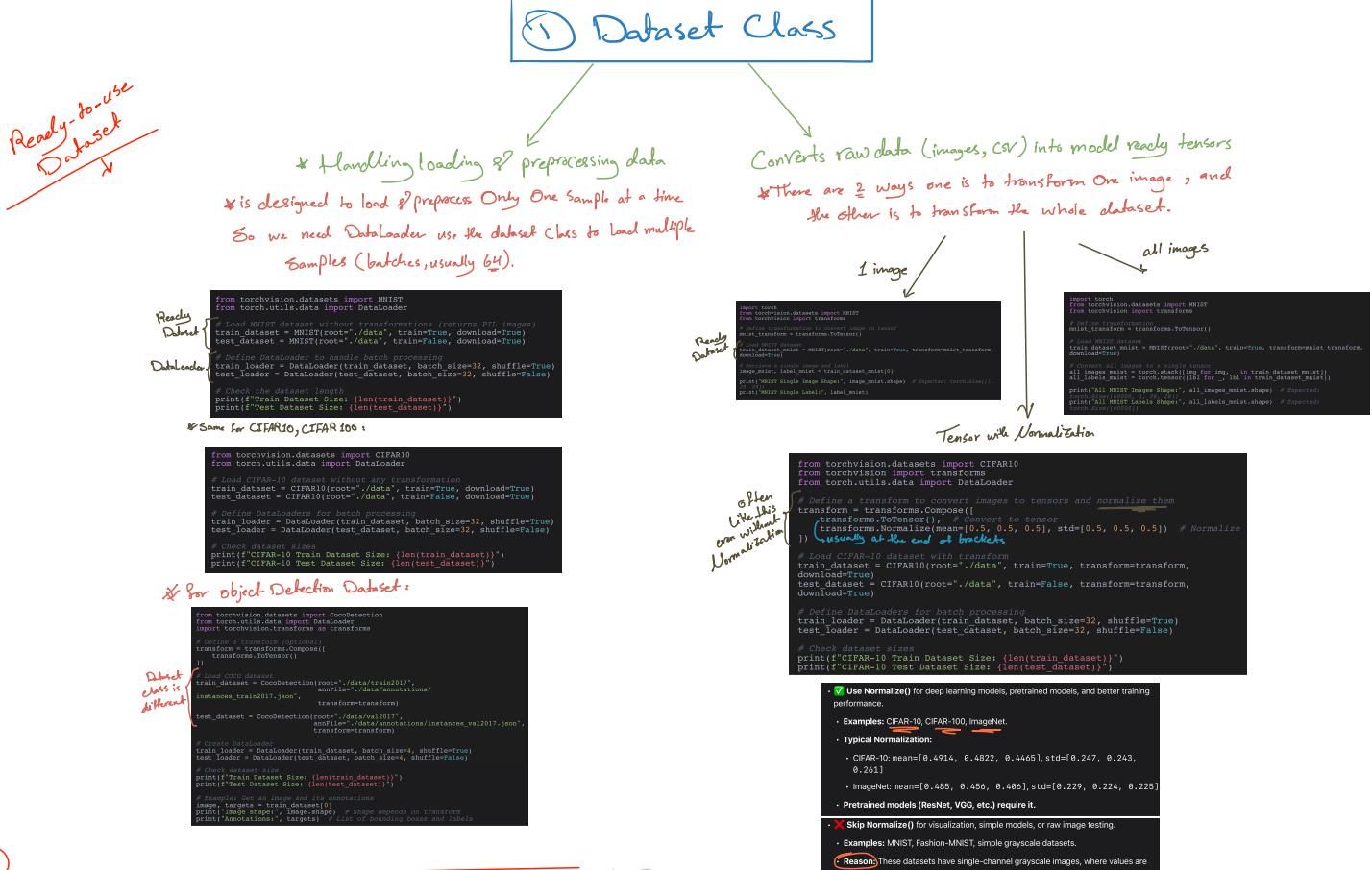


CNN model Workflow in pytorch

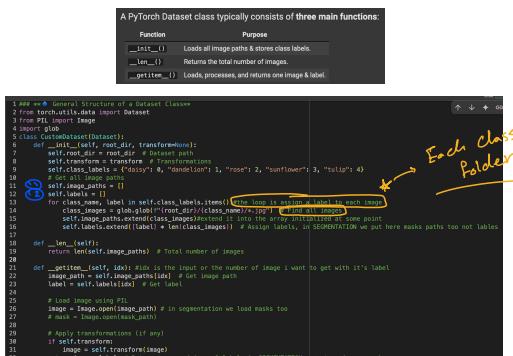


① *** Each Class in one Folder:**



- ① how to read data :
load an image path using (`join`) with data path,
then read it using (`open`)
 - ② Finding path of All images
use `glob.glob` & store it in (`image_files`)
 - ③ Transform To Tensor Step
Load image then apply tensor,
then check its shape
 - ④ Check shape of multiple images, do
in `for` loop if necessary on multiple images.

