

# Detecção de objetos em vídeo

## Introdução

Este trabalho foi desenvolvido por Igor Carvalho de Brito Batista, Rony de Sena Lourenço e Thatiana Jéssica da Silva Ribeiro.

O objetivo se trata de reconhecer placas de trânsito em vídeo. As principais razões para a escolha desse objeto foi de fazer uma detecção de algo que faz parte da vida cotidiana do ser humano e que houvesse em grande quantidade. Assim, seria possível demonstrar a tecnologia para detecção de objetos em imagens.

## Metodologia

Para a realização deste trabalho foi utilizado uma ferramenta chamada Yolo, que utiliza uma rede neural profunda, cuja a arquitetura é chamada de *darknet*, com 19 camadas convolucionais e 5 camadas de *maxpooling*. Uma rede neural convolucional é um dos conceitos mais importante no ramo inteligência artificial e aprendizagem de máquina. Seu uso está, normalmente, sendo utilizada para classificação de imagens. Uma CNN pode ser dividida em duas partes: extração de características e aplicação de uma rede neural.

Para realizar a extração de características é necessário que a imagem passe por 4 etapas: *convolução*, *relu*, *pooling* e *flattening*.

- **Convolução**

A convolução, dentro do contexto de imagens, é um processo que transforma uma imagem. Para realizar uma convolução é necessário dois elementos: uma imagem de entrada e o detector de características que pode ser chamado de filtro ou kernel, a operação realizada entre ambas vai gerar um mapa de características, que vai ressaltar as características que o filtro possui. Esse filtro é uma matriz utilizada para realizar uma operação em várias regiões da imagem, os mais comuns são: *sharpen*, *blur* e *edge detect*.

- **ReLU**

Dependendo do detector de características aplicado, existe a possibilidade de valores negativos serem gerados, para adicionar a não linearidade a rede, deixando apenas os valores positivos é necessário a aplicação de uma funções de ativação.

O mais comum, e principalmente no contexto de imagens, a mais utilizada é a função ReLU.

- **Pooling**

Pooling é um processo de redução do mapa de características. O objetivo é reduzir a imagem, mas conservar o máximo de características da imagem original. Um dos principais motivos dessa operação no modelo, é de diminuir sua variância a pequenas alterações e também de reduzir a quantidade de parâmetros treinados pela rede.

Existem diversas formas para realizar esse processo, mas a forma mais utilizada dentro do campo de inteligência artificial é a *MaxPooling*, em que é retirado o maior elemento do mapa, assim, formando uma nova matriz.

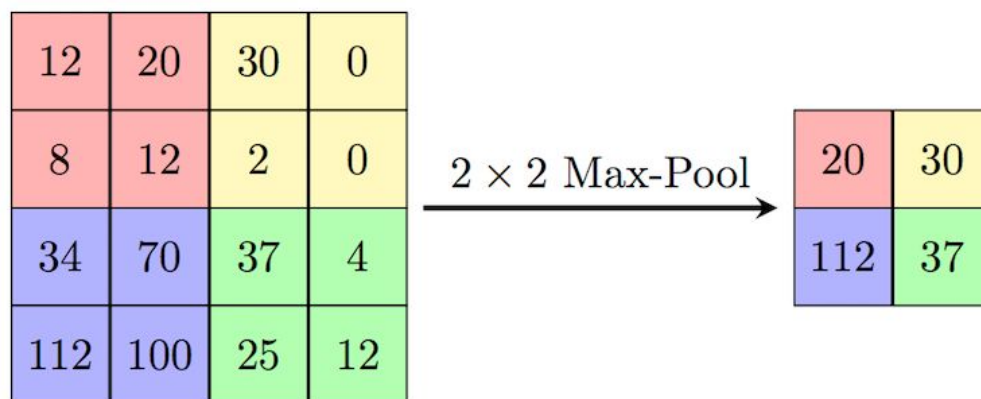


Imagem 1 - Processo de pooling

- **Flattening**

Essa operação consiste em transformar a matriz da imagem para um *array*. Assim, é possível inserir esses dados na entrada de uma rede neural.

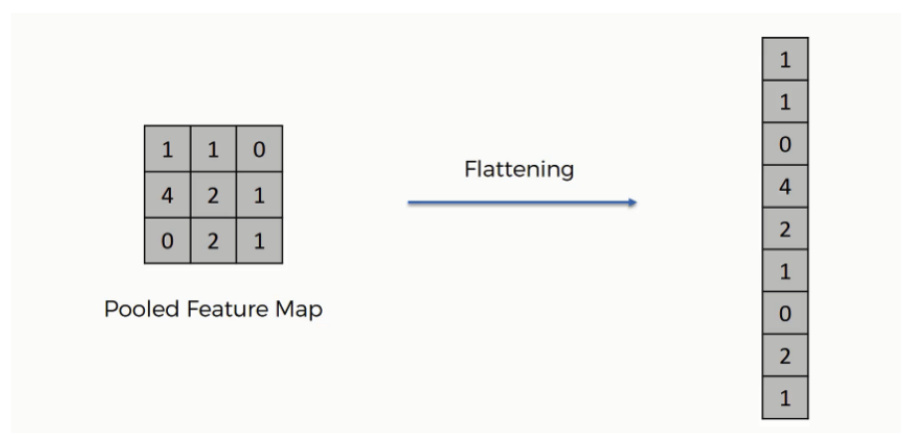


Imagem 2 - Processo de flattening

Para utilizar a ferramenta Yolo, foi necessário a instalação da linguagem de programação C e da biblioteca OpenCV. Todo o processamento do projeto foi feito em uma GPU Nvidia Geforce GTX750Ti com somente 2 gigabytes de memória. Foi necessário utilizar um software chamado Yolo Annotation Tool-New para realizar a marcação das placas e salvar as coordenadas em um arquivo .txt.

No que diz respeito ao sistema operacional, o Mint 19 Cinnamon foi a escolha por ser bastante popular e ter um desempenho notável.

A detecção de objetos no YoLo se dá em função da imagem escolhida e o peso gerado durante o processo de treinamento. Para este trabalho foi escolhido um pequeno conjunto de imagens estáticas. A real aplicação do projeto é destinada à detecção de objetos em vídeos, sendo o foco na análise dos resultados. Ainda a respeito dos vídeos, o YoLo é capaz de verificar quadro a quadro a presença de padrões que caracterizam os objetos relacionados com o peso previamente gerado no treinamento. Ou seja, a cada quadro são verificadas semelhanças e as mesmas são representadas por meio de porcentagens.

## Conjunto de dados

O dataset de imagens foi elaborado a partir de placas das ruas da cidade de Natal/RN. Para isso, foi necessário gravar vídeos em um carro em movimento. Como foi preciso de fotos para poder realizar o treinamento da rede neural, então, alguns dos vídeos foram separados para a extração de frames, durante esse processo, foi necessário buscar por imagens com ângulos variados, com formatos diversificados, com diferentes condições de luminosidade e com cenários distintos. Também foram escolhidas imagens com resoluções diferenciadas, a fim de obter uma gama diversa de imagens e aumentar a precisão na identificação dos objetos pós treinamento.

