **while candidates:**
1800. **crop = candidates.pop(np.random.randint(0, len(candidates)))**
1801. **new\_bbox = bbox\_crop(bbox, crop, allow\_outside\_center=False)**
1802. **if new\_bbox.size < 1:**
1803. **continue**
1804. **new\_crop = (crop[0], crop[1], crop[2], crop[3])**
1805. **return new\_bbox, new\_crop**
1806. **return bbox, (0, 0, w, h)**
1807. **def random\_color\_distort(img, brightness\_delta=32, hue\_vari=18, sat\_vari=0.5, val\_vari=0.5):**
1808. **def random\_hue(img\_hsv, hue\_vari, p=0.5):**
1809. **if np.random.uniform(0, 1) > p:**
1810. **hue\_delta = np.random.randint(-hue\_vari, hue\_vari)**
1811. **img\_hsv[:, :, 0] = (img\_hsv[:, :, 0] + hue\_delta) % 180**
1812. **return img\_hsv**
1813. **def random\_saturation(img\_hsv, sat\_vari, p=0.5):**
1814. **if np.random.uniform(0, 1) > p:**
1815. **sat\_mult = 1 + np.random.uniform(-sat\_vari, sat\_vari)**
1816. **img\_hsv[:, :, 1] \*= sat\_mult**
1817. **return img\_hsv**
1818. **def random\_value(img\_hsv, val\_vari, p=0.5):**
1819. **if np.random.uniform(0, 1) > p:**
1820. **val\_mult = 1 + np.random.uniform(-val\_vari, val\_vari)**
1821. **img\_hsv[:, :, 2] \*= val\_mult**
1822. **return img\_hsv**
1823. **def random\_brightness(img, brightness\_delta, p=0.5):**
1824. **if np.random.uniform(0, 1) > p:**
1825. **img = img.astype(np.float32)**
1826. **brightness\_delta = int(np.random.uniform(-brightness\_delta, brightness\_delta))**
1827. **img = img + brightness\_delta**
1828. **return np.clip(img, 0, 255)**
1829. **img = random\_brightness(img, brightness\_delta)**
1830. **img = img.astype(np.uint8)**
1831. **img\_hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV).astype(np.float32)**
1832. **if np.random.randint(0, 2):**
1833. **img\_hsv = random\_value(img\_hsv, val\_vari)**
1834. **img\_hsv = random\_saturation(img\_hsv, sat\_vari)**
1835. **img\_hsv = random\_hue(img\_hsv, hue\_vari)**
1836. **else:**
1837. **img\_hsv = random\_saturation(img\_hsv, sat\_vari)**
1838. **img\_hsv = random\_hue(img\_hsv, hue\_vari)**
1839. **img\_hsv = random\_value(img\_hsv, val\_vari)**
1840. **img\_hsv = np.clip(img\_hsv, 0, 255)**
1841. **img = cv2.cvtColor(img\_hsv.astype(np.uint8), cv2.COLOR\_HSV2BGR)**
1842. **return img**
1843. **def letterbox\_resize(img, new\_width, new\_height, interp=0):**
1844. **ori\_height, ori\_width = img.shape[:2]**
1845. **resize\_ratio = min(new\_width / ori\_width, new\_height / ori\_height)**
1846. **resize\_w = int(resize\_ratio \* ori\_width)**
1847. **resize\_h = int(resize\_ratio \* ori\_height)**
1848. **img = cv2.resize(img, (resize\_w, resize\_h), interpolation=interp)**
1849. **image\_padded = np.full((new\_height, new\_width, 3), 128, np.uint8)**
1850. **dw = int((new\_width - resize\_w) / 2)**
1851. **dh = int((new\_height - resize\_h) / 2)**
1852. **image\_padded[dh: resize\_h + dh, dw: resize\_w + dw, :] = img**
1853. **return image\_padded, resize\_ratio, dw, dh**
1854. **def resize\_with\_bbox(img, bbox, new\_width, new\_height, interp=0, letterbox=False):**
1855. **if letterbox:**
1856. **image\_padded, resize\_ratio, dw, dh = letterbox\_resize(img, new\_width, new\_height, interp)**
1857. **bbox[:, [0, 2]] = bbox[:, [0, 2]] \* resize\_ratio + dw**
1858. **bbox[:, [1, 3]] = bbox[:, [1, 3]] \* resize\_ratio + dh**
1859. **return image\_padded, bbox**
1860. **else:**
1861. **ori\_height, ori\_width = img.shape[:2]**
1862. **img = cv2.resize(img, (new\_width, new\_height), interpolation=interp)**
1863. **bbox[:, [0, 2]] = bbox[:, [0, 2]] / ori\_width \* new\_width**
1864. **bbox[:, [1, 3]] = bbox[:, [1, 3]] / ori\_height \* new\_height**
1865. **return img, bbox**
1866. **def random\_flip(img, bbox, px=0, py=0):**
1867. **height, width = img.shape[:2]**
1868. **if np.random.uniform(0, 1) < px:**
1869. **img = cv2.flip(img, 1)**
1870. **xmax = width - bbox[:, 0]**
1871. **xmin = width - bbox[:, 2]**
1872. **bbox[:, 0] = xmin**
1873. **bbox[:, 2] = xmax**
1874. **if np.random.uniform(0, 1) < py:**
1875. **img = cv2.flip(img, 0)**
1876. **ymax = height - bbox[:, 1]**
1877. **ymin = height - bbox[:, 3]**
1878. **bbox[:, 1] = ymin**
1879. **bbox[:, 3] = ymax**
1880. **return img, bbox**
1881. **def random\_expand(img, bbox, max\_ratio=4, fill=0, keep\_ratio=True):**
1882. **h, w, c = img.shape**
1883. **ratio\_x = random.uniform(1, max\_ratio)**
1884. **if keep\_ratio:**
1885. **ratio\_y = ratio\_x**
1886. **else:**
1887. **ratio\_y = random.uniform(1, max\_ratio)**
1888. **oh, ow = int(h \* ratio\_y), int(w \* ratio\_x)**
1889. **off\_y = random.randint(0, oh - h)**
1890. **off\_x = random.randint(0, ow - w)**
1891. **dst = np.full(shape=(oh, ow, c), fill\_value=fill, dtype=img.dtype)**
1892. **dst[off\_y:off\_y + h, off\_x:off\_x + w, :] = img**
1893. **bbox[:, :2] += (off\_x, off\_y)**
1894. **bbox[:, 2:4] += (off\_x, off\_y)**
1895. **return dst, bbox**
1896. **from \_\_future\_\_ import division, print\_function**
1897. **import tensorflow as tf**
1898. **slim = tf.contrib.slim**
1899. **from utils.layer\_utils import conv2d, darknet53\_body, yolo\_block, upsample\_layer**
1900. **class yolov3(object):**
1901. **def \_\_init\_\_(self, class\_num, anchors, use\_label\_smooth=False, use\_focal\_loss=False, batch\_norm\_decay=0.999,**
1902. **weight\_decay=5e-4, use\_static\_shape=True):**
1903. **self.class\_num = class\_num**
1904. **self.anchors = anchors**
1905. **self.batch\_norm\_decay = batch\_norm\_decay**
1906. **self.use\_label\_smooth = use\_label\_smooth**
1907. **self.use\_focal\_loss = use\_focal\_loss**
1908. **self.weight\_decay = weight\_decay**
1909. **self.use\_static\_shape = use\_static\_shape**
1910. **def forward(self, inputs, is\_training=False, reuse=False):**
1911. **self.img\_size = tf.shape(inputs)[1:3]**
1912. **batch\_norm\_params = {**
1913. **'decay': self.batch\_norm\_decay,**
1914. **'epsilon': 1e-05,**
1915. **'scale': True,**
1916. **'is\_training': is\_training,**
1917. **'fused': None,**
1918. **}**
1919. **with slim.arg\_scope([slim.conv2d, slim.batch\_norm], reuse=reuse):**
1920. **with slim.arg\_scope([slim.conv2d],**
1921. **normalizer\_fn=slim.batch\_norm,**
1922. **normalizer\_params=batch\_norm\_params,**
1923. **biases\_initializer=None,**