⑤ DataSet class



⑥ Transform & initialize the dataset



⑦ Data loader to handle batching automatically:

```

120 1 from torch.utils.data import DataLoader
121 2
122 3 # Define batch size
123 4 # Define number of epochs
124 5 # Define DataLoader
125 6
126 7 def train_dataloader(dataset, batch_size=batch_size, num_workers=0):
127 8     return DataLoader(
128 9         dataset,
130 10        batch_size=batch_size,
131 11        shuffle=True,
132 12        num_workers=num_workers)
133 13
134 14
135 15 # Define model
136 16
137 17 # Define loss function
138 18
139 19 # Define optimizer
140 20
141 21
142 22
143 23
144 24
145 25
146 26
147 27
148 28
149 29
150 30
151 31
152 32
153 33
154 34
155 35
156 36
157 37
158 38
159 39
160 40
161 41
162 42
163 43
164 44
165 45
166 46
167 47
168 48
169 49
170 50
171 51
172 52
173 53
174 54
175 55
176 56
177 57
178 58
179 59
180 60
181 61
182 62
183 63
184 64
185 65
186 66
187 67
188 68
189 69
190 70
191 71
192 72
193 73
194 74
195 75
196 76
197 77
198 78
199 79
200 80
201 81
202 82
203 83
204 84
205 85
206 86
207 87
208 88
209 89
210 90
211 91
212 92
213 93
214 94
215 95
216 96
217 97
218 98
219 99
220 100
221 101
222 102
223 103
224 104
225 105
226 106
227 107
228 108
229 109
230 110
231 111
232 112
233 113
234 114
235 115
236 116
237 117
238 118
239 119
240 120
241 121
242 122
243 123
244 124
245 125
246 126
247 127
248 128
249 129
250 130
251 131
252 132
253 133
254 134
255 135
256 136
257 137
258 138
259 139
260 140
261 141
262 142
263 143
264 144
265 145
266 146
267 147
268 148
269 149
270 150
271 151
272 152
273 153
274 154
275 155
276 156
277 157
278 158
279 159
280 160
281 161
282 162
283 163
284 164
285 165
286 166
287 167
288 168
289 169
290 170
291 171
292 172
293 173
294 174
295 175
296 176
297 177
298 178
299 179
300 180
301 181
302 182
303 183
304 184
305 185
306 186
307 187
308 188
309 189
310 190
311 191
312 192
313 193
314 194
315 195
316 196
317 197
318 198
319 199
320 200
321 201
322 202
323 203
324 204
325 205
326 206
327 207
328 208
329 209
330 210
331 211
332 212
333 213
334 214
335 215
336 216
337 217
338 218
339 219
340 220
341 221
342 222
343 223
344 224
345 225
346 226
347 227
348 228
349 229
350 230
351 231
352 232
353 233
354 234
355 235
356 236
357 237
358 238
359 239
360 240
361 241
362 242
363 243
364 244
365 245
366 246
367 247
368 248
369 249
370 250
371 251
372 252
373 253
374 254
375 255
376 256
377 257
378 258
379 259
380 260
381 261
382 262
383 263
384 264
385 265
386 266
387 267
388 268
389 269
390 270
391 271
392 272
393 273
394 274
395 275
396 276
397 277
398 278
399 279
400 280
401 281
402 282
403 283
404 284
405 285
406 286
407 287
408 288
409 289
410 290
411 291
412 292
413 293
414 294
415 295
416 296
417 297
418 298
419 299
420 300
421 301
422 302
423 303
424 304
425 305
426 306
427 307
428 308
429 309
430 310
431 311
432 312
433 313
434 314
435 315
436 316
437 317
438 318
439 319
440 320
441 321
442 322
443 323
444 324
445 325
446 326
447 327
448 328
449 329
450 330
451 331
452 332
453 333
454 334
455 335
456 336
457 337
458 338
459 339
460 340
461 341
462 342
463 343
464 344
465 345
466 346
467 347
468 348
469 349
470 350
471 351
472 352
473 353
474 354