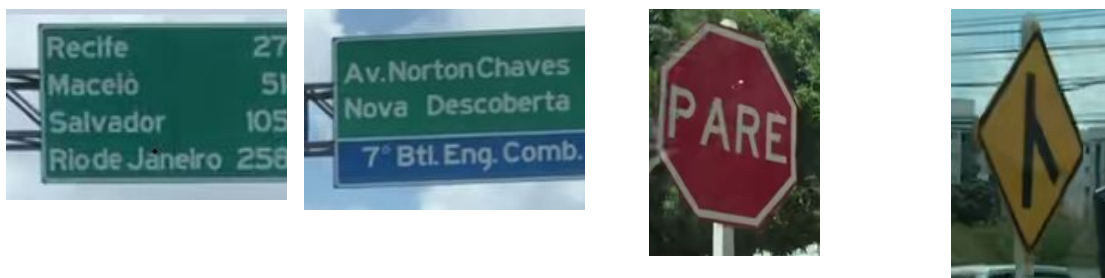


Imagem 3 - Amostras de imagens da base de dados

Após a separação de todo o conjunto de dados, foi necessário a marcação de todas as imagens utilizando o Yolo Annotation Tool-New, esse software gera um arquivo no formato .txt. Todas as imagens e as respectivas coordenadas dos

objetos foram colocadas em uma pasta para que fossem acessíveis no momento do treinamento. Na Imagem 4, é possível observar o processo de marcação.

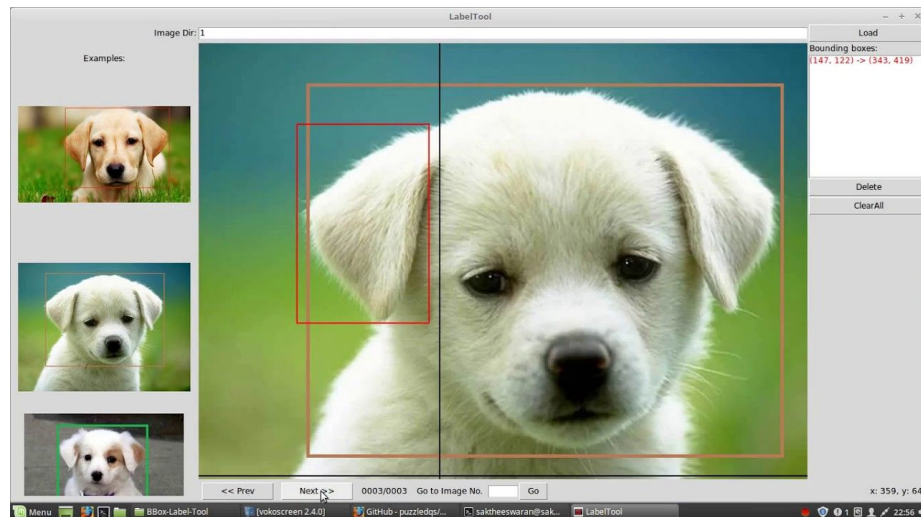


Imagem 4 - Processo de anotação de imagem

O treinamento do conjunto de dados é feito em loop até que seja dado o comando de parada caso se obtenha uma acurácia satisfatória. Embora o treinamento seja feito em loop, o Yolo permite que seja salvo automaticamente um backup dos pesos que serão utilizados na detecção das imagens. Existe uma grande vantagem nesse sistema de backup de pesos que é a segurança em caso de eventuais panes ou desligamentos que possam vir a ocorrer.

## Códigos

Os códigos utilizados neste trabalho são referentes ao Yolo3 contido no repositório do github <https://github.com/pjreddie/darknet> . O processo de treinamento da rede é feito utilizando-se a linha de comando mostrada na Imagem 5.

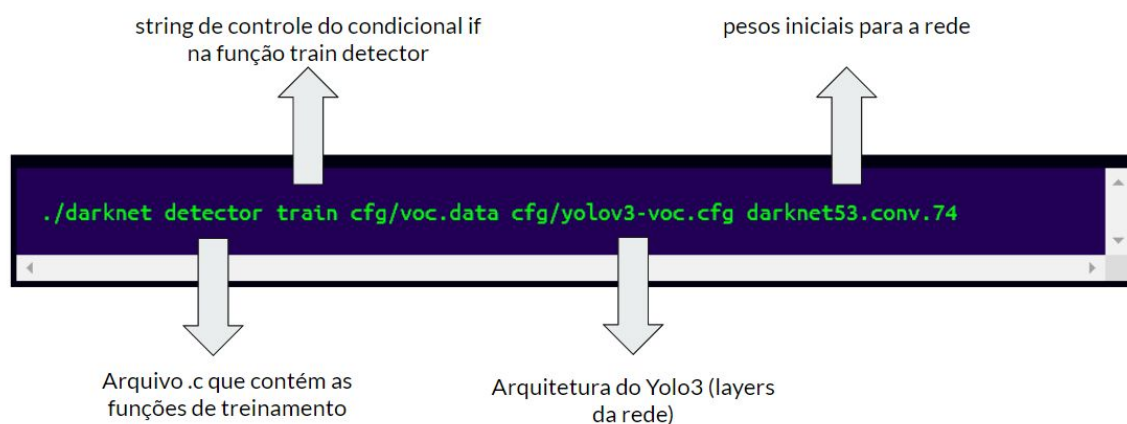
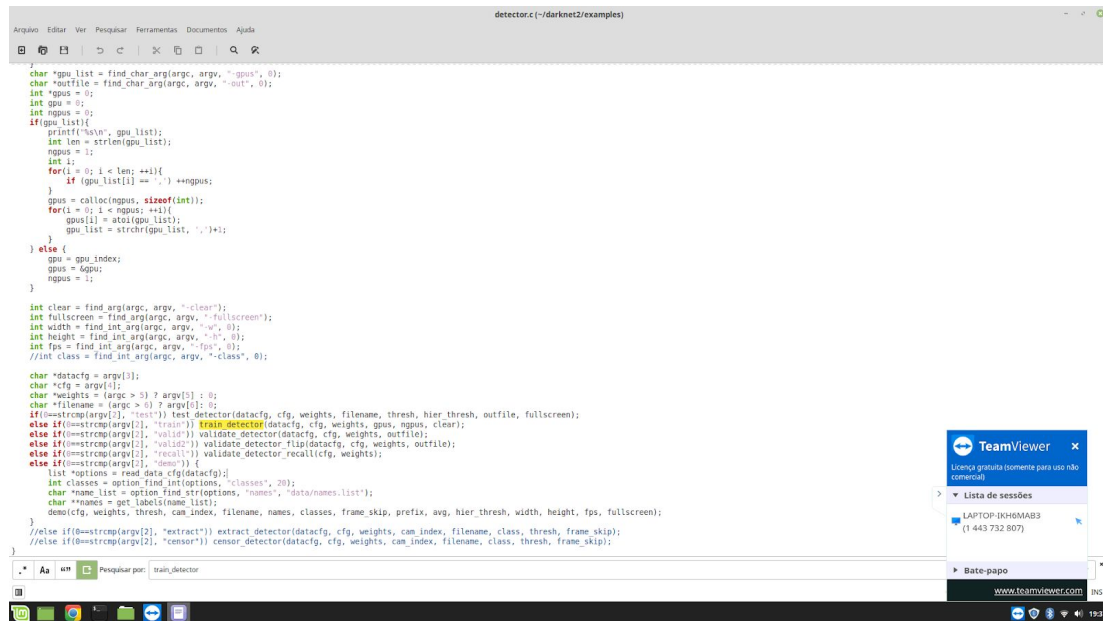