1924. **activation\_fn=lambda x: tf.nn.leaky\_relu(x, alpha=0.1),**
1925. **weights\_regularizer=slim.l2\_regularizer(self.weight\_decay)):**
1926. **with tf.variable\_scope('darknet53\_body'):**
1927. **route\_1, route\_2, route\_3 = darknet53\_body(inputs)**
1928. **with tf.variable\_scope('yolov3\_head'):**
1929. **inter1, net = yolo\_block(route\_3, 512)**
1930. **feature\_map\_1 = slim.conv2d(net, 3 \* (5 + self.class\_num), 1,**
1931. **stride=1, normalizer\_fn=None,**
1932. **activation\_fn=None, biases\_initializer=tf.zeros\_initializer())**
1933. **feature\_map\_1 = tf.identity(feature\_map\_1, name='feature\_map\_1')**
1934. **inter1 = conv2d(inter1, 256, 1)**
1935. **inter1 = upsample\_layer(inter1,**
1936. **route\_2.get\_shape().as\_list() if self.use\_static\_shape else tf.shape(**
1937. **route\_2))**
1938. **concat1 = tf.concat([inter1, route\_2], axis=3)**
1939. **inter2, net = yolo\_block(concat1, 256)**
1940. **feature\_map\_2 = slim.conv2d(net, 3 \* (5 + self.class\_num), 1,**
1941. **stride=1, normalizer\_fn=None,**
1942. **activation\_fn=None, biases\_initializer=tf.zeros\_initializer())**
1943. **feature\_map\_2 = tf.identity(feature\_map\_2, name='feature\_map\_2')**
1944. **inter2 = conv2d(inter2, 128, 1)**
1945. **inter2 = upsample\_layer(inter2,**
1946. **route\_1.get\_shape().as\_list() if self.use\_static\_shape else tf.shape(**
1947. **route\_1))**
1948. **concat2 = tf.concat([inter2, route\_1], axis=3)**
1949. **\_, feature\_map\_3 = yolo\_block(concat2, 128)**
1950. **feature\_map\_3 = slim.conv2d(feature\_map\_3, 3 \* (5 + self.class\_num), 1,**
1951. **stride=1, normalizer\_fn=None,**
1952. **activation\_fn=None, biases\_initializer=tf.zeros\_initializer())**
1953. **feature\_map\_3 = tf.identity(feature\_map\_3, name='feature\_map\_3')**
1954. **return feature\_map\_1, feature\_map\_2, feature\_map\_3**
1955. **def reorg\_layer(self, feature\_map, anchors):**
1956. **grid\_size = feature\_map.get\_shape().as\_list()[1:3] if self.use\_static\_shape else tf.shape(feature\_map)[**
1957. **1:3]**
1958. **ratio = tf.cast(self.img\_size / grid\_size, tf.float32)**
1959. **rescaled\_anchors = [(anchor[0] / ratio[1], anchor[1] / ratio[0]) for anchor in anchors]**
1960. **feature\_map = tf.reshape(feature\_map, [-1, grid\_size[0], grid\_size[1], 3, 5 + self.class\_num])**
1961. **box\_centers, box\_sizes, conf\_logits, prob\_logits = tf.split(feature\_map, [2, 2, 1, self.class\_num], axis=-1)**
1962. **box\_centers = tf.nn.sigmoid(box\_centers)**
1963. **grid\_x = tf.range(grid\_size[1], dtype=tf.int32)**
1964. **grid\_y = tf.range(grid\_size[0], dtype=tf.int32)**
1965. **grid\_x, grid\_y = tf.meshgrid(grid\_x, grid\_y)**
1966. **x\_offset = tf.reshape(grid\_x, (-1, 1))**
1967. **y\_offset = tf.reshape(grid\_y, (-1, 1))**
1968. **x\_y\_offset = tf.concat([x\_offset, y\_offset], axis=-1)**
1969. **x\_y\_offset = tf.cast(tf.reshape(x\_y\_offset, [grid\_size[0], grid\_size[1], 1, 2]), tf.float32)**
1970. **box\_centers = box\_centers + x\_y\_offset**
1971. **box\_centers = box\_centers \* ratio[::-1]**
1972. **box\_sizes = tf.exp(box\_sizes) \* rescaled\_anchors**
1973. **box\_sizes = box\_sizes \* ratio[::-1]**
1974. **boxes = tf.concat([box\_centers, box\_sizes], axis=-1)**
1975. **return x\_y\_offset, boxes, conf\_logits, prob\_logits**
1976. **def predict(self, feature\_maps):**
1977. **feature\_map\_1, feature\_map\_2, feature\_map\_3 = feature\_maps**
1978. **feature\_map\_anchors = [(feature\_map\_1, self.anchors[6:9]),**
1979. **(feature\_map\_2, self.anchors[3:6]),**
1980. **(feature\_map\_3, self.anchors[0:3])]**
1981. **reorg\_results = [self.reorg\_layer(feature\_map, anchors) for (feature\_map, anchors) in feature\_map\_anchors]**
1982. **def \_reshape(result):**
1983. **x\_y\_offset, boxes, conf\_logits, prob\_logits = result**
1984. **grid\_size = x\_y\_offset.get\_shape().as\_list()[:2] if self.use\_static\_shape else tf.shape(x\_y\_offset)[:2]**
1985. **boxes = tf.reshape(boxes, [-1, grid\_size[0] \* grid\_size[1] \* 3, 4])**
1986. **conf\_logits = tf.reshape(conf\_logits, [-1, grid\_size[0] \* grid\_size[1] \* 3, 1])**
1987. **prob\_logits = tf.reshape(prob\_logits, [-1, grid\_size[0] \* grid\_size[1] \* 3, self.class\_num])**
1988. **return boxes, conf\_logits, prob\_logits**
1989. **boxes\_list, confs\_list, probs\_list = [], [], []**
1990. **for result in reorg\_results:**
1991. **boxes, conf\_logits, prob\_logits = \_reshape(result)**
1992. **confs = tf.sigmoid(conf\_logits)**
1993. **probs = tf.sigmoid(prob\_logits)**
1994. **boxes\_list.append(boxes)**
1995. **confs\_list.append(confs)**
1996. **probs\_list.append(probs)**
1997. **boxes = tf.concat(boxes\_list, axis=1)**
1998. **confs = tf.concat(confs\_list, axis=1)**
1999. **probs = tf.concat(probs\_list, axis=1)**
2000. **center\_x, center\_y, width, height = tf.split(boxes, [1, 1, 1, 1], axis=-1)**
2001. **x\_min = center\_x - width / 2**
2002. **y\_min = center\_y - height / 2**
2003. **x\_max = center\_x + width / 2**
2004. **y\_max = center\_y + height / 2**
2005. **boxes = tf.concat([x\_min, y\_min, x\_max, y\_max], axis=-1)**
2006. **return boxes, confs, probs**
2007. **def loss\_layer(self, feature\_map\_i, y\_true, anchors):**
2008. **grid\_size = tf.shape(feature\_map\_i)[1:3]**
2009. **ratio = tf.cast(self.img\_size / grid\_size, tf.float32)**
2010. **N = tf.cast(tf.shape(feature\_map\_i)[0], tf.float32)**
2011. **x\_y\_offset, pred\_boxes, pred\_conf\_logits, pred\_prob\_logits = self.reorg\_layer(feature\_map\_i, anchors)**
2012. **object\_mask = y\_true[..., 4:5]**
2013. **ignore\_mask = tf.TensorArray(tf.float32, size=0, dynamic\_size=True)**
2014. **def loop\_cond(idx, ignore\_mask):**
2015. **return tf.less(idx, tf.cast(N, tf.int32))**
2016. **def loop\_body(idx, ignore\_mask):**
2017. **valid\_true\_boxes = tf.boolean\_mask(y\_true[idx, ..., 0:4], tf.cast(object\_mask[idx, ..., 0], 'bool'))**
2018. **iou = self.box\_iou(pred\_boxes[idx], valid\_true\_boxes)**
2019. **best\_iou = tf.reduce\_max(iou, axis=-1)**
2020. **ignore\_mask\_tmp = tf.cast(best\_iou < 0.5, tf.float32)**
2021. **ignore\_mask = ignore\_mask.write(idx, ignore\_mask\_tmp)**
2022. **return idx + 1, ignore\_mask**
2023. **\_, ignore\_mask = tf.while\_loop(cond=loop\_cond, body=loop\_body, loop\_vars=[0, ignore\_mask])**
2024. **ignore\_mask = ignore\_mask.stack()**
2025. **ignore\_mask = tf.expand\_dims(ignore\_mask, -1)**
2026. **pred\_box\_xy = pred\_boxes[..., 0:2]**
2027. **pred\_box\_wh = pred\_boxes[..., 2:4]**
2028. **true\_xy = y\_true[..., 0:2] / ratio[::-1] - x\_y\_offset**
2029. **pred\_xy = pred\_box\_xy / ratio[::-1] - x\_y\_offset**
2030. **true\_tw\_th = y\_true[..., 2:4] / anchors**
2031. **pred\_tw\_th = pred\_box\_wh / anchors**
2032. **true\_tw\_th = tf.where(condition=tf.equal(true\_tw\_th, 0),**
2033. **x=tf.ones\_like(true\_tw\_th), y=true\_tw\_th)**
2034. **pred\_tw\_th = tf.where(condition=tf.equal(pred\_tw\_th, 0),**
2035. **x=tf.ones\_like(pred\_tw\_th), y=pred\_tw\_th)**
2036. **true\_tw\_th = tf.log(tf.clip\_by\_value(true\_tw\_th, 1e-9, 1e9))**
2037. **pred\_tw\_th = tf.log(tf.clip\_by\_value(pred\_tw\_th, 1e-9, 1e9))**
2038. **box\_loss\_scale = 2. - (y\_true[..., 2:3] / tf.cast(self.img\_size[1], tf.float32)) \* (**
2039. **y\_true[..., 3:4] / tf.cast(self.img\_size[0], tf.float32))**
2040. **mix\_w = y\_true[..., -1:]**
2041. **xy\_loss = tf.reduce\_sum(tf.square(true\_xy - pred\_xy) \* object\_mask \* box\_loss\_scale \* mix\_w) / N**
2042. **wh\_loss = tf.reduce\_sum(tf.square(true\_tw\_th - pred\_tw\_th) \* object\_mask \* box\_loss\_scale \* mix\_w) / N**
2043. **conf\_pos\_mask = object\_mask**
2044. **conf\_neg\_mask = (1 - object\_mask) \* ignore\_mask**
2045. **conf\_loss\_pos = conf\_pos\_mask \* tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=object\_mask,**
2046. **logits=pred\_conf\_logits)**
2047. **conf\_loss\_neg = conf\_neg\_mask \* tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=object\_mask,**
2048. **logits=pred\_conf\_logits)**
2049. **conf\_loss = conf\_loss\_pos + conf\_loss\_neg**
2050. **if self.use\_focal\_loss:**
2051. **alpha = 1.0**
2052. **gamma = 2.0**
2053. **focal\_mask = alpha \* tf.pow(tf.abs(object\_mask - tf.sigmoid(pred\_conf\_logits)), gamma)**
2054. **conf\_loss \*= focal\_mask**
2055. **conf\_loss = tf.reduce\_sum(conf\_loss \* mix\_w) / N**
2056. **if self.use\_label\_smooth:**
2057. **delta = 0.01**
2058. **label\_target = (1 - delta) \* y\_true[..., 5:-1] + delta \* 1. / self.class\_num**
2059. **else:**
2060. **label\_target = y\_true[..., 5:-1]**
2061. **class\_loss = object\_mask \* tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=label\_target,**
2062. **logits=pred\_prob\_logits) \* mix\_w**
2063. **class\_loss = tf.reduce\_sum(class\_loss) / N**
2064. **return xy\_loss, wh\_loss, conf\_loss, class\_loss**
2065. **def box\_iou(self, pred\_boxes, valid\_true\_boxes):**
2066. **pred\_box\_xy = pred\_boxes[..., 0:2]**
2067. **pred\_box\_wh = pred\_boxes[..., 2:4]**
2068. **pred\_box\_xy = tf.expand\_dims(pred\_box\_xy, -2)**
2069. **pred\_box\_wh = tf.expand\_dims(pred\_box\_wh, -2)**
2070. **true\_box\_xy = valid\_true\_boxes[:, 0:2]**
2071. **true\_box\_wh = valid\_true\_boxes[:, 2:4]**
2072. **intersect\_mins = tf.maximum(pred\_box\_xy - pred\_box\_wh / 2.,**
2073. **true\_box\_xy - true\_box\_wh / 2.)**
2074. **intersect\_maxs = tf.minimum(pred\_box\_xy + pred\_box\_wh / 2.,**
2075. **true\_box\_xy + true\_box\_wh / 2.)**
2076. **intersect\_wh = tf.maximum(intersect\_maxs - intersect\_mins, 0.)**
2077. **intersect\_area = intersect\_wh[..., 0] \* intersect\_wh[..., 1]**
2078. **pred\_box\_area = pred\_box\_wh[..., 0] \* pred\_box\_wh[..., 1]**
2079. **true\_box\_area = true\_box\_wh[..., 0] \* true\_box\_wh[..., 1]**
2080. **true\_box\_area = tf.expand\_dims(true\_box\_area, axis=0)**
2081. **iou = intersect\_area / (pred\_box\_area + true\_box\_area - intersect\_area + 1e-10)**
2082. **return iou**
2083. **def compute\_loss(self, y\_pred, y\_true):**
2084. **loss\_xy, loss\_wh, loss\_conf, loss\_class = 0., 0., 0., 0.**
2085. **anchor\_group = [self.anchors[6:9], self.anchors[3:6], self.anchors[0:3]]**
2086. **for i in range(len(y\_pred)):**
2087. **result = self.loss\_layer(y\_pred[i], y\_true[i], anchor\_group[i])**
2088. **loss\_xy += result[0]**
2089. **loss\_wh += result[1]**
2090. **loss\_conf += result[2]**
2091. **loss\_class += result[3]**
2092. **total\_loss = loss\_xy + loss\_wh + loss\_conf + loss\_class**
2093. **return [total\_loss, loss\_xy, loss\_wh, loss\_conf, loss\_class]**
2094. **import socket**
2095. **import time**
2096. **import sys**
2097. **import cv2**
2098. **import numpy as np**
2099. **import urllib**
2100. **import threading**
2101. **import signal**
2102. **import LeArm**
2103. **import kinematics as kin**
2104. **import RPi.GPIO as GPIO**
2105. **HOST\_IP = "0.0.0.0"**
2106. **HOST\_PORT = 8888**
2107. **print("Starting socket: TCP...")**
2108. **socket\_tcp = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**
2109. **print("TCP server listen @ %s:%d!" % (HOST\_IP, HOST\_PORT))**
2110. **host\_addr = (HOST\_IP, HOST\_PORT)**
2111. **socket\_tcp.bind(host\_addr)**
2112. **socket\_tcp.listen(1)**
2113. **socket\_con, (client\_ip, client\_port) = socket\_tcp.accept()**
2114. **print("Connection accepted from %s." % client\_ip)**
2115. **print("Receiving package...")**
2116. **data1 = "finish"**
2117. **triangle = 0**
2118. **circle = 0**
2119. **square = 0**
2120. **while True:**
2121. **try:**
2122. **data = socket\_con.recv(512)**
2123. **if len(data) > 0:**
2124. **if data == "init":**
2125. **print("arm init...")**
2126. **LeArm.runActionGroup('init', 1)**
2127. **print("arm init complete...")**
2128. **socket\_con.send(data1.encode("utf-8"))**
2129. **elif data == "apple":**
2130. **print("apple...")**
2131. **if triangle == 0:**
2132. **LeArm.runActionGroup('apple1', 1)**
2133. **print("apple complete")**
2134. **triangle = 1**
2135. **elif triangle == 1:**
2136. **LeArm.runActionGroup('apple2', 1)**
2137. **print("apple complete")**
2138. **triangle = 0**
2139. **LeArm.runActionGroup('init', 1)**
2140. **socket\_con.send(data1.encode("utf-8"))**
2141. **elif data == "peach":**
2142. **print("peach...")**
2143. **if square == 0:**
2144. **LeArm.runActionGroup('peach1', 1)**
2145. **print("peach complete")**
2146. **square = 1**
2147. **elif square == 1:**
2148. **LeArm.runActionGroup('peach2', 1)**
2149. **print("peach complete")**
2150. **square = 0**
2151. **LeArm.runActionGroup('init', 1)**
2152. **socket\_con.send(data1.encode("utf-8"))**
2153. **elif data == "mango":**
2154. **print("mango...")**
2155. **if circle == 0:**
2156. **LeArm.runActionGroup('mango1', 1)**
2157. **print("mango complete")**
2158. **circle = 1**
2159. **elif circle == 1:**
2160. **LeArm.runActionGroup('mango2', 1)**
2161. **print("mango complete")**
2162. **circle = 0**
2163. **LeArm.runActionGroup('init', 1)**
2164. **socket\_con.send(data1.encode("utf-8"))**
2165. **elif data == "strawberry":**
2166. **print("strawberry...")**
2167. **if circle == 0:**
2168. **LeArm.runActionGroup('strawberry1', 1)**
2169. **print("strawberry complete")**
2170. **circle = 1**
2171. **elif circle == 1:**
2172. **LeArm.runActionGroup('strawberry2', 1)**
2173. **print("strawberry complete")**
2174. **circle = 0**
2175. **LeArm.runActionGroup('init', 1)**
2176. **socket\_con.send(data1.encode("utf-8"))**
2177. **continue**
2178. **except Exception:**
2179. **socket\_tcp.close()**
2180. **sys.exit(1)**
2181. **import pigpio**
2182. **import thread**
2183. **import time**
2184. **import threading**
2185. **class PWM\_Servo(object):**
2186. **def \_\_init\_\_(self, pi, pin, freq=50, min\_width=500, max\_width=2500, deviation=0, control\_speed=False):**
2187. **self.pi = pi**
2188. **self.SPin = pin**
2189. **self.Freq = freq**
2190. **self.Min = min\_width**
2191. **self.Max = max\_width**
2192. **self.Deviation = deviation**
2193. **self.speedControl = control\_speed**
2194. **self.Position = 1500**
2195. **self.positionSet = self.Position**
2196. **self.stepTime = 20**
2197. **self.positionInc = 0.0**
2198. **self.Time = 0**
2199. **self.Time\_t = 0**
2200. **self.incTimes = 0**
2201. **self.positionSet\_t = 0**
2202. **self.posChanged = False**
2203. **self.servoRunning = False**
2204. **self.pi.set\_PWM\_range(self.SPin, 1000000 / self.Freq)**
2205. **self.pi.set\_PWM\_frequency(self.SPin, self.Freq)**
2206. **if control\_speed is True:**
2207. **def setPosition(self, pos, time=0):**
2208. **if pos < self.Min or pos > self.Max:**
2209. **print(pos)**
2210. **return**
2211. **if time == 0:**
2212. **self.Position = pos**
2213. **self.positionSet = self.Position**
2214. **self.pi.set\_PWM\_dutycycle(self.SPin, self.Position + self.Deviation)**
2215. **else:**
2216. **if time < 20:**
2217. **self.Time\_t = 20**
2218. **elif time > 30000:**
2219. **self.Time\_t = 30000**
2220. **else:**
2221. **self.Time\_t = time**
2222. **self.positionSet\_t = pos**
2223. **self.posChanged = True**
2224. **def getPosition(self):**
2225. **return self.Position**
2226. **def updatePosition(self):**
2227. **while True:**
2228. **if self.posChanged is True:**
2229. **self.Time = self.Time\_t**
2230. **self.positionSet = self.positionSet\_t**
2231. **self.posChanged = False**
2232. **self.incTimes = self.Time / self.stepTime**
2233. **if self.positionSet > self.Position:**
2234. **self.positionInc = self.positionSet - self.Position**
2235. **self.positionInc = -self.positionInc**
2236. **else:**
2237. **self.positionInc = self.Position - self.positionSet**
2238. **self.positionInc = self.positionInc / float(self.incTimes)**
2239. **self.servoRunning = True**
2240. **if self.servoRunning is True:**
2241. **self.incTimes -= 1**
2242. **if self.incTimes == 0:**
2243. **self.Position = self.positionSet**
2244. **self.servoRunning = False**
2245. **else:**
2246. **self.Position = self.positionSet + self.positionInc \* self.incTimes**
2247. **self.pi.set\_PWM\_dutycycle(self.SPin, self.Position + self.Deviation)**
2248. **time.sleep(0.02)**
2249. **def setDeviation(newD=0):**
2250. **if newD > 300 or newD < -300:**
2251. **return**
2252. **self.Deviation = newD**
2253. **import SocketServer**
2254. **import threading**
2255. **import time**
2256. **import re**
2257. **import LeArm**
2258. **import LeConf**
2259. **from LeCmd import LeError**
2260. **import LeCmd**
2261. **class ServoServer(SocketServer.BaseRequestHandler):**
2262. **def handle(self):**
2263. **print("已连接")**
2264. **conn = self.request**
2265. **Flag = True**
2266. **recv\_data = ""**
2267. **while Flag:**
2268. **try:**
2269. **recv\_data += conn.recv(1024)**
2270. **if not recv\_data:**
2271. **Flag = False**
2272. **print("break")**
2273. **break**
2274. **recv\_data = recv\_data.replace(' ', '')**
2275. **cp = re.compile(r'\r\n')**
2276. **test = cp.search(recv\_data)**
2277. **if test:**
2278. **rdata = recv\_data.split("\r\n")**
2279. **recv\_data = recv\_data[len(rdata[0]) + 2:]**
2280. **rdata = [rdata[0]]**
2281. **for data in rdata:**
2282. **if data:**
2283. **rex = re.compile(r'^(I[0-9]{3}).\*')**
2284. **match = data**
2285. **match = rex.match(match)**
2286. **if match:**
2287. **if not 0 == match.start() or not len(data) == match.end():**
2288. **print("错误指令 1")**
2289. **else:**
2290. **data = data.split('-')**
2291. **cmd = data[0][1:5]**
2292. **del data[0]**
2293. **par = []**
2294. **try:**
2295. **cmd = int(cmd)**
2296. **if cmd >= 3 and cmd <= 7:**
2297. **LeCmd.cmd\_list[cmd](conn, data)**
2298. **else:**
2299. **for p in data:**
2300. **par.append(int(p))**
2301. **LeCmd.cmd\_list[cmd](par)**
2302. **except LeError as err:**
2303. **print(err.msg)**
2304. **print(err.data)**
2305. **except:**
2306. **print("指令执行错误")**
2307. **else:**
2308. **print(data)**
2309. **print("错误指令 2")**
2310. **else:**
2311. **pass**
2312. **if not Flag:**
2313. **print("break1")**
2314. **break**
2315. **except Exception as e:**
2316. **print(e)**
2317. **break**
2318. **def finish(self):**
2319. **print("已断开")**
2320. **class LeServer(SocketServer.ThreadingTCPServer):**
2321. **allow\_reuse\_address = True**
2322. **if \_\_name\_\_ == "\_\_main\_\_":**
2323. **if not len(LeConf.Deviation) == 6:**
2324. **print("偏差数量错误")**
2325. **sys.exit()**
2326. **else:**
2327. **d = []**
2328. **for i in range(0, len(LeConf.Deviation), 1):**
2329. **if LeConf.Deviation[i] > 1600 or LeConf.Deviation < 1400:**
2330. **print("偏差值超出范围1200-1800")**
2331. **sys.exit()**
2332. **else:**
2333. **d.append(LeConf.Deviation[i] - 1500)**
2334. **LeArm.initLeArm(tuple(d))**
2335. **LeCmd.cmd\_i001([1000, 6, 1, 1500, 2, 1500, 3, 1500, 4, 1500, 5, 1500, 6, 1500])**
2336. **server = LeServer(("", 8947), ServoServer)**
2337. **try:**
2338. **server\_thread = threading.Thread(target=server.serve\_forever)**
2339. **server\_thread.start()**
2340. **while True:**
2341. **time.sleep(0.1)**
2342. **except:**
2343. **server.shutdown()**
2344. **server.server\_close()**
2345. **LeArm.stopLeArm()**
2346. **import pygame**
2347. **import time**
2348. **import os**
2349. **from socket import \***
2350. **HOST = "127.0.0.1"**
2351. **PORT = 8947**
2352. **step\_width = 40**
2353. **key\_map = {"PSB\_CROSS": 2, "PSB\_CIRCLE": 1, "PSB\_SQUARE": 3, "PSB\_TRIANGLE": 0,**
2354. **"PSB\_L1": 4, "PSB\_R1": 5, "PSB\_L2": 6, "PSB\_R2": 7,**
2355. **"PSB\_SELECT": 8, "PSB\_START": 9, "PSB\_L3": 10, "PSB\_R3": 11};**
2356. **action\_map = ["CROSS", "CIRCLE", "", "SQUARE", "TRIANGLE", "L1", "R1", "L2", "R2", "SELECT", "START", "", "L3", "R3"]**
2357. **pygame.init()**
2358. **os.environ["SDL\_VIDEODRIVER"] = "dummy"**
2359. **pygame.display.init()**
2360. **pygame.joystick.init()**
2361. **if pygame.joystick.get\_count() > 0:**
2362. **js = pygame.joystick.Joystick(0)**
2363. **js.init()**
2364. **jsName = js.get\_name()**
2365. **print("Name of the joystick:", jsName)**
2366. **jsAxes = js.get\_numaxes()**
2367. **print("Number of axis:", jsAxes)**
2368. **jsButtons = js.get\_numbuttons()**
2369. **print("Number of buttons:", jsButtons);**
2370. **jsBall = js.get\_numballs()**
2371. **print("Numbe of balls:", jsBall)**
2372. **jsHat = js.get\_numhats()**
2373. **print("Number of hats:", jsHat)**
2374. **pygame.joystick.init()**
2375. **connected = False**
2376. **count = 0**
2377. **while True:**
2378. **if os.path.exists("/dev/input/js0") is True:**
2379. **if connected is False:**
2380. **pygame.joystick.quit()**
2381. **pygame.joystick.init()**
2382. **jscount = pygame.joystick.get\_count()**
2383. **if jscount > 0:**
2384. **try:**
2385. **js = pygame.joystick.Joystick(0)**
2386. **js.init()**
2387. **js.get\_hat(0)**
2388. **client = socket(AF\_INET, SOCK\_STREAM)**
2389. **client.connect((HOST, PORT))**
2390. **connected = True**
2391. **except Exception as e:**
2392. **print(e)**
2393. **else:**
2394. **pygame.joystick.quit()**
2395. **else:**
2396. **pass**
2397. **else:**
2398. **if connected is True:**
2399. **connected = False**
2400. **js.quit();**
2401. **pygame.joystick.quit()**
2402. **client.close()**
2403. **else:**
2404. **pass**
2405. **if connected is True:**
2406. **pygame.event.pump()**
2407. **change = [0, 0, 0, 0, 0, 0]**
2408. **try:**
2409. **if js.get\_button(key\_map["PSB\_R1"]):**
2410. **change[0] -= step\_width**
2411. **if js.get\_button(key\_map["PSB\_R2"]):**
2412. **change[0] += step\_width**
2413. **if js.get\_button(key\_map["PSB\_SQUARE"]):**
2414. **change[1] -= step\_width**
2415. **if js.get\_button(key\_map["PSB\_CIRCLE"]):**
2416. **change[1] += step\_width**
2417. **if js.get\_button(key\_map["PSB\_TRIANGLE"]):**
2418. **change[2] -= step\_width**
2419. **if js.get\_button(key\_map["PSB\_CROSS"]):**
2420. **change[2] += step\_width**
2421. **if js.get\_button(key\_map["PSB\_L1"]):**
2422. **change[3] -= step\_width**
2423. **if js.get\_button(key\_map["PSB\_L2"]):**
2424. **change[3] += step\_width**
2425. **hat = js.get\_hat(0)**
2426. **if hat[0] > 0:**
2427. **change[5] += step\_width**
2428. **elif hat[0] < 0:**
2429. **change[5] -= step\_width**
2430. **else:**
2431. **pass**
2432. **if hat[1] > 0:**
2433. **change[4] += step\_width**
2434. **elif hat[1] < 0:**
2435. **change[4] -= step\_width**
2436. **else:**
2437. **pass**
2438. **lx = js.get\_axis(0)**
2439. **ly = js.get\_axis(1)**
2440. **rx = js.get\_axis(2)**
2441. **ry = js.get\_axis(3)**
2442. **if lx > 0.5:**
2443. **change[5] += step\_width**
2444. **elif lx < -0.5:**
2445. **change[5] -= step\_width**
2446. **else:**
2447. **pass**
2448. **l3\_state = js.get\_button(key\_map["PSB\_L3"])**
2449. **if ly > 0.5:**
2450. **if l3\_state:**
2451. **change[4] -= step\_width**
2452. **else:**
2453. **change[3] += step\_width**
2454. **elif ly < -0.5:**
2455. **if l3\_state:**
2456. **change[4] += step\_width**
2457. **else:**
2458. **change[3] -= step\_width**
2459. **else:**
2460. **pass**
2461. **if rx > 0.5:**
2462. **change[1] += step\_width**
2463. **elif rx < -0.5:**
2464. **change[1] -= step\_width**
2465. **else:**
2466. **pass**
2467. **if ry > 0.5:**
2468. **change[2] += step\_width**
2469. **elif ry < -0.5:**
2470. **change[2] -= step\_width**
2471. **else:**
2472. **pass**
2473. **cmd = ""**
2474. **cu = 0;**
2475. **for i in range(6):**
2476. **if not change[i] is 0:**
2477. **cu += 1**
2478. **change[i] += 10000**
2479. **cmd += '-' + str(i + 1) + '-' + str(change[i])**
2480. **if cu > 0 and not js.get\_button(key\_map["PSB\_START"]):**
2481. **cmd = "I007-" + "50-" + str(cu) + cmd + "\r\n"**
2482. **print(cmd)**
2483. **client.send(cmd.encode())**
2484. **elif js.get\_button(key\_map["PSB\_START"]):**
2485. **cmd = "I001-1000-6-1-1500-2-1500-3-1500-4-1500-5-1500-6-1500\r\n"**
2486. **client.send(cmd.encode())**
2487. **else:**
2488. **pass**
2489. **except Exception as e:**
2490. **print(e)**
2491. **connected = False**
2492. **client.close()**
2493. **time.sleep(0.06)**
2494. **Deviation = (1500, 1500, 1500, 1500, 1500, 1500)**
2495. **import os**
2496. **import re**
2497. **import LeArm**
2498. **import LeConf**
2499. **import LeActList**
2500. **import threading**
2501. **actdir = "./ActionGroups/"**
2502. **class LeError(Exception):**
2503. **def \_\_init\_\_(self, data=(), msg="LeError"):**
2504. **self.data = data**
2505. **self.msg = msg**
2506. **def cmd\_i000(par):**
2507. **pass**
2508. **def cmd\_i001(par):**
2509. **global Deviation**
2510. **if par[0] > 30000:**
2511. **par[0] = 30000**
2512. **if par[0] < 20:**
2513. **par[0] = 20**
2514. **if not par[1] \* 2 + 2 == len(par) or not len(par) >= 4:**
2515. **raise LeError(tuple(par), "舵机运动指令长度错误")**
2516. **Servos = par[2:]**
2517. **for i in range(0, len(Servos), 2):**
2518. **if Servos[i] > 6 or Servos[i] < 1 or Servos[i + 1] > 2500 or Servos[i + 1] < 500:**