475 355
476 356
477 357
478 358
479 359
480 360
481 361
482 362
483 363
484 364
485 365
486 366
487 367
488 368
489 369
490 370
491 371
492 372
493 373
494 374
495 375
496 376
497 377
498 378
499 379
500 380
501 381
502 382
503 383
504 384
505 385
506 386
507 387
508 388
509 389
510 390
511 391
512 392
513 393
514 394
515 395
516 396
517 397
518 398
519 399
520 400
521 401
522 402
523 403
524 404
525 405
526 406
527 407
528 408
529 409
530 410
531 411
532 412
533 413
534 414
535 415
536 416
537 417
538 418
539 419
540 420
541 421
542 422
543 423
544 424
545 425
546 426
547 427
548 428
549 429
550 430
551 431
552 432
553 433
554 434
555 435
556 436
557 437
558 438
559 439
560 440
561 441
562 442
563 443
564 444
565 445
566 446
567 447
568 448
569 449
570 450
571 451
572 452
573 453
574 454
575 455
576 456
577 457
578 458
579 459
580 460
581 461
582 462
583 463
584 464
585 465
586 466
587 467
588 468
589 469
590 470
591 471
592 472
593 473
594 474
595 475
596 476
597 477
598 478
599 479
600 480
601 481
602 482
603 483
604 484
605 485
606 486
607 487
608 488
609 489
610 490
611 491
612 492
613 493
614 494
615 495
616 496
617 497
618 498
619 499
620 500
621 501
622 502
623 503
624 504
625 505
626 506
627 507
628 508
629 509
630 510
631 511
632 512
633 513
634 514
635 515
636 516
637 517
638 518
639 519
640 520
641 521
642 522
643 523
644 524
645 525
646 526
647 527
648 528
649 529
650 530
651 531
652 532
653 533
654 534
655 535
656 536
657 537
658 538
659 539
660 540
661 541
662 542
663 543
664 544
665 545
666 546
667 547
668 548
669 549
670 550
671 551
672 552
673 553
674 554
675 555
676 556
677 557
678 558
679 559
680 560
681 561
682 562
683 563
684 564
685 565
686 566
687 567
688 568
689 569
690 570
691 571
692 572
693 573
694 574
695 575
696 576
697 577
698 578
699 579
700 580
701 581
702 582
703 583
704 584
705 585
706 586
707 587
708 588
709 589
710 590
711 591
712 592
713 593
714 594
715 595
716 596
717 597
718 598
719 599
720 600
721 601
722 602
723 603
724 604
725 605
726 606
727 607
728 608
729 609
730 610
731 611
732 612
733 613
734 614
735 615
736 616
737 617
738 618
739 619
740 620
741 621
742 622
743 623
744 624
745 625
746 626
747 627
748 628
749 629
750 630
751 631
752 632
753 633
754 634
755 635
756 636
757 637
758 638
759 639
760 640
761 641
762 642
763 643
764 644
765 645
766 646
767 647
768 648
769 649
770 650
771 651
772 652
773 653
774 654
775 655
776 656
777 657
778 658
779 659
780 660
781 661
782 662
783 663
784 664
785 665
786 666
787 667
788 668
789 669
790 670
791 671
792 672
793 673
794 674
795 675
796 676
797 677
798 678
799 679
800 680
801 681
802 682
803 683
804 684
805 685
806 686
807 687
808 688
809 689
810 690
811 691
812 692
813 693
814 694
815 695
816 696
817 697
818 698
819 699
820 700
821 701
822 702
823 703
824 704
825 705
826 706
827 707
828 708
829 709
830 710
831 711
832 712
833 713
834 714
835 715
836 716
837 717
838 718
839 719
840 720
841 721
842 722
843 723
844 724
845 725
846 726
847 727
848 728
849 729
850 730
851 731
852 732
853 733
854 734
855 735
856 736
857 737
858 738
859 739
860 740
861 741
862 742
863 743
864 744
865 745
866 746
867 747
868 748
869 749
870 750
871 751
872 752
873 753
874 754
875 755
876 756
877 757
878 758
879 759
880 760
881 761
882 762
883 763
884 764
885 765
886 766
887 767
888 768
889 769
890 770
891 771
892 772
893 773
894 774
895 775
896 776
897 777
898 778
899 779
900 780
901 781
902 782
903 783
904 784
905 785
906 786
907 787
908 788
909 789
910 790
911 791
912 792
913 793
914 794
915 795
916 796
917 797
918 798
919 799
920 800
921 801
922 802
923 803
924 804
925 805
926 806
927 807
928 808
929 809
930 810
931 811
932 812
933 813
934 814
935 815
936 816
937 817
938 818
939 819
940 820
941 821
942 822
943 823
944 824
945 825
946 826
947 827
948 828
949 829
950 830
951 831
952 832
953 833
954 834
955 835
956 836
957 837
958 838
959 839
960 840
961 841
962 842
963 843
964 844
965 845
966 846
967 847
968 848
969 849
970 850
971 851
972 852
973 853
974 854
975 855
976 856
977 857
978 858
979 859
980 860
981 861
982 862
983 863
984 864
985 865
986 866
987 867
988 868
989 869
990 870
991 871
992 872
993 873
994 874
995 875
996 876
997 877
998 878
999 879
1000 880
1001 881
1002 882
1003 883
1004 884
1005 885
1006 886
1007 887
1008 888
1009 889
1010 890
1011 891
1012 892
1013 893
1014 894
1015 895
1016 896
1017 897
1018 898
1019 899
1020 900
1021 901
1022 902
1023 903
1024 904
1025 905
1026 906
1027 907
1028 908
1029 909
1030 910
1031 911
1032 912
1033 913
1034 914
1035 915
1036 916
1037 917
1038 918
1039 919
1040 920
1041 921
1042 922
1043 923
1044 924
1045 925
1046 926
1047 927
1048 928
1049 929
1050 930
1051 931
1052 932
1053 933
1054 934
1055 935
1056 936
1057 937
1058 938
1059 939
1060 940
1061 941
1062 942
1063 943
1064 944
1065 945
1066 946
1067 947
1068 948
1069 949
1070 950
1071 951
1072 952
1073 953
1074 954
1075 955
1076 956
1077 957
1078 958
1079 959
1080 960
1081 961
1082 962
1083 963
1084 964
1085 965
1086 966
1087 967
1088 968
1089 969
1090 970
1091 971
1092 972
1093 973
1094 974
1095 975
1096 976
1097 977
1098 978
1099 979
1100 980
1101 981
1102 982
1103 983
1104 984
1105 985
1106 986
1107 987
1108 988
1109 989
1110 990
1111 991
1112 992
1113 993
1114 994
1115 995
1116 996
1117 997
1118 998
1119 999
1120 1000
1121 1001
1122 1002
1123 1003
1124 1004
1125 1005
1126 1006
1127 1007
1128 1008
1129 1009
1130 1010
1131 1011
1132 1012
1133 1013
1134 1014
1135 1015
1136 1016
1137 1017
1138 1018
1139 1019
1140 1020
1141 1021
1142 1022
1143 1023
1144 1024
1145 1025
1146 1026
1147 1027
1148 1028
1149 1029
1150 1030
1151 1031
1152 1032
1153 1033
1154 1034
1155 1035
1156 1036
1157 1037
1158 1038
1159 1039
1160 1040
1161 1041
1162 1042
1163 1043
1164 1044
1165 1045
1166 1046
1167 1047
1168 1048
1169 1049
1170 1050
1171 1051
1172 1052
1173 1053
1174 1054
1175 1055
1176 1056
1177 1057
1178 1058
1179 1059
1180 1060
1181 1061
1182 1062
1183 1063
1184 1064
1185 1065
1186 1066
1187 1067
1188 1068
1189 1069
1190 1070
1191 1071
1192 1072
1193 1073
1194 1074
1195 1075
1196 1076
1197 1077
1198 1078
1199 1079
1200 1080
1201 1081
1202 1082
1203 1083
1204 1084
1205 1085
1206 1086
1207 1087
1208 1088
1209 1089
1210 1090
1211 1091
1212 1092
1213 1093
1214 1094
1215 1095
1216 1096
1217 1097
1218 1098
1219 1099
1220 1100
1221 1101
1222 1102
1223 1103
1224 1104
1225 1105
1226 1106
1227 1107
1228 1108
1229 1109
1230 1110
1231 1111
1232 1112
1233 1113
1234 1114
1235 1115
1236 1116
1237 1117
1238 1118
1239 1119
1240 1120
1241 1121
1242 1122
1243 1123
1244 1124
1245 1125
1246 1126
1247 1127
1248 1128
1249 1129
1250 1130
1251 1131
1252 1132
1253 1133
1254 1134
1255 1135
1256 1136
1257 1137
1258 1138
1259 1139
1260 1140
1261 1141
1262 1142
1263 1143
1264 1144
1265 1145
1266 1146
1267 1147
1268 1148
1269 1149
1270 1150
1271 1151
1272 1152
1273 1153
1274 1154
1275 1155
1276 1156
1277 1157
1278 1158
1279 1159
1280 1160
1281 1161
1282 1162
1283 1163
1284 1164
1285 1165
1286 1166
1287 1167
1288 1168
1289 1169
1290 1170
1291 1171
1292 1172
1293 1173
1294 1174
1295 1175
1296 1176
1297 1177
1298 1178
1299 1179
1300 1180
1301 1181
1302 1182
1303 1183
1304 1184
1305 1185
1306 1186
1307 1187
1308 1188
1309 1189
1310 1190
1311 1191
1312 1192
1313 1193
1314 1194
1315 1195
1316 1196
1317 1197
1318 1198
1319 1199
1320 1200
1321 1201
1322 1202
1323 1203
1324 1204
1325 1205
1326 1206
1327 1207
1328 1208
1329 1209
1330 1210
1331 1211
1332 1212
1333 1213
1334 1214
1335 1215
1336 1216
1337 1217
1338 1218
1339 1219
1340 1220
1341 1221
1342 1222
1343 1223
1344 1224
1345 1225
1346 1226
1347 1227
1348 1228
1349 1229
1350 1230
1351 1231
1352 1232
1353 1233
1354 1234
1355 1235
1356 1236
1357 1237
1358 1238
1359 1239
1360 1240
1361 1241
1362 1242
1363 1243
1364 1244
1365 1245
1366 1246
1367 1247
1368 1248
1369 1249
1370 1250
1371 1251
1372 1252
1373 1253
1374 1254
1375 1255
1376 1256
1377 1257
1378 1258
1379 1259
1380 1260
1381 1261
1382 1262
1383 1263
1384 1264
1385 1265
1386 1266
1387 1267
1388 1268
1389 1269
1390 1270
1391 1271
1392 1272
1393 1273
1394 1274
1395 1275
1396 1276
1397 1277
1398 1278
1399 1279
1400 1280
1401 1281
1402 1282
1403 1283
1404 1284
1405 1285
1406 1286
1407 1287
1408 1288
1409 1289
1410 1290
1411 1291
1412 1292
1413 1293
1414 1294
1415 1295
1416 1296
1417 1297
1418 1298
1419 1299
1420 1300
1421 1301
1422 1302
1423 1303
1424 1304
1425 1305
1426 1306
1427 1307
1428 1308
1429 1309
1430 1310
1431 1311
1432 1312
1433 1313
1434 1314
1435 1315
1436 1316
1437 1317
1438 1318
1439 1319
1440 1320
1441 1321
1442 1322
1443 1323
1444 1324
1445 1325
1446 1326
1447 1327
1448 1328
1449 1329
1450 1330
1451 1331
1452 1332
1453 1333
1454 1334
1455 1335
1456 1336
1457 1337
1458 1338
1459 1339
1460 1340
1461 1341
1462 1342
1463 1343
1464 1344
1465 1345
1466 1346
1467 1347
1468 1348
1469 1349
1470 1350
1471 1351
1472 1352
1473 1353
1474 1354
1475 1355
1476 1356
1477 1357
1478 1358
1479 1359
1480 1360
1481 1361
1482 1362
1483 1363
1484 1364
1485 1365
1486 1366
1487 1367
1488 1368
1489 1369
1490 1370
1491 1371
1492 1372
1493 1373
1494 1374
1495 1375
1496 1376
1497 1377
1498 1378
1499 1379
1500 1380
1501 1381
1502 1382
1503 1383
1504 1384
1505 1385
1506 1386
1507 1387
1508 1388
1509 1389
1510 1390
1511 1391
1512 1392
1513 1393
1514 1394
1515 1395
1516 1396
1517 1397
1518 1398
1519 1399
1520 1400
1521 1401
1522 1402
1523 1403
1524 1404
1525 1405
1526 1406
1527 1407
1528 1408
1529 1409
1530 1410
1531 1411
1532 1412
1533 1413
1534 1414
1535 1415
1536 1416
1537 1417
1538 1418
1539 1419
1540 1420
1541 1421
1542 1422
1543 1423
1544 1424
1545 1425
1546 1426
1547 1427
1548 1428

```

* we can replace Dataset Class (ImageFolder)

```
Call It →
1 from torch.utils.data import DataLoader
2 from torchvision.datasets import ImageFolder # if you have dataset that each class in folder
3 import os
4 # Define transformations
5 transform = transforms.Compose([
6     transforms.Resize(64, 64), # Resize Images
7     transforms.RandomRotation(15), # Rotate Images randomly within ±15 degrees
8     transforms.ColorJitter(brightness=0.2), # Adjust brightness randomly
9     transforms.ToTensor(), # Convert to tensor
10    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]), # value for each channel
11 ])
12
13 # Validation and testing data typically don't require augmentations, as we only evaluate the model performance on these sets.
14 # Instead, we apply basic transformations to prepare the images.
15 transform_valid_test = transforms.Compose([
16     transforms.Resize(64, 64), # Resize Images to 64x64
17     transforms.ToTensor(), # Convert to tensor
18     transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
19 ])
20
21 # Initialize dataset for Train
22 train_path = os.path.join(path, "Skin cancer ISIC The International Skin Imaging Collaboration", "Train")
23 test_path = os.path.join(path, "Skin cancer ISIC The International Skin Imaging Collaboration", "Test")
24
25 train_dataset = ImageFolder(train_path, transform=transform) #pass ImageFolder here, equivlant to Dataset class that has len and getitem
26 test_dataset = ImageFolder(test_path, transform=transform_valid_test) # Replaced SkinCancerDataset with ImageFolder
27
28 # Create Dataloder
29 batch_size = 32
30 train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
31 test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
32
33 # Get a batch of training Images
34 images, labels = next(iter(train_loader))
35 print("Batch shape: {} Images, Labels: {}".format(images.shape, labels))
```

③ Folders inside Folders

```
Smoking_Dataset/
|-- Training/ Folder
|   |-- Training/ Folder
|   |   |-- notsmoking_0006.jpg
|   |   |-- smoking_0007.jpg
|
|-- Validation/
|   |-- Validation/
|   |   |-- notsmoking_0012.jpg
|   |   |-- smoking_0015.jpg
|
|-- Testing/
|   |-- Testing/
|   |   |-- notsmoking_0032.jpg
|   |   |-- smoking_0034.jpg
```

The labels in the image name

(S) Extract labels outside the class for simplicity

```

    dataset = ds
    dataset.root = dataset.root + "train"
    dataset.image_paths = glob.glob(dataset.root + "/*")
    dataset.labels = []
    dataset._load_labels()
    dataset.root = dataset.root + "val"
    dataset.image_paths = glob.glob(dataset.root + "/*")
    dataset.labels = []
    dataset._load_labels()
    dataset.root = dataset.root + "test"
    dataset.image_paths = glob.glob(dataset.root + "/*")
    dataset.labels = []
    dataset._load_labels()

    train_labels = []
    val_labels = []
    test_labels = []

    for img in dataset.image_paths:
        if "train" in img:
            train_labels.append(1)
        elif "val" in img:
            val_labels.append(0)
        else:
            test_labels.append(1)

    train_labels.append(0)
    val_labels.append(0)
    test_labels.append(0)

    for path in dataset.image_paths:
        if "train" in path:
            print("Training")
        elif "val" in path:
            print("Validation")
        else:
            print("Testing")

```

② Dataset Class

```

1 from torch.utils.data import Dataset
2 from PIL import Image
3
4 class SmokingDataset(Dataset):
5     def __init__(self, image_paths, labels, transform=None):
6         self.image_paths = image_paths # Corresponding image paths
7         self.labels = labels # Corresponding labels
8         self.transform = transform # Transformations to apply
9
10    def __len__(self):
11        return len(self.image_paths) # Total number of images
12
13    def __getitem__(self, idx):
14        image_path = self.image_paths[idx] # Get image path
15        label = self.labels[idx] # Get corresponding label
16
17        # Load image
18        image = Image.open(image_path)
19
20        # Apply transformations (if any)
21        if self.transform:
22            image = self.transform(image)
23
24        return image, label # Return processed image and its label

```

we already extract labels!

③ Transform & initialize data & Dataloader:

```
1 From torchvision import transforms
2 From torch.utils.data import DataLoader
3
4 # Define transformations
5 train_transform = transforms.Compose([
6     transforms.Resize((224, 224)),
7     transforms.RandomRotation(15),
8     transforms.RandomResizedCrop(224, scale=(0.8, 1.2)),
9     transforms.ToTensor(),
10    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
11])
12
13 transform_val_test = transforms.Compose([
14    transforms.Resize((224, 224)),
15    transforms.ToTensor(),
16    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
17])
18
19 # Initialize datasets
20 train_dataset = SmogDataset(train_image_paths, train_labels, transform=train_transform)
21 valid_dataset = SmogDataset(valid_image_paths, valid_labels, transform=transform_val_test)
22 test_dataset = SmogDataset(test_image_paths, test_labels, transform=transform_val_test)
23
24 # Create DataLoaders
25 train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
26 valid_loader = DataLoader(valid_dataset, batch_size=32, shuffle=False)
27 test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
28
29 # Get a batch of training images
30 images, labels = next(iter(train_loader))
31 print(f'Batch size: {len(images)}, Images: {images}, Labels: {labels}')
```

Labels are inside file paths

flat design

- 27
- 01_001.png
- 01_002.png
- 01_003.png
- 01_004.png
- 01_005.png
- 01_006.png
- 01_007.png
- 01_008.png
- 01_009.png
- 01_010.png
- 01_011.png
- 01_012.png
- 01_013.png
- 01_014.png
- 01_015.png
- 01_016.png
- 01_017.png
- ns ns1.mnw

① Extract Labels outside the data class

Important

```

# 2 Import necessary libraries
# 3 import dict = 0
# 4 import numpy as np
# 5 from os import Path
# 6 import cv2
# 7 import random
# 8 import time
# 9 import pandas as pd
# 10 import tensorflow as tf
# 11 import keras
# 12 # Enable the option to load truncated images.
# 13 # This is useful if you want to attempt loading images even if they are corrupted or incomplete.
# 14 Imagefile.FLAG_TRUNCATED_IMAGES = True

# 15 num_SAMPLES = 100 # only include labels with as many samples
# 16 # Initialize empty lists to store file names & labels
# 17 fileNames = []
# 18 labels = []
# 19 labels_dict = {'0': 'Arabian', '1': 'Andalusian', '2': 'Akhal-Tekin', '3': 'Appaloosa', '4': 'AQHA', '5': 'Arabian', '6': 'Arabian', '7': 'Arabian', '8': 'Arabian', '9': 'Arabian', '10': 'Arabian', '11': 'Arabian', '12': 'Arabian', '13': 'Arabian', '14': 'Arabian', '15': 'Arabian', '16': 'Arabian', '17': 'Arabian', '18': 'Arabian', '19': 'Arabian', '20': 'Arabian', '21': 'Arabian', '22': 'Arabian', '23': 'Arabian', '24': 'Arabian', '25': 'Arabian', '26': 'Arabian', '27': 'Arabian', '28': 'Arabian', '29': 'Arabian', '30': 'Arabian', '31': 'Arabian', '32': 'Arabian', '33': 'Arabian', '34': 'Arabian', '35': 'Arabian', '36': 'Arabian', '37': 'Arabian', '38': 'Arabian', '39': 'Arabian', '40': 'Arabian', '41': 'Arabian', '42': 'Arabian', '43': 'Arabian', '44': 'Arabian', '45': 'Arabian', '46': 'Arabian', '47': 'Arabian', '48': 'Arabian', '49': 'Arabian', '50': 'Arabian', '51': 'Arabian', '52': 'Arabian', '53': 'Arabian', '54': 'Arabian', '55': 'Arabian', '56': 'Arabian', '57': 'Arabian', '58': 'Arabian', '59': 'Arabian', '60': 'Arabian', '61': 'Arabian', '62': 'Arabian', '63': 'Arabian', '64': 'Arabian', '65': 'Arabian', '66': 'Arabian', '67': 'Arabian', '68': 'Arabian', '69': 'Arabian', '70': 'Arabian', '71': 'Arabian', '72': 'Arabian', '73': 'Arabian', '74': 'Arabian', '75': 'Arabian', '76': 'Arabian', '77': 'Arabian', '78': 'Arabian', '79': 'Arabian', '80': 'Arabian', '81': 'Arabian', '82': 'Arabian', '83': 'Arabian', '84': 'Arabian', '85': 'Arabian', '86': 'Arabian', '87': 'Arabian', '88': 'Arabian', '89': 'Arabian', '90': 'Arabian', '91': 'Arabian', '92': 'Arabian', '93': 'Arabian', '94': 'Arabian', '95': 'Arabian', '96': 'Arabian', '97': 'Arabian', '98': 'Arabian', '99': 'Arabian', '100': 'Arabian'}
```

no + necessary

```

# 4 Create a Pandas DataFrame from the collected file names and labels
# 5 df = pd.DataFrame.from_dict({'label': labels, 'image': fileNames})
# 6 df.to_csv('train.csv')

# 7 labels = list(df['label'].values)
# 8 print(labels)
# 9 print(df.shape)
```

② Dataset Class & Transform & Initialize

③ Data Loader:

```

1 # Import libraries
2 import tensorflow as tf
3 from tensorflow import keras
4 from tensorflow.keras import layers
5 from tensorflow.keras import datasets, layers, models
6 import numpy as np
7 import matplotlib.pyplot as plt

8 # Set up training data
9 (train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
10 train_images = train_images / 255.0
11 test_images = test_images / 255.0
12 train_labels = train_labels.astype("float32")
13 test_labels = test_labels.astype("float32")
14 train_labels = keras.utils.to_categorical(train_labels, 10)
15 test_labels = keras.utils.to_categorical(test_labels, 10)

16 # Create a neural network model
17 model = models.Sequential([
18     layers.Flatten(input_shape=(28, 28)),
19     layers.Dense(128, activation="relu"),
20     layers.Dense(10, activation="softmax")
21 ])
22
23 # Compile the model
24 model.compile(optimizer="adam",
25                 loss="categorical_crossentropy",
26                 metrics=["accuracy"])
27
28 # Train the model
29 epochs = 10
30 batch_size = 64
31 train_size = 60000
32 validation_size = 10000
33
34 history = model.fit(
35     train_images[:train_size],
36     train_labels[:train_size],
37     batch_size=batch_size,
38     epochs=epochs,
39     validation_data=(train_images[validation_size:],
40                      train_labels[validation_size:]),
41     shuffle=True)
42
43 # Evaluate the model
44 eval_size = 10000
45 eval_loss, eval_accuracy = model.evaluate(
46     test_images[:eval_size],
47     test_labels[:eval_size])
48 print(f"Test accuracy: {eval_accuracy * 100}%")

```

② Model Class



```

def forward(self, x): # If we use then not only determine them
    # Define the forward pass (how data flows through the model).
    x = self.conv1(x) # Convolution
    x = self.relu1(x) # Activation Function
    x = x.view(-1, 16*14*14) # Flatten
    x = self.fc1(x) # Fully Connected Layer
    x = self.fc2(x) # Convert logits to probabilities
    return x # Return probability