Imagem 5 - Comando para treinamento da rede

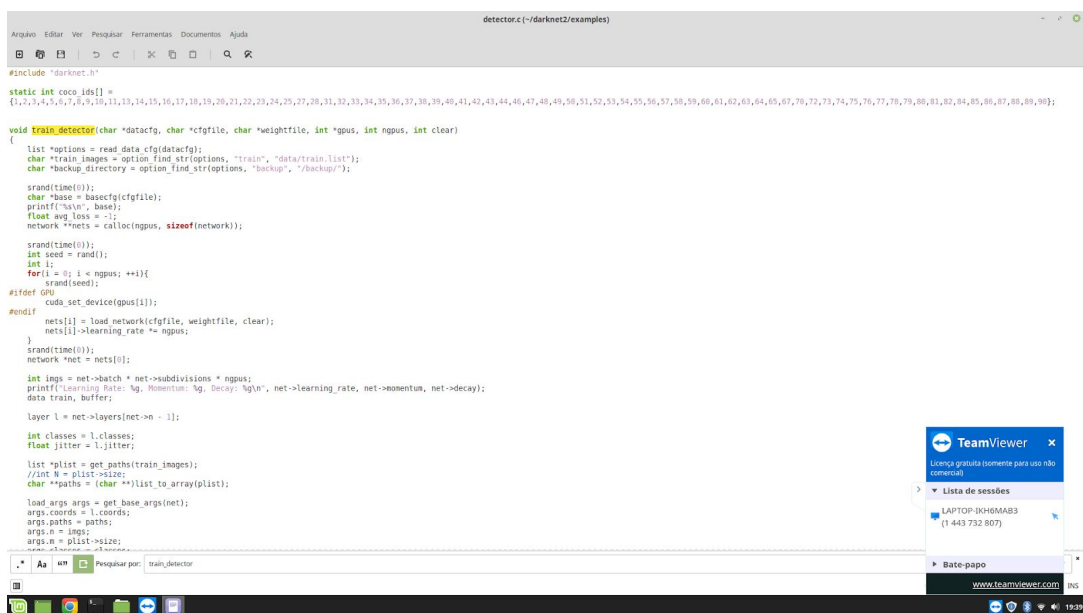
O primeiro termo refere-se ao framework utilizado, o darknet, que é um framework de rede neural open source escrito em C e CUDA. O Detector é o arquivo principal para o treinamento, escrito em C. Dentro dele há diversas funções, que, dependendo da string passada como terceiro parâmetro, são realizadas. Para o caso do treinamento, a string “train” será responsável pela invocação da função de treinamento, mostrada em detalhes na Imagem 6 e 7. Além disso, o quarto parâmetro faz referência a arquitetura do Yolo3 e o último parâmetro refere-se aos pesos usados inicialmente para a rede.



```
1
2
3 char *gpu_list = find_char_arg(argc, argv, "-gpus", 0);
4 char *outfile = find_char_arg(argc, argv, "-out", 0);
5 int ngpus = 0;
6 int gpu = 0;
7 if(gpu_list){
8     printf("%s\n", gpu_list);
9     int len = strlen(gpu_list);
10    ngpus = 1;
11    for(i = 0; i < len; ++i){
12        if (gpu_list[i] == ',') ++ngpus;
13    }
14    gpu = calloc(ngpus, sizeof(int));
15    for(i = 0; i < ngpus; ++i){
16        gpu[i] = atoi(gpu_list);
17        gpu_list = strchr(gpu_list, ',')+1;
18    }
19 } else {
20     gpu = gpu_index;
21     ngpus = 1;
22 }
23
24 int clear = find_arg(argc, argv, "-clear");
25 int fullscreen = find_arg(argc, argv, "-fullscreen");
26 int width = find_int_arg(argc, argv, "-w", 0);
27 int height = find_int_arg(argc, argv, "-h", 0);
28 int fps = find_int_arg(argc, argv, "-fps", 0);
29 //int class = find_int_arg(argc, argv, "-class", 0);
30
31 char *datacfg = argv[3];
32 char *cfg = argv[4];
33 char *weights = (argc > 5) ? argv[5] : 0;
34 char *filename = (argc > 6) ? argv[6] : 0;
35 if(!strcmp(argv[2], "test")) test_detector(datacfg, cfg, weights, filename, thresh, hier_thresh, outfile, fullscreen);
36 else if(!strcmp(argv[2], "train")) train_detector(datacfg, cfg, weights, ngpus, clear);
37 else if(!strcmp(argv[2], "val")) validate_detector(datacfg, cfg, weights, outfile);
38 else if(!strcmp(argv[2], "recall")) validate_detector_recall(cfg, weights, outfile);
39 else if(!strcmp(argv[2], "demo")) {
40     list *options = read_data_cfg(datacfg);
41     int classes = option_find_int(options, "classes", 20);
42     char *name_list = option_find_str(options, "names", "data/names.list");
43     char **names = get_labels(name_list);
44     demo(cfg, weights, thresh, cam_index, filename, names, classes, frame_skip, prefix, avg, hier_thresh, width, height, fps, fullscreen);
45 }
46 //else if(!strcmp(argv[2], "extract")) extract_detector(datacfg, cfg, weights, cam_index, filename, class, thresh, frame_skip);
47 //else if(!strcmp(argv[2], "censor")) censor_detector(datacfg, cfg, weights, cam_index, filename, class, thresh, frame_skip);
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

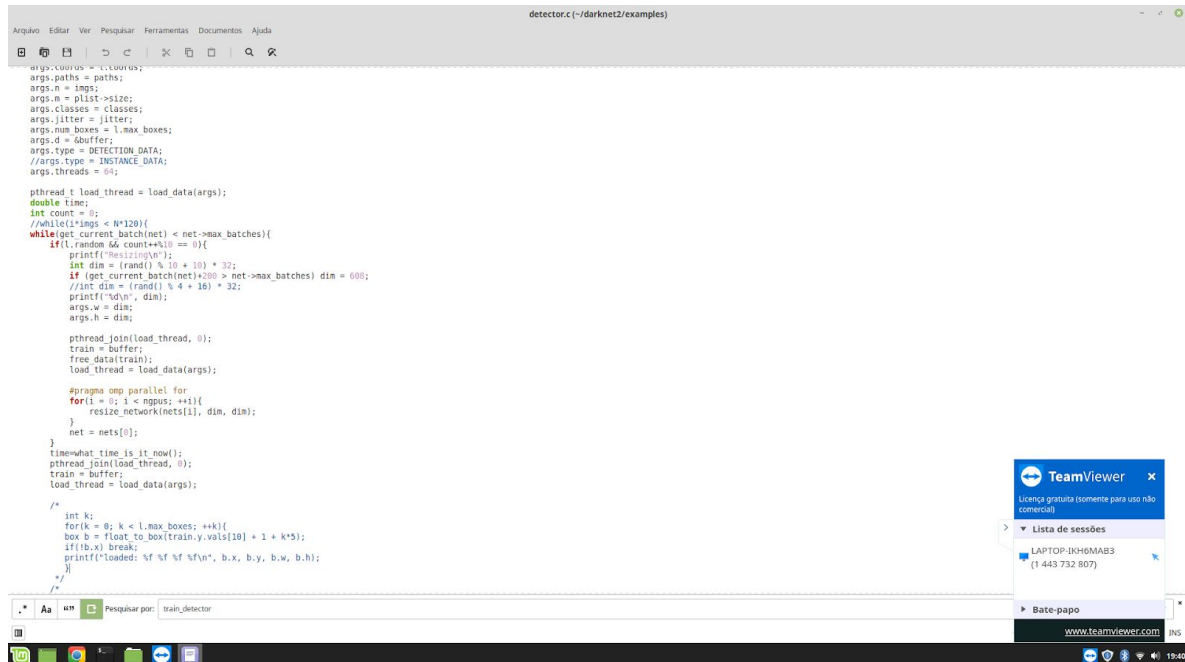
Imagem 6 - Detalhes do arquivo detector.c

Após entrada no “if” referente a string train, passada como parâmetro na linha de comando, a função “train detector”, mostrada na Imagem 7, é chamada.



```
1
2
3 #include "darknet.h"
4
5 static int coco_ids[] =
6 {1,2,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,27,28,31,32,33,34,35,36,37,38,39,40,41,42,43,44,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,67,70,72,73,74,75,76,77,78,79,80,81,82,84,85,86,87,88,89,90};
7
8
9 void train_detector(char *datacfg, char *cfgfile, char *weightfile, int *ngpus, int ngpus, int clear)
10 {
11     list *options = read_data_cfg(datacfg);
12     char *train_images = option_find_str(options, "train", "data/train.list");
13     char *backup_directory = option_find_str(options, "backup", "/backup/");
14
15     srand(time(0));
16     char *base = basecfg(cfgfile);
17     printf("%s\n", base);
18     float avg_loss = -1;
19     network **nets = calloc(ngpus, sizeof(network));
20
21     srand(time(0));
22     int seed = rand();
23     int i;
24     for(i = 0; i < ngpus; ++i){
25         srand(seed);
26     }
27 #ifdef GPU
28     cuda_set_device(gpus[i]);
29 #endif
30     nets[i] = load_network(cfgfile, weightfile, clear);
31     nets[i]->learning_rate *= ngpus;
32
33     srand(time(0));
34     network *net = nets[0];
35
36     int inps = net->batch * net->subdivisions * ngpus;
37     printf("Learning Rate: %g, Momentum: %g, Decay: %g\n", net->learning_rate, net->momentum, net->decay);
38     data_train, buffer;
39
40     layer l = net->layers[net->n - 1];
41
42     int classes = l.classes;
43     float jitter = l.jitter;
44
45     list *plist = get_paths(train_images);
46     //int N = plist->size;
47     char **paths = (char **)list_to_array(plist);
48
49     load_args args = get_base_args(net);
50     args.coords = l.coords;
51     args.paths = paths;
52     args.n = inps;
53     args.m = plist->size;
54     //args.batch = net->batch;
55
56     //args.batch = net->batch;
57
58     //args.batch = net->batch;
59
60     //args.batch = net->batch;
61
62     //args.batch = net->batch;
63
64     //args.batch = net->batch;
65
66     //args.batch = net->batch;
67
68     //args.batch = net->batch;
69
70     //args.batch = net->batch;
71
72     //args.batch = net->batch;
73
74     //args.batch = net->batch;
75
76     //args.batch = net->batch;
77
78     //args.batch = net->batch;
79
80     //args.batch = net->batch;
81
82     //args.batch = net->batch;
83
84     //args.batch = net->batch;
85
86     //args.batch = net->batch;
87
88     //args.batch = net->batch;
89
90     //args.batch = net->batch;
91
92     //args.batch = net->batch;
93
94     //args.batch = net->batch;
95
96     //args.batch = net->batch;
97
98     //args.batch = net->batch;
99
100     //args.batch = net->batch;
101
102     //args.batch = net->batch;
103
104     //args.batch = net->batch;
105
106     //args.batch = net->batch;
107
108     //args.batch = net->batch;
109
110     //args.batch = net->batch;
111
112     //args.batch = net->batch;
113
114     //args.batch = net->batch;
115
116     //args.batch = net->batch;
117
118     //args.batch = net->batch;
119
120     //args.batch = net->batch;
121
122     //args.batch = net->batch;
123
124     //args.batch = net->batch;
125
126     //args.batch = net->batch;
127
128     //args.batch = net->batch;
129
130     //args.batch = net->batch;
131
132     //args.batch = net->batch;
133
134     //args.batch = net->batch;
135
136     //args.batch = net->batch;
137
138     //args.batch = net->batch;
139
140     //args.batch = net->batch;
141
142     //args.batch = net->batch;
143
144     //args.batch = net->batch;
145
146     //args.batch = net->batch;
147
148     //args.batch = net->batch;
149
150     //args.batch = net->batch;
151
152     //args.batch = net->batch;
153
154     //args.batch = net->batch;
155
156     //args.batch = net->batch;
157
158     //args.batch = net->batch;
159
160     //args.batch = net->batch;
161
162     //args.batch = net->batch;
163
164     //args.batch = net->batch;
165
166     //args.batch = net->batch;
167
168     //args.batch = net->batch;
169
170     //args.batch = net->batch;
171
172     //args.batch = net->batch;
173
174     //args.batch = net->batch;
175
176     //args.batch = net->batch;
177
178     //args.batch = net->batch;
179
180     //args.batch = net->batch;
181
182     //args.batch = net->batch;
183
184     //args.batch = net->batch;
185
186     //args.batch = net->batch;
187
188     //args.batch = net->batch;
189
190     //args.batch = net->batch;
191
192     //args.batch = net->batch;
193
194     //args.batch = net->batch;
195
196     //args.batch = net->batch;
197
198     //args.batch = net->batch;
199
200     //args.batch = net->batch;
201
202     //args.batch = net->batch;
203
204     //args.batch = net->batch;
205
206     //args.batch = net->batch;
207
208     //args.batch = net->batch;
209
210     //args.batch = net->batch;
211
212     //args.batch = net->batch;
213
214     //args.batch = net->batch;
215
216     //args.batch = net->batch;
217
218     //args.batch = net->batch;
219
220     //args.batch = net->batch;
221
222     //args.batch = net->batch;
223
224     //args.batch = net->batch;
225
226     //args.batch = net->batch;
227
228     //args.batch = net->batch;
229
230     //args.batch = net->batch;
231
232     //args.batch = net->batch;
233
234     //args.batch = net->batch;
235
236     //args.batch = net->batch;
237
238     //args.batch = net->batch;
239
240     //args.batch = net->batch;
241
242     //args.batch = net->batch;
243
244     //args.batch = net->batch;
245
246     //args.batch = net->batch;
247
248     //args.batch = net->batch;
249
250     //args.batch = net->batch;
251
252     //args.batch = net->batch;
253
254     //args.batch = net->batch;
255
256     //args.batch = net->batch;
257
258     //args.batch = net->batch;
259
260     //args.batch = net->batch;
261
262     //args.batch = net->batch;
263
264     //args.batch = net->batch;
265
266     //args.batch = net->batch;
267
268     //args.batch = net->batch;
269
270     //args.batch = net->batch;
271
272     //args.batch = net->batch;
273
274     //args.batch = net->batch;
275
276     //args.batch = net->batch;
277
278     //args.batch = net->batch;
279
280     //args.batch = net->batch;
281
282     //args.batch = net->batch;
283
284     //args.batch = net->batch;
285
286     //args.batch = net->batch;
287
288     //args.batch = net->batch;
289
290     //args.batch = net->batch;
291
292     //args.batch = net->batch;
293
294     //args.batch = net->batch;
295
296     //args.batch = net->batch;
297
298     //args.batch = net->batch;
299
300     //args.batch = net->batch;
301
302     //args.batch = net->batch;
303
304     //args.batch = net->batch;
305
306     //args.batch = net->batch;
307
308     //args.batch = net->batch;
309
310     //args.batch = net->batch;
311
312     //args.batch = net->batch;
313
314     //args.batch = net->batch;
315
316     //args.batch = net->batch;
317
318     //args.batch = net->batch;
319
320     //args.batch = net->batch;
321
322     //args.batch = net->batch;
323
324     //args.batch = net->batch;
325
326     //args.batch = net->batch;
327
328     //args.batch = net->batch;
329
330     //args.batch = net->batch;
331
332     //args.batch = net->batch;
333
334     //args.batch = net->batch;
335
336     //args.batch = net->batch;
337
338     //args.batch = net->batch;
339
340     //args.batch = net->batch;
341
342     //args.batch = net->batch;
343
344     //args.batch = net->batch;
345
346     //args.batch = net->batch;
347
348     //args.batch = net->batch;
349
350     //args.batch = net->batch;
351
352     //args.batch = net->batch;
353
354     //args.batch = net->batch;
355
356     //args.batch = net->batch;
357
358     //args.batch = net->batch;
359
360     //args.batch = net->batch;
361
362     //args.batch = net->batch;
363
364     //args.batch = net->batch;
365
366     //args.batch = net->batch;
367
368     //args.batch = net->batch;
369
370     //args.batch = net->batch;
371
372     //args.batch = net->batch;
373
374     //args.batch = net->batch;
375
376     //args.batch = net->batch;
377
378     //args.batch = net->batch;
379
380     //args.batch = net->batch;
381
382     //args.batch = net->batch;
383
384     //args.batch = net->batch;
385
386     //args.batch = net->batch;
387
388     //args.batch = net->batch;
389
390     //args.batch = net->batch;
391
392     //args.batch = net->batch;
393
394     //args.batch = net->batch;
395
396     //args.batch = net->batch;
397
398     //args.batch = net->batch;
399
400     //args.batch = net->batch;
401
402     //args.batch = net->batch;
403
404     //args.batch = net->batch;
405
406     //args.batch = net->batch;
407
408     //args.batch = net->batch;
409
410     //args.batch = net->batch;
411
412     //args.batch = net->batch;
413
414     //args.batch = net->batch;
415
416     //args.batch = net->batch;
417
418     //args.batch = net->batch;
419
420     //args.batch = net->batch;
421
422     //args.batch = net->batch;
423
424     //args.batch = net->batch;
425
426     //args.batch = net->batch;
427
428     //args.batch = net->batch;
429
430     //args.batch = net->batch;
431
432     //args.batch = net->batch;
433
434     //args.batch = net->batch;
435
436     //args.batch = net->batch;
437
438     //args.batch = net->batch;
439
440     //args.batch = net->batch;
441
442     //args.batch = net->batch;
443
444     //args.batch = net->batch;
445
446     //args.batch = net->batch;
447
448     //args.batch = net->batch;
449
450     //args.batch = net->batch;
451
452     //args.batch = net->batch;
453
454     //args.batch = net->batch;
455
456     //args.batch = net->batch;
457
458     //args.batch = net->batch;
459
460     //args.batch = net->batch;
461
462     //args.batch = net->batch;
463
464     //args.batch = net->batch;
465
466     //args.batch = net->batch;
467
468     //args.batch = net->batch;
469
470     //args.batch = net->batch;
471
472     //args.batch = net->batch;
473
474     //args.batch = net->batch;
475
476     //args.batch = net->batch;
477
478     //args.batch = net->batch;
479
480     //args.batch = net->batch;
481
482     //args.batch = net->batch;
483
484     //args.batch = net->batch;
485
486     //args.batch = net->batch;
487
488     //args.batch = net->batch;
489
490     //args.batch = net->batch;
491
492     //args.batch = net->batch;
493
494     //args.batch = net->batch;
495
496     //args.batch = net->batch;
497
498     //args.batch = net->batch;
499
500     //args.batch = net->batch;
501
502     //args.batch = net->batch;
503
504     //args.batch = net->batch;
505
506     //args.batch = net->batch;
507
508     //args.batch = net->batch;
509
510     //args.batch = net->batch;
511
512     //args.batch = net->batch;
513
514     //args.batch = net->batch;
515
516     //args.batch = net->batch;
517
518     //args.batch = net->batch;
519
520     //args.batch = net->batch;
521
522     //args.batch = net->batch;
523
524     //args.batch = net->batch;
525
526     //args.batch = net->batch;
527
528     //args.batch = net->batch;
529
530     //args.batch = net->batch;
531
532     //args.batch = net->batch;
533
534     //args.batch = net->batch;
535
536     //args.batch = net->batch;
537
538     //args.batch = net->batch;
539
540     //args.batch = net->batch;
541
542     //args.batch = net->batch;
543
544     //args.batch = net->batch;
545
546     //args.batch = net->batch;
547
548     //args.batch = net->batch;
549
550     //args.batch = net->batch;
551
552     //args.batch = net->batch;
553
554     //args.batch = net->batch;
555
556     //args.batch = net->batch;
557
558     //args.batch = net->batch;
559
560     //args.batch = net->batch;
561
562     //args.batch = net->batch;
563
564     //args.batch = net->batch;
565
566     //args.batch = net->batch;
567
568     //args.batch = net->batch;
569
570     //args.batch = net->batch;
571
572     //args.batch = net->batch;
573
574     //args.batch = net->batch;
575
576     //args.batch = net->batch;
577
578     //args.batch = net->batch;
579
580     //args.batch = net->batch;
581
582     //args.batch = net->batch;
583
584     //args.batch = net->batch;
585
586     //args.batch = net->batch;
587
588     //args.batch = net->batch;
589
590     //args.batch = net->batch;
591
592     //args.batch = net->batch;
593
594     //args.batch = net->batch;
595
596     //args.batch = net->batch;
597
598     //args.batch = net->batch;
599
600     //args.batch = net->batch;
601
602     //args.batch = net->batch;
603
604     //args.batch = net->batch;
605
606     //args.batch = net->batch;
607
608     //args.batch = net->batch;
609
610     //args.batch = net->batch;
611
612     //args.batch = net->batch;
613
614     //args.batch = net->batch;
615
616     //args.batch = net->batch;
617
618     //args.batch = net->batch;
619
620     //args.batch = net->batch;
621
622     //args.batch = net->batch;
623
624     //args.batch = net->batch;
625
626     //args.batch = net->batch;
627
628     //args.batch = net->batch;
629
630     //args.batch = net->batch;
631
632     //args.batch = net->batch;
633
634     //args.batch = net->batch;
635
636     //args.batch = net->batch;
637
638     //args.batch = net->batch;
639
640     //args.batch = net->batch;
641
642     //args.batch = net->batch;
643
644     //args.batch = net->batch;
645
646     //args.batch = net->batch;
647
648     //args.batch = net->batch;
649
650     //args.batch = net->batch;
651
652     //args.batch = net->batch;
653
654     //args.batch = net->batch;
655
656     //args.batch = net->batch;
657
658     //args.batch = net->batch;
659
660     //args.batch = net->batch;
661
662     //args.batch = net->batch;
663
664     //args.batch = net->batch;
665
666     //args.batch = net->batch;
667
668     //args.batch = net->batch;
669
670     //args.batch = net->batch;
671
672     //args.batch = net->batch;
673
674     //args.batch = net->batch;
675
676     //args.batch = net->batch;
677
678     //args.batch = net->batch;
679
680     //args.batch = net->batch;
681
682     //args.batch = net->batch;
683
684     //args.batch = net->batch;
685
686     //args.batch = net->batch;
687
688     //args.batch = net->batch;
689
690     //args.batch = net->batch;
691
692     //args.batch = net->batch;
693
694     //args.batch = net->batch;
695
696     //args.batch = net->batch;
697
698     //args.batch = net->batch;
699
700     //args.batch = net->batch;
701
702     //args.batch = net->batch;
703
704     //args.batch = net->batch;
705
706     //args.batch = net->batch;
707
708     //args.batch = net->batch;
709
710     //args.batch = net->batch;
711
712     //args.batch = net->batch;
713
714     //args.batch = net->batch;
715
716     //args.batch = net->batch;
717
718     //args.batch = net->batch;
719
720     //args.batch = net->batch;
721
722     //args.batch = net->batch;
723
724     //args.batch = net->batch;
725
726     //args.batch = net->batch;
727
728     //args.batch = net->batch;
729
730     //args.batch = net->batch;
731
732     //args.batch = net->batch;
733
734     //args.batch = net->batch;
735
736     //args.batch = net->batch;
737
738     //args.batch = net->batch;
739
740     //args.batch = net->batch;
741
742     //args.batch = net->batch;
743
744     //args.batch = net->batch;
745
746     //args.batch = net->batch;
747
748     //args.batch = net->batch;
749
750     //args.batch = net->batch;
751
752     //args.batch = net->batch;
753
754     //args.batch = net->batch;
755
756     //args.batch = net->batch;
757
758     //args.batch = net->batch;
759
760     //args.batch = net->batch;
761
762     //args.batch = net->batch;
763
764     //args.batch = net->batch;
765
766     //args.batch = net->batch;
767
768     //args.batch = net->batch;
769
770     //args.batch = net->batch;
771
772     //args.batch = net->batch;
773
774     //args.batch = net->batch;
775
776     //args.batch = net->batch;
777
778     //args.batch = net->batch;
779
780     //args.batch = net->batch;
781
782     //args.batch = net->batch;
783
784     //args.batch = net->batch;
785
786     //args.batch = net->batch;
787
788     //args.batch = net->batch;
789
790     //args.batch = net->batch;
791
792     //args.batch = net->batch;
793
794     //args.batch = net->batch;
795
796     //args.batch = net->batch;
797
798     //args.batch = net->batch;
799
800     //args.batch = net->batch;
801
802     //args.batch = net->batch;
803
804     //args.batch = net->batch;
805
806     //args.batch = net->batch;
807
808     //args.batch = net->batch;
809
810     //args.batch = net->batch;
811
812     //args.batch = net->batch;
813
814     //args.batch = net->batch;
815
816     //args.batch = net->batch;
817
818     //args.batch = net->batch;
819
820     //args.batch = net->batch;
821
822     //args.batch = net->batch;
823
824     //args.batch = net->batch;
825
826     //args.batch = net->batch;
827
828     //args.batch = net->batch;
829
830     //args.batch = net->batch;
831
832     //args.batch = net->batch;
833
834     //args.batch = net->batch;
835
836     //args.batch = net->batch;
837
838     //args.batch = net->batch;
839
840     //args.batch = net->batch;
841
842     //args.batch = net->batch;
843
844     //args.batch = net->batch;
845
846     //args.batch = net->batch;
847
848     //args.batch = net->batch;
849
850     //args.batch = net->batch;
851
852     //args.batch = net->batch;
853
854     //args.batch = net->batch;
855
856     //args.batch = net->batch;
857
858     //args.batch = net->batch;
859
860     //args.batch = net->batch;
861
862     //args.batch = net->batch;
863
864     //args.batch = net->batch;
865
866     //args.batch = net->batch;
867
868     //args.batch = net->batch;
869
870     //args.batch = net->batch;
871
872     //args.batch = net->batch;
873
874     //args.batch = net->batch;
875
876     //args.batch = net->batch;
877
878     //args.batch = net->batch;
879
880     //args.batch = net->batch;
881
882     //args.batch = net->batch;
883
884     //args.batch = net->batch;
885
886     //args.batch = net->batch;
887
888     //args.batch = net->batch;
889
890     //args.batch = net->batch;
891
892     //args.batch = net->batch;
893
894     //args.batch = net->batch;
895
896     //args.batch = net->batch;
897
898     //args.batch = net->batch;
899
900     //args.batch = net->batch;
901
902     //args.batch = net->batch;
903
904     //args.batch = net->batch;
905
906     //args.batch = net->batch;
907
908     //args.batch = net->batch;
909
910     //args.batch = net->batch;
911
912     //args.batch = net->batch;
913
914     //args.batch = net->batch;
915
916     //args.batch = net->batch;
917
918     //args.batch = net->batch;
919
920     //args.batch = net->batch;
921
922     //args.batch = net->batch;
923
924     //args.batch = net->batch;
925
926     //args.batch = net->batch;
927
928     //args.batch = net->batch;
929
930     //args.batch = net->batch;
931
932     //args.batch = net->batch;
933
934     //args.batch = net->batch;
935
936     //args.batch = net->batch;
937
938     //args.batch = net->batch;
939
940     //args.batch = net->batch;
941
942     //args.batch = net->batch;
943
944     //args.batch = net->batch;
945
946     //args.batch = net->batch;
947
948     //args.batch = net->batch;
949
950     //args.batch = net->batch;
951
952     //args.batch = net->batch;
953
954     //args.batch = net->batch;
955
956     //args.batch = net->batch;
957
958     //args.batch = net->batch;
959
960     //args.batch = net->batch;
961
962     //args.batch = net->batch;
963
964     //args.batch = net->batch;
965
966     //args.batch = net->batch;
967
968     //args.batch = net->batch;
969
970     //args.batch = net->batch;
971
972     //args.batch = net->batch;
973
974     //args.batch = net->batch;
975
976     //args.batch = net->batch;
977
978     //args.batch = net->batch;
979
980     //args.batch = net->batch;
981
982     //args.batch = net->batch;
983
984     //args.batch = net->batch;
985
986     //args.batch = net->batch;
987
988     //args.batch = net->batch;
989
990     //args.batch = net->batch;
991
992     //args.batch = net->batch;
993
994     //args.batch = net->batch;
995
996     //args.batch = net->batch;
997
998     //args.batch = net->batch;
999
1000    //args.batch = net->batch;
1001
1002    //args.batch = net->batch;
1003
1004    //args.batch = net->batch;
1005
1006    //args.batch = net->batch;
1007
1008    //args.batch = net->batch;
1009
1010    //args.batch = net->batch;
1011
1012    //args.batch = net->batch;
1013
1014    //args.batch = net->batch;
1015
1016    //args.batch = net->batch;
1017
1018    //args.batch = net->batch;
1019
1020    //args.batch = net->batch;
1021
1022    //args.batch = net->batch;
1023
1024    //args.batch = net->batch;
1025
1026    //args.batch = net->batch;
1027
1028    //args.batch = net->batch;
1029
1030    //args.batch = net->batch;
1031
1032    //args.batch = net->batch;
1033
1034    //args.batch = net->batch;
1035
1036    //args.batch = net->batch;
1037
1038    //args.batch = net->batch;
1039
1040    //args.batch = net->batch;
1041
1042    //args.batch = net->batch;
1043
1044    //args.batch = net->batch;
1045
1046    //args.batch = net->batch;
1047
1048    //args.batch = net->batch;
1049
1050    //args.batch = net->batch;
1051
1052    //args.batch = net->batch;
1053
1054    //args.batch = net->batch;
1055
1056    //args.batch = net->batch;
1057
1058    //args.batch = net->batch;
1059
1060    //args.batch = net->batch;
1061
1062    //args.batch = net->batch;
1063
1064    //args.batch = net->batch;
1065
1066    //args.batch = net->batch;
1067
1068    //args.batch = net->batch;
1069
1070    //args.batch = net->batch;
1071
1072    //args.batch = net->batch;
1073
1074    //args.batch = net->batch;
1075
1076    //args.batch = net->batch;
1077
1078    //args.batch = net->batch;
1079
1080    //args.batch = net->batch;
1081
1082    //args.batch = net->batch;
1083
1084    //args.batch = net->batch;
1085
1086    //args.batch = net->batch;
1087
1088    //args.batch = net->batch;
1089
1090    //args.batch = net->batch;
1091
1092    //args.batch = net->batch;
1093
1094    //args.batch = net->batch;
1095
1096    //args.batch = net->batch;
1097
1098    //args.batch = net->batch;
1099
1100    //args.batch = net->batch;
1101
1102    //args.batch = net->batch;
1103
1104    //args.batch = net->batch;
1105
1106    //args.batch = net->batch;
1107
1108    //args.batch = net->batch;
1109
1110    //args.batch = net->batch;
1111
1112    //args.batch = net->batch;
1113
1114    //args.batch = net->batch;
1115
1116    //args.batch = net->batch;
1117
1118    //args.batch = net->batch;
1119
1120    //args.batch = net->batch;
1121
1122    //args.batch = net->batch;
1123
1124   
```

Dentro desta função, há um loop no qual ocorre o treinamento, mostrado nas Imagens 8 e 9. Na Imagem 8 ocorre inicialmente a preparação das amostras.



```
args->coords = coords;
args->paths = paths;
args->n = imgs;
args->n = plist->size;
args->classes = classes;
args->jitter = jitter;
args->num_boxes = l_max_boxes;
args->d = dbuffer;
args->type = DETECTION_DATA;
//args->type = INSTANCE_DATA;
args->threads = 0;

pthread_t load_thread = load_data(args);
double time;
int count = 0;
//while(1){
while((get_current_batch(net) < net->max_batches){
    if(l_random(66, count++%10) == 0){
        printf("Resizing\n");
        int dim = (rand() % 10 + 10) * 32;
        if (get_current_batch(net)*200 > net->max_batches) dim = 600;
        //int dim = (rand() % 4 + 16) * 32;
        printf("%d\n", dim);
        args->w = dim;
        args->h = dim;

        pthread_join(load_thread, 0);
        train = buffer;
        free_data(train);
        load_thread = load_data(args);

        #pragma omp parallel for
        for(i = 0; i < ngpus; ++i){
            resize_network(nets[i], dim, dim);
        }
        net = nets[0];
    }
    time=what_time_is_it_now();
    pthread_join(load_thread, 0);
    train = buffer;
    load_thread = load_data(args);

    /*
    int k;
    for(k = 0; k < l_max_boxes; ++k){
        box b = float_to_box(train->y.vals[10] + 1 + k*5);
        if(b.x) break;
        printf("Loaded: %f %f %f %f\n", b.x, b.y, b.w, b.h);
    }
    */
}

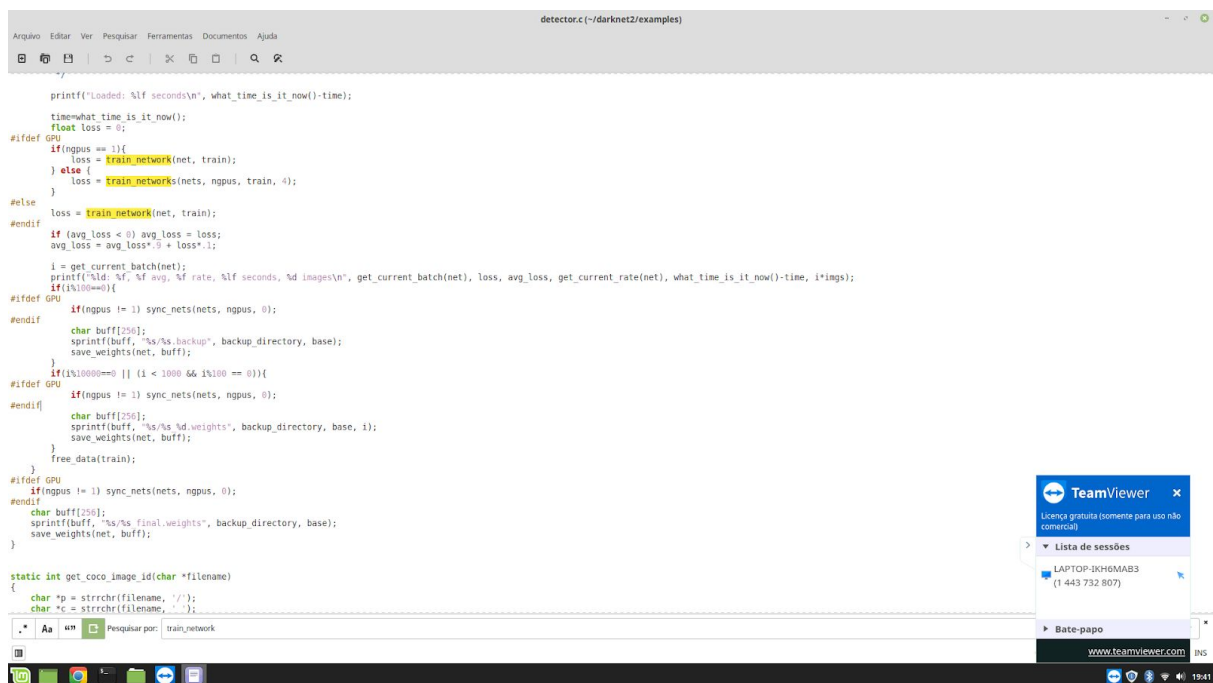
/*
printf("Loaded: %f seconds\n", what_time_is_it_now()-time);
time=what_time_is_it_now();
float loss = 0;
#ifdef GPU
    if(ngpus == 1){
        loss = train_network(net, train);
    } else {
        loss = train_networks(nets, ngpus, train, 4);
    }
#else
    loss = train_network(net, train);
#endif
    if (avg_loss < 0) avg_loss = loss;
    avg_loss = avg_loss*0.9 + loss*.1;

    i = get_current_batch(net);
    printf("%d: %f, %f avg, %f rate, %f seconds, %d images\n", get_current_batch(net), loss, avg_loss, get_current_rate(net), what_time_is_it_now()-time, i*imgs);
    if(i%100==0){
#ifdef GPU
        if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
        char buff[256];
        sprintf(buff, "%s/%s.backup", backup_directory, base);
        save_weights(net, buff);
    }
    if(i%10000==0 || (i < 1000 && i%100 == 0)){
#ifdef GPU
        if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
        char buff[256];
        sprintf(buff, "%s/%s %d.weights", backup_directory, base, i);
        save_weights(net, buff);
    }
    free_data(train);
}
#ifdef GPU
    if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
    char buff[256];
    sprintf(buff, "%s/%s final.weights", backup_directory, base);
    save_weights(net, buff);
}

static int get_coco_image_id(char *filename)
{
    char *p = strrchr(filename, '/');
    char *c = strrchr(filename, '.');
}
```

Imagem 8 - Loop de preparação das amostras

Em seguida, inicia-se então o treinamento, no loop mostrado na Imagem 9. Durante este processo, algumas métricas importantes, como o IoU (Intersection of Union) são mostradas para o usuário. Dessa forma é possível acompanhar se o treinamento está sendo bem sucedido ou não ao longo do processo.



```
printf("Loaded: %f seconds\n", what_time_is_it_now()-time);
time=what_time_is_it_now();
float loss = 0;
#ifdef GPU
    if(ngpus == 1){
        loss = train_network(net, train);
    } else {
        loss = train_networks(nets, ngpus, train, 4);
    }
#else
    loss = train_network(net, train);
#endif
    if (avg_loss < 0) avg_loss = loss;
    avg_loss = avg_loss*0.9 + loss*.1;

    i = get_current_batch(net);
    printf("%d: %f, %f avg, %f rate, %f seconds, %d images\n", get_current_batch(net), loss, avg_loss, get_current_rate(net), what_time_is_it_now()-time, i*imgs);
    if(i%100==0){
#ifdef GPU
        if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
        char buff[256];
        sprintf(buff, "%s/%s.backup", backup_directory, base);
        save_weights(net, buff);
    }
    if(i%10000==0 || (i < 1000 && i%100 == 0)){
#ifdef GPU
        if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
        char buff[256];
        sprintf(buff, "%s/%s %d.weights", backup_directory, base, i);
        save_weights(net, buff);
    }
    free_data(train);
}
#ifdef GPU
    if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
    char buff[256];
    sprintf(buff, "%s/%s final.weights", backup_directory, base);
    save_weights(net, buff);
}

static int get_coco_image_id(char *filename)
{
    char *p = strrchr(filename, '/');
    char *c = strrchr(filename, '.');
}
```

Imagem 9 - Loop de treinamento



## Resultados

Após realizar o predict em vídeo, não foi obtido sucesso, foi observado algumas caixas delimitadoras com a tentativa de realizar a detecção, porém foi em objetos aleatórios. Nas Imagens 10 e 11 é possível observar a detecção.



Imagem 10 - Detecção mal-sucedida



Imagem 11 - Detecção mal-sucedida

Inicialmente, foi pensado que a falha no processo de detecção seria devido a problemas com o database feito pelos integrantes. Entretanto, o processo de treinamento e de predict foi realizado novamente utilizando imagens de base de dados bastante difundidas em estudos da literatura e mesmo assim resultados satisfatórios não foram obtidos. Portanto, para dar uma dimensão do que de fato deveria ter sido o resultado obtido, é mostrado na Imagem 12 os resultados obtidos

de um trabalho anterior realizado no semestre de 2019.1. Neste estudo, o objetivo foi de se realizar a detecção de balões de ar quente.

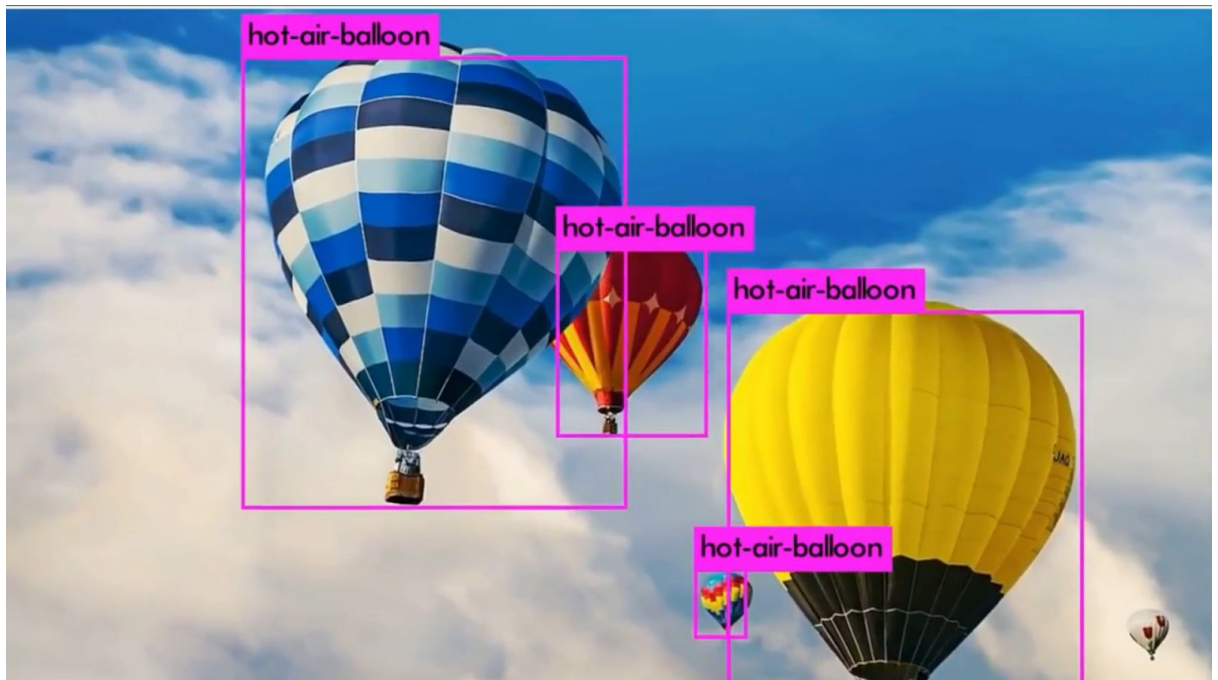


Imagem 12 - Detecção bem-sucedida

Em uma última tentativa de análise do porque o programa não estava fazendo a detecção conforme esperado, o mesmo processo de treinamento utilizando a base de dados de balões de ar quente foi realizado. Foram realizadas exatamente todas as etapas feitas no estudo elaborado no semestre de 2019.1 e o resultado obtido na data deste estudo, em 2019.2, foi insatisfatório. Os balões também não foram detectados com sucesso, assim podemos concluir que alguma falha interna ao algoritmo foi causada devido a atualização da biblioteca OpenCV, realizada na máquina no mês de novembro.

## Conclusão

Com a intenção de verificar qual era o problema que estava sendo gerado, então, foi gerado um novo treinamento utilizando balões, o qual foi um trabalho sucedido utilizado em Tópicos Avançados em Informática II. Para isso, foram utilizados os mesmos parâmetros da época da disciplina. Só que, dessa vez, ao realizar o predict, não foi obtido sucesso, ocorreu o mesmo problema da detecção das placas. Como a versão da biblioteca OpenCV é a única diferença do treinamento anterior, então, pensa-se que o bug gerado seja por causa da nova versão dessa biblioteca.