2519. **raise LeError((Servos[i], Servos[i + 1]), "舵机运动参数错误")**
2520. **LeArm.setServo(Servos[i], Servos[i + 1], par[0])**
2521. **def cmd\_i002(par):**
2522. **LeArm.stopActionGroup()**
2523. **def cmd\_i003(sock, data=["", ""]):**
2524. **if (not len(data) == 2) or (not data) or (not data[0]) or (not data[1]):**
2525. **raise LeError(tuple(data), "运行动作组指令错误")**
2526. **par = None**
2527. **try:**
2528. **par = int(data[1])**
2529. **except:**
2530. **raise LeError((data[1],), "动作组运行次数错误")**
2531. **print(data[0])**
2532. **print(par)**
2533. **if not par is None:**
2534. **try:**
2535. **threading.Thread(target=LeArm.runActionGroup, args=(data[0], par)).start()**
2536. **except Exception, e:**
2537. **print(e)**
2538. **def cmd\_i004(sock, data=[]):**
2539. **actList = LeActList.listActions(actdir)**
2540. **actList.sort()**
2541. **if not len(actList) is 0:**
2542. **for i in range(0, len(actList), 10):**
2543. **str\_head = "I004-" + str(len(actList))**
2544. **str\_tial = "-" + str(i + 1) + "-"**
2545. **str\_tial1 = ""**
2546. **t = 10**
2547. **for j in range(0, 10, 1):**
2548. **if i + j < len(actList):**
2549. **str\_tial1 += "-" + actList[i + j][:-4]**
2550. **else:**
2551. **if t == 10:**
2552. **t = j**
2553. **if str\_tial1:**
2554. **str\_head = str\_head + str\_tial + str(i + t) + str\_tial1 + "\r\n"**
2555. **sock.sendall(str\_head)**
2556. **else:**
2557. **s = "I004-0-0-0\r\n"**
2558. **sock.sendall(s)**
2559. **print(len(actList))**
2560. **print(actList)**
2561. **def cmd\_i005(sock, data=[]):**
2562. **if data:**
2563. **for d in data:**
2564. **if d:**
2565. **os.remove(actdir + d + ".d6a")**
2566. **def cmd\_i006(sock, data=[]):**
2567. **actList = LeActList.listActions(actdir)**
2568. **for d in actList:**
2569. **os.remove(actdir + d)**
2570. **def cmd\_i007(sock, data=[]):**
2571. **try:**
2572. **time = int(data[0])**
2573. **servo\_num = int(data[1])**
2574. **servo\_data = []**
2575. **for i in range(servo\_num):**
2576. **servo\_id = int(data[2 + i \* 2])**
2577. **servo\_pos = int(data[3 + i \* 2])**
2578. **servo\_data.append((servo\_id, servo\_pos - 10000))**
2579. **except:**
2580. **raise LeError(tuple(data), "canshucuowu")**
2581. **try:**
2582. **for d in servo\_data:**
2583. **LeArm.setServo\_CMP(d[0], d[1], time)**
2584. **except Exception as e:**
2585. **print(cuowu, e)**
2586. **cmd\_list = [cmd\_i000, cmd\_i001, cmd\_i002, cmd\_i003, cmd\_i004, cmd\_i005, cmd\_i006, cmd\_i007]**
2587. **import thread**
2588. **import time**
2589. **import sqlite3 as sql**
2590. **import pigpio**
2591. **from LeServo import PWM\_Servo**
2592. **import os**
2593. **Servos = ()**
2594. **runningAction = False**
2595. **pi = None**
2596. **stopRunning = False**
2597. **def setServo(servoId, pos, time):**
2598. **global runningAction**
2599. **if servoId < 1 or servoId > 6:**
2600. **return**
2601. **if pos > 2500:**
2602. **pos = 2500**
2603. **elif pos < 500:**
2604. **pos = 500**
2605. **else:**
2606. **pass**
2607. **if time > 30000:**
2608. **time = 30000**
2609. **elif time < 20:**
2610. **time = 20**
2611. **else:**
2612. **pass**
2613. **if runningAction is False:**
2614. **Servos[servoId - 1].setPosition(pos, time)**
2615. **def setServo\_CMP(servoId, pos, time):**
2616. **if servoId < 1 or servoId > 6:**
2617. **return**
2618. **setServo(servoId, Servos[servoId - 1].getPosition() + pos, time);**
2619. **def setDeviation(servoId, d):**
2620. **global runningAction**
2621. **if servoId < 1 or servoId > 6:**
2622. **return**
2623. **if d < -300 or d > 300:**
2624. **return**
2625. **if runningAction is False:**
2626. **Servos[servoId - 1].setDeviation(d)**
2627. **def stopActionGroup():**
2628. **global stopRunning**
2629. **stopRunning = True**
2630. **def runActionGroup(actNum, times):**
2631. **global runningAction**
2632. **global stopRunning**
2633. **actNum = "./ActionGroups/" + actNum + ".d6a"**
2634. **if os.path.exists(actNum) is True:**
2635. **ag = sql.connect(actNum)**
2636. **cu = ag.cursor()**
2637. **cu.execute("select \* from ActionGroup")**
2638. **if runningAction is False:**
2639. **runningAction = True**
2640. **while True:**
2641. **if stopRunning is True:**
2642. **stopRunning = False**
2643. **runningAction = False**
2644. **cu.close()**
2645. **ag.close()**
2646. **break**
2647. **act = cu.fetchone()**
2648. **if act is not None:**
2649. **for i in range(0, 6, 1):**
2650. **Servos[i].setPosition(act[2 + i], act[1])**
2651. **time.sleep(float(act[1]) / 1000.0)**
2652. **else:**
2653. **runningAction = False**
2654. **cu.close()**
2655. **ag.close()**
2656. **break**
2657. **else:**
2658. **runningAction = False**
2659. **print("未能找到动作组文件")**
2660. **def initLeArm(d):**
2661. **global Servos**
2662. **global pi**
2663. **pi = pigpio.pi()**
2664. **servo1 = PWM\_Servo(pi, 12, deviation=d[0], control\_speed=True)**
2665. **servo2 = PWM\_Servo(pi, 16, deviation=d[1], control\_speed=True)**
2666. **servo3 = PWM\_Servo(pi, 20, deviation=d[2], control\_speed=True)**
2667. **servo4 = PWM\_Servo(pi, 21, deviation=d[3], control\_speed=True)**
2668. **servo5 = PWM\_Servo(pi, 19, deviation=d[4], control\_speed=True)**
2669. **servo6 = PWM\_Servo(pi, 13, deviation=d[5], control\_speed=True)**
2670. **Servos = (servo1, servo2, servo3, servo4, servo5, servo6)**
2671. **for i in range(0, 6, 1):**
2672. **Servos[i].setPosition(1500, 2000)**
2673. **def stopLeArm():**
2674. **print("停止机械臂")**
2675. **pi.stop()**
2676. **import os**
2677. **def listActions(path):**
2678. **if not os.path.exists(path):**
2679. **os.mkdir(path)**
2680. **pathlist = os.listdir(path)**
2681. **actList = []**
2682. **for f in pathlist:**
2683. **if f[0] == '.':**
2684. **pass**
2685. **else:**
2686. **if f[-4:] == '.d6a':**
2687. **f.replace('-', '')**
2688. **if f:**
2689. **actList.append(f)**
2690. **else:**
2691. **pass**
2692. **return actList**
2693. **from math import \***
2694. **import LeArm**
2695. **import time**
2696. **import matplotlib.pyplot as plt**
2697. **import LeConf**
2698. **d = []**
2699. **for i in range(0, len(LeConf.Deviation), 1):**
2700. **if LeConf.Deviation[i] > 1600 or LeConf.Deviation < 1400:**
2701. **print("偏差值超出范围1200-1800")**
2702. **else:**
2703. **d.append(LeConf.Deviation[i] - 1500)**
2704. **LeArm.initLeArm(tuple(d))**
2705. **targetX = 0**
2706. **targetY = 0**
2707. **targetZ = 4000**
2708. **lastX = 0**
2709. **lastY = 0**
2710. **lastZ = 4000**
2711. **l0 = 960**
2712. **l1 = 1050**
2713. **l2 = 880**
2714. **l3 = 1650**
2715. **alpha = 0**
2716. **def kinematic\_analysis(x, y, z, Alpha):**
2717. **global alpha**
2718. **global l0, l1, l2, l3**
2719. **if x == 0:**
2720. **theta6 = 90.0**
2721. **elif x < 0:**
2722. **theta6 = atan(y / x)**
2723. **theta6 = 180 + (theta6 \* 180.0 / pi)**
2724. **else:**
2725. **theta6 = atan(y / x)**
2726. **theta6 = theta6 \* 180.0 / pi**
2727. **y = sqrt(x \* x + y \* y)**
2728. **y = y - l3 \* cos(Alpha \* pi / 180.0)**
2729. **z = z - l0 - l3 \* sin(Alpha \* pi / 180.0)**
2730. **if z < -l0:**
2731. **return False**
2732. **if sqrt(y \* y + z \* z) > (l1 + l2):**
2733. **return False**
2734. **aaa = -(y \* y + z \* z - l1 \* l1 - l2 \* l2) / (2 \* l1 \* l2)**
2735. **if aaa > 1 or aaa < -1:**
2736. **return False**
2737. **theta4 = acos(aaa)**
2738. **theta4 = 180.0 - theta4 \* 180.0 / pi**
2739. **if theta4 > 135.0 or theta4 < -135.0:**
2740. **return False**
2741. **alpha = acos(y / sqrt(y \* y + z \* z))**
2742. **bbb = (y \* y + z \* z + l1 \* l1 - l2 \* l2) / (2 \* l1 \* sqrt(y \* y + z \* z))**
2743. **if bbb > 1 or bbb < -1:**
2744. **return False**
2745. **if z < 0:**
2746. **zf\_flag = -1**
2747. **else:**
2748. **zf\_flag = 1**
2749. **theta5 = alpha \* zf\_flag + acos(bbb)**
2750. **theta5 = theta5 \* 180.0 / pi**
2751. **if theta5 > 180.0 or theta5 < 0:**
2752. **return False**
2753. **theta3 = Alpha - theta5 + theta4**
2754. **if theta3 > 90.0 or theta3 < -90.0:**
2755. **return False**
2756. **return theta3, theta4, theta5, theta6**
2757. **def averagenum(num):**
2758. **nsum = 0**
2759. **for i in range(len(num)):**
2760. **nsum += num[i]**
2761. **return nsum / len(num)**
2762. **def ki\_move(x, y, z, movetime):**
2763. **alpha\_list = []**
2764. **for alp in range(-25, -65, -1):**
2765. **if kinematic\_analysis(x, y, z, alp):**
2766. **alpha\_list.append(alp)**
2767. **if len(alpha\_list) > 0:**
2768. **if y > 2150:**
2769. **best\_alpha = max(alpha\_list)**
2770. **else:**
2771. **best\_alpha = min(alpha\_list)**
2772. **theta3, theta4, theta5, theta6 = kinematic\_analysis(x, y, z, best\_alpha)**
2773. **pwm\_6 = int(2000.0 \* theta6 / 180.0 + 500.0)**
2774. **pwm\_5 = int(2000.0 \* (90.0 - theta5) / 180.0 + 1500.0)**
2775. **pwm\_4 = int(2000.0 \* (135.0 - theta4) / 270.0 + 500.0)**
2776. **pwm\_3 = int(2000.0 \* theta3 / 180.0 + 1500.0)**
2777. **LeArm.setServo(3, pwm\_3, movetime)**
2778. **LeArm.setServo(4, pwm\_4, movetime)**
2779. **LeArm.setServo(5, pwm\_5, movetime)**
2780. **LeArm.setServo(6, pwm\_6, movetime)**
2781. **time.sleep(movetime / 1000.0)**
2782. **return True**
2783. **else:**
2784. **return False**
2785. **def plt\_image(x\_list, y\_list):**
2786. **plt.title('Scatter Plot')**
2787. **plt.xlabel('x-value')**
2788. **plt.ylabel('y-label')**
2789. **plt.legend()**
2790. **plt.scatter(x\_list, y\_list, s=10, c="ff1212", marker='o')**
2791. **import cv2**
2792. **import numpy as np**
2793. **import time**
2794. **import urllib**
2795. **import threading**
2796. **import signal**
2797. **import LeArm**
2798. **import kinematics as kin**
2799. **import RPi.GPIO as GPIO**
2800. **stream = None**
2801. **bytes = ''**
2802. **orgFrame = None**
2803. **minFrame = None**
2804. **Running = False**
2805. **get\_image\_ok = False**
2806. **GPIO.setmode(GPIO.BCM)**
2807. **GPIO.setwarnings(False)**
2808. **key = 22**
2809. **GPIO.setup(key, GPIO.IN, GPIO.PUD\_UP)**
2810. **correction\_flag = False**
2811. **color\_dist = {'red': {'Lower': np.array([0, 60, 60]), 'Upper': np.array([6, 255, 255])},**
2812. **'blue': {'Lower': np.array([100, 80, 46]), 'Upper': np.array([124, 255, 255])},**
2813. **'green': {'Lower': np.array([35, 43, 35]), 'Upper': np.array([90, 255, 255])},**
2814. **}**
2815. **position\_color\_list = []**
2816. **cv\_blocks\_ok = False**
2817. **step = 0**
2818. **num\_random = None**
2819. **cv\_count = 0**
2820. **last\_blocks = []**
2821. **last\_x = 0**
2822. **stable = False**
2823. **storage\_blocks = []**
2824. **def cv\_stop(signum, frame):**
2825. **global Running**
2826. **print("Stop face detection")**
2827. **if Running is True:**
2828. **Running = False**
2829. **cv2.destroyWindow('face\_detection')**
2830. **cv2.destroyAllWindows()**
2831. **def cv\_continue(signum, frame):**
2832. **global stream**
2833. **global Running**
2834. **if Running is False:**
2835. **if stream:**
2836. **stream.close()**
2837. **stream = urllib.urlopen("http://127.0.0.1:8080/?action=stream?dummy=param.mjpg")**
2838. **bytes = ''**
2839. **Running = True**
2840. **signal.signal(signal.SIGTSTP, cv\_stop)**
2841. **signal.signal(signal.SIGCONT, cv\_continue)**
2842. **def get\_image():**
2843. **global Running**
2844. **global orgFrame, minFrame**
2845. **global bytes**
2846. **global get\_image\_ok**
2847. **while True:**
2848. **if Running:**
2849. **try:**
2850. **bytes += stream.read(2048)**
2851. **a = bytes.find('\xff\xd8')**
2852. **b = bytes.find('\xff\xd9')**
2853. **if a != -1 and b != -1:**
2854. **jpg = bytes[a:b + 2]**
2855. **bytes = bytes[b + 2:]**
2856. **orgFrame = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8), cv2.CV\_LOAD\_IMAGE\_COLOR)**
2857. **minFrame = cv2.resize(orgFrame, (320, 240), interpolation=cv2.INTER\_LINEAR)**
2858. **get\_image\_ok = True**
2859. **except Exception as e:**
2860. **print(e)**
2861. **continue**
2862. **else:**
2863. **time.sleep(0.01)**
2864. **th1 = threading.Thread(target=get\_image)**
2865. **th1.setDaemon(True)**
2866. **th1.start()**
2867. **def leMap(x, in\_min, in\_max, out\_min, out\_max):**
2868. **return (x - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min**
2869. **def move\_blocks():**
2870. **global cv\_blocks\_ok, position\_color\_list, step**
2871. **while True:**
2872. **if cv\_blocks\_ok is True:**
2873. **if len(position\_color\_list) == 1:**
2874. **print**
2875. **position\_color\_list, 'pos'**
2876. **if step == 0:**
2877. **x\_pix\_cm = position\_color\_list[0][1]**
2878. **y\_pix\_cm = position\_color\_list[0][2]**
2879. **angle = position\_color\_list[0][3]**
2880. **n\_x = int(leMap(x\_pix\_cm, 0.0, 320.0, -1250.0, 1250.0)) \* 1.0**
2881. **n\_y = int(leMap(240 - y\_pix\_cm, 0.0, 240.0, 1250, 3250.0)) \* 1.0**
2882. **if n\_x < -100:**
2883. **n\_x -= 120**
2884. **LeArm.setServo(1, 700, 500)**
2885. **time.sleep(0.5)**
2886. **step = 1**
2887. **elif step == 1:**
2888. **if kin.ki\_move(n\_x, n\_y, 200.0, 1500):**
2889. **step = 2**
2890. **else:**
2891. **step = 6**
2892. **elif step == 2:**
2893. **if angle <= -45:**
2894. **angle = -(90 + angle)**
2895. **n\_angle = leMap(angle, 0.0, -45.0, 1500.0, 1750.0)**
2896. **if n\_x > 0:**
2897. **LeArm.setServo(2, 3000 - n\_angle, 500)**
2898. **else:**
2899. **LeArm.setServo(2, n\_angle, 500)**
2900. **time.sleep(0.5)**
2901. **step = 3**
2902. **elif step == 3:**
2903. **print**
2904. **'3 ok'**
2905. **LeArm.setServo(1, 1200, 500)**
2906. **time.sleep(0.5)**
2907. **step = 4**
2908. **elif step == 4:**
2909. **print**
2910. **'4 ok'**
2911. **kin.ki\_move(n\_x, n\_y, 700.0, 1000)**
2912. **step = 5**
2913. **elif step == 5:**
2914. **print**
2915. **'5 ok'**
2916. **if position\_color\_list[0][0] == 'red':**
2917. **LeArm.runActionGroup('red', 1)**
2918. **elif position\_color\_list[0][0] == 'blue':**
2919. **LeArm.runActionGroup('blue', 1)**
2920. **elif position\_color\_list[0][0] == 'green':**
2921. **LeArm.runActionGroup('green', 1)**
2922. **step = 6**
2923. **elif step == 6:**
2924. **print**
2925. **'6 ok'**
2926. **LeArm.runActionGroup('rest', 1)**
2927. **threadLock.acquire()**
2928. **position\_color\_list = []**
2929. **cv\_blocks\_ok = False**
2930. **threadLock.release()**
2931. **step = 0**
2932. **else:**
2933. **time.sleep(0.01)**
2934. **th2 = threading.Thread(target=move\_blocks)**
2935. **th2.setDaemon(True)**
2936. **th2.start()**
2937. **threadLock = threading.Lock()**
2938. **lens\_mtx = np.array([**
2939. **[993.17745922, 0., 347.76412756],**
2940. **[0., 992.6210587, 198.08924031],**
2941. **[0., 0., 1.],**
2942. **])**
2943. **lens\_dist = np.array([[-2.22696961e-01, 3.34897836e-01, 1.43573965e-03, -5.99140365e-03, -2.03168813e+00]])**
2944. **def lens\_distortion\_adjustment(image):**
2945. **global lens\_mtx, lens\_dist**
2946. **h, w = image.shape[:2]**
2947. **newcameramtx, roi = cv2.getOptimalNewCameraMatrix(lens\_mtx, lens\_dist, (w, h), 0, (w, h))**
2948. **dst = cv2.undistort(image, lens\_mtx, lens\_dist, None, newcameramtx)**
2949. **return dst**
2950. **def Arm\_Pos\_Corr():**
2951. **LeArm.setServo(1, 1200, 500)**
2952. **time.sleep(0.5)**
2953. **kin.ki\_move(0, 2250, 200.0, 1500)**
2954. **run\_corr\_one = 0**
2955. **cv\_continue(1, 1)**
2956. **LeArm.runActionGroup('rest', 1)**
2957. **while True:**
2958. **if GPIO.input(key) == 0:**
2959. **time.sleep(0.1)**
2960. **if GPIO.input(key) == 0:**
2961. **correction\_flag = not correction\_flag**
2962. **if correction\_flag is False:**
2963. **LeArm.runActionGroup('rest', 1)**
2964. **if correction\_flag is False:**
2965. **run\_corr\_one = 0**
2966. **if minFrame is not None and get\_image\_ok:**
2967. **t1 = cv2.getTickCount()**
2968. **frame = lens\_distortion\_adjustment(minFrame)**
2969. **img\_h, img\_w = frame.shape[:2]**
2970. **img\_center\_x = img\_w / 2**
2971. **img\_center\_y = img\_h / 2**
2972. **if cv\_blocks\_ok is False:**
2973. **gs\_frame = cv2.GaussianBlur(frame, (5, 5), 0)**
2974. **hsv = cv2.cvtColor(gs\_frame, cv2.COLOR\_BGR2HSV)**
2975. **for i in color\_dist:**
2976. **mask = cv2.inRange(hsv, color\_dist[i]['Lower'], color\_dist[i]['Upper'])**
2977. **mask = cv2.erode(mask, None, iterations=2)**
2978. **kernel = np.ones((5, 5), np.uint8)**
2979. **mask = cv2.dilate(mask, kernel, iterations=2)**
2980. **cnts = cv2.findContours(mask.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[-2]**
2981. **if len(cnts) > 0:**
2982. **c = max(cnts, key=cv2.contourArea)**
2983. **rect = cv2.minAreaRect(c)**
2984. **box = cv2.cv.BoxPoints(rect)**
2985. **cv2.drawContours(frame, [np.int0(box)], -1, (0, 255, 255), 2)**
2986. **c\_x, c\_y = rect[0]**
2987. **h, w = rect[1]**
2988. **c\_angle = rect[2]**
2989. **if h \* w >= 1350:**
2990. **cv2.circle(frame, (int(c\_x), int(c\_y)), 3, (216, 0, 255), -1)**
2991. **last\_blocks.append([int(c\_x), i])**
2992. **if stable:**
2993. **storage\_blocks.append((int(c\_y), int(c\_x), i, int(c\_angle)))**
2994. **stable = False**
2995. **if len(last\_blocks) > 0:**
2996. **if -10 <= int(last\_blocks[len(last\_blocks) - 1][0] - last\_x) <= 10:**
2997. **print**
2998. **cv\_count**
2999. **cv\_count += 1**
3000. **else:**
3001. **cv\_count = 0**
3002. **last\_x = int(last\_blocks[len(last\_blocks) - 1][0])**
3003. **last\_blocks = []**
3004. **if cv\_count >= 5:**
3005. **cv\_count = 0**
3006. **stable = True**
3007. **if len(storage\_blocks) > 0:**
3008. **max\_y = storage\_blocks.index(max(storage\_blocks))**
3009. **position\_color\_list.append((storage\_blocks[max\_y][2], storage\_blocks[max\_y][1],**
3010. **storage\_blocks[max\_y][0], storage\_blocks[max\_y][3]))**
3011. **storage\_blocks = []**
3012. **cv\_blocks\_ok = True**
3013. **cv2.line(frame, ((img\_w / 2) - 20, (img\_h / 2)), ((img\_w / 2) + 20, (img\_h / 2)), (0, 0, 255), 1)**
3014. **cv2.line(frame, ((img\_w / 2), (img\_h / 2) - 20), ((img\_w / 2), (img\_h / 2) + 20), (0, 0, 255), 1)**
3015. **cv2.imshow('image', frame)**
3016. **cv2.waitKey(1)**
3017. **get\_image\_ok = False**
3018. **t2 = cv2.getTickCount()**
3019. **time\_r = (t2 - t1) / cv2.getTickFrequency() \* 1000**
3020. **else:**
3021. **time.sleep(0.01)**
3022. **else:**
3023. **if correction\_flag and run\_corr\_one == 0:**
3024. **run\_corr\_one += 1**
3025. **Arm\_Pos\_Corr()**
3026. **else:**
3027. **frame = lens\_distortion\_adjustment(minFrame)**
3028. **img\_h, img\_w = frame.shape[:2]**
3029. **img\_center\_x = img\_w / 2**
3030. **img\_center\_y = img\_h / 2**
3031. **cv2.line(frame, ((img\_w / 2) - 20, (img\_h / 2)), ((img\_w / 2) + 20, (img\_h / 2)), (0, 0, 255), 1)**
3032. **cv2.line(frame, ((img\_w / 2), (img\_h / 2) - 20), ((img\_w / 2), (img\_h / 2) + 20), (0, 0, 255), 1)**
3033. **cv2.imshow('image', frame)**
3034. **cv2.waitKey(1)**
3035. **time.sleep(0.01)**