

50.039 - Theory and Practice of Deep Learning

The Small Project

Done by - Bhavy Mital (1002945) & Nigel Chan Terng Tseng (1002027)

Steps For Running The Code:

1. Get the dependencies from the 'requirements.txt' file.
 - a. Access to your preferred virtual environment.
 - b. At the root folder of the project, run 'pip install -r requirements.txt'.
2. Download the pascal VOC dataset using the following link
http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
3. Extract the files straight into the project root folder that we sent. Make sure the directory tree is something like this:

```
data/
|   VOCdevkit/
|   |   VOC2012/
|   |   |   Annotations/
|   |   |   csvs/
|   |   |   ImageSets/
|   |   |   JPEGImages/
|   |   |   SegmentationClass/
|   |   |   SegmentationObject/
```

4. To train the model, please run all the cells in the Ipython notebook file named 'dl_small_project_velocity_9.ipynb'.
5. And grab a cup of coffee while running the code

Model Used:

We used ResNet-18, a convolutional neural network that was trained on the images in the ImageNet database. To speed things up, we used transfer learning (pre-trained parameters from the ResNet) to optimize and help to improve learning in our model.

Loss Function Used:

The output from our model was a vector of 20 values that ranged from 0 to 1, where each of these soft-labels represented a distinct class. As such, we used the

BCEWithLogitsLoss loss function to get the loss for the 20 separate binary classifiers.

This loss function was used as it combines a Sigmoid layer and the BCELoss in one single class, which is numerically more stable as opposed to using a plain Sigmoid followed by the BCELoss. The numerical stability comes from the advantage of combining the mentioned operations into a single layer and using the log-sum-exp trick.

Training Procedures:

Dataset: Using a slightly altered version of the 'vocparseclslabels.py' file provided combined with our PascalVOCDataset class, we generated a dataset where each instance consisted of the image tensor, and it's respective one-hot encoded labels.

Dataloader: Using the dataset generated above, we created a dataloader using PyTorch's in-built dataloader function, where it has a batch_size of 16 and shuffle is set to true. To ensure the reproducibility of our results we used a random seed. (*torch.manual.seed(7)*).

train_epoch() Function: Runs both the train and validation functions for a set number of epochs and returns the train and validation losses and mean average precision scores for each epoch.

train() Function: Our training function does the standard training procedure where it iterates through the training dataloader batch by batch and calculates the average loss per batch. After it finishes training, it returns the average training loss and mean average precision over all classes.

validation() Function: The validation function basically validates and returns validation loss, together with the total average precision score over all classes. It helps us observe the mean average precision and loss after each validation for the current training epoch.

test() Function: We use this function to test our model after a set number of training epochs. This returns the prediction scores and targets for each image instance along with their image paths for the next function.

ap() Function: This function takes the target label tensor, prediction scores tensor and the total number of classes and returns the mean average precision score.

classwise_ap() Function: This function takes the target label tensor, prediction scores tensor and the list of all possible classes and returns a dictionary with classwise average precision scores.

top_bot_scores_50() Function: This function takes in the image_paths, prediction scores, and the class index, and gets the reordered index of the prediction scores from highest to lowest. This function returns the top 50 highest and top 50 lowest scoring images' paths, along with the reordered index.

img_grid_plotter() Function: Helper function for plotting the top 50 images in a grid of 10x5 for a single class.

plot_top_images() Function: Iterates through a list containing pairs consisting of the class index and its respective top/bottom 50 images, and uses the *ImageOnlyDataset* class to create a dataset followed by a dataloader to be passed into *img_grid_plotter()* function for the image grid plot.

get_tail_acc() Function: This function takes the sorted prediction scores and target labels of a single class and also some other hyperparameters such as *t_min* (defined by the user), *t_max* (max of all prediction scores), *t_num* (defined by the user) and *t_vals* (linspace from *t_min* to *t_max* with *t_num* steps) and returns the tail accuracy.

Hyperparameters:

For Image Normalization (*torchvision.transforms.Normalize*)

Mean: [0.4589, 0.4355, 0.4032]

Standard Deviation: [0.2239, 0.2186, 0.2206]

For Dataloader

Batch Size: 16

Shuffle: True

For Optimizer (Stochastic Gradient Descent)

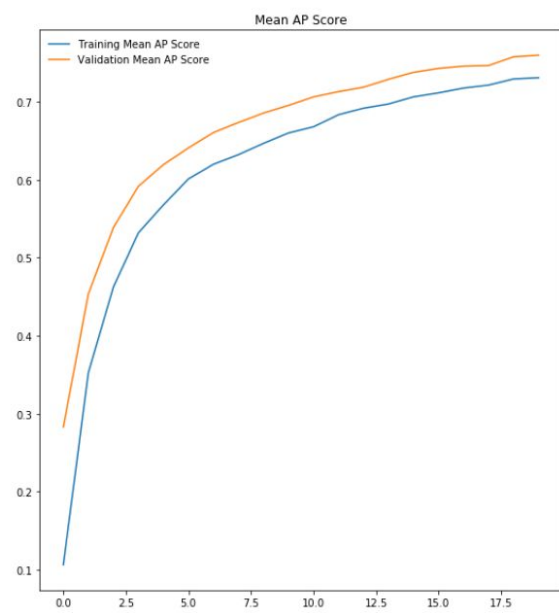
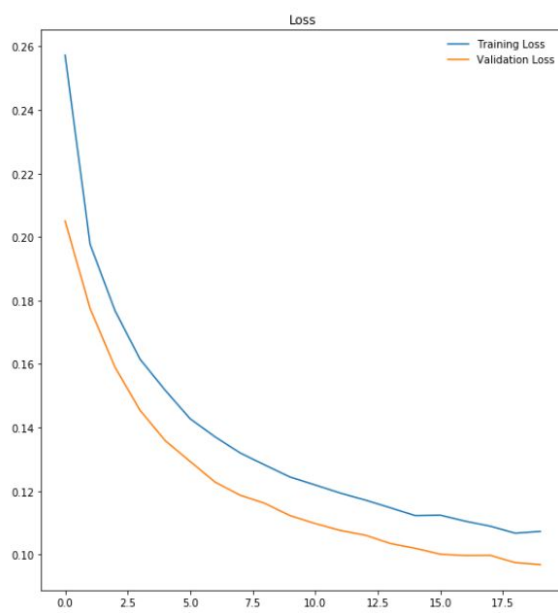
Learning Rate: 0.001

Momentum: 0.9

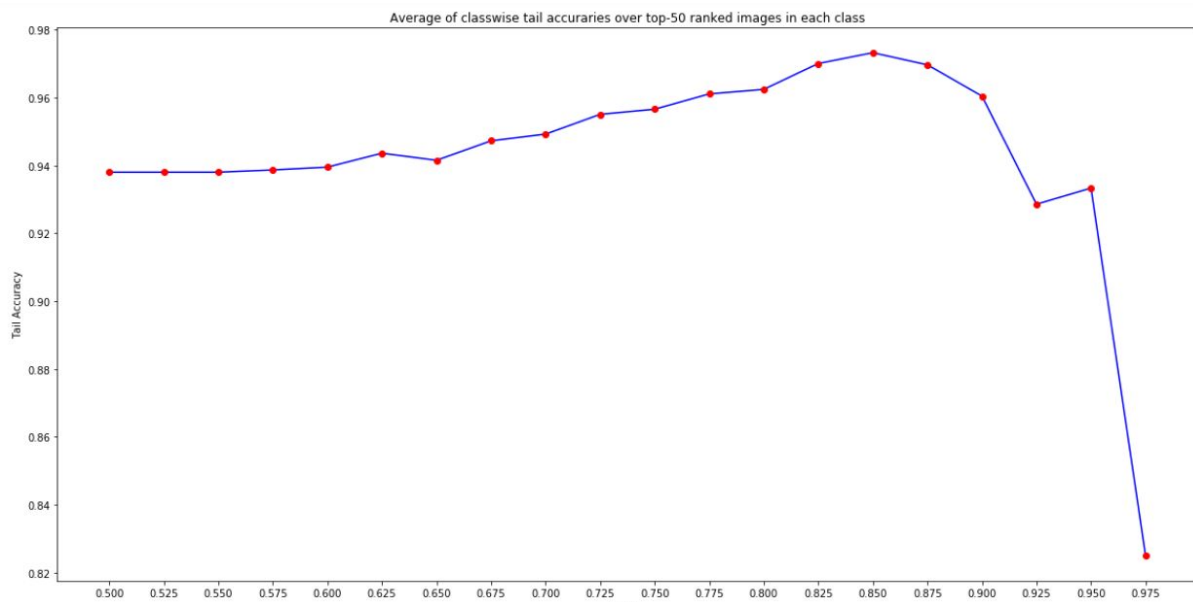
For Training In General

Number of Epochs: 20

Loss and Mean Average Precision Scores over 20 epochs:



Graph of Tailacc(t) averaged over all 20 classes



Top-10 Highest vs Top-10 Lowest Scored Images

Cat

Top 50 highest scored images for class cat



Top 50 lowest scored images for class cat



Dining Table

Top 50 highest scored images for class diningtable



Top 50 lowest scored images for class diningtable



Bus

Top 50 highest scored images for class bus



Top 50 lowest scored images for class bus



Car

Top 50 highest scored images for class car



Top 50 lowest scored images for class car

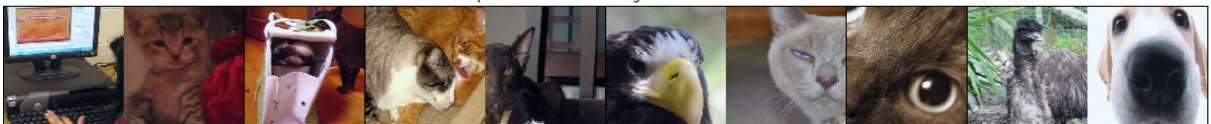


Boat

Top 50 highest scored images for class boat



Top 50 lowest scored images for class boat



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
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
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
84



Plus points for Nezuko please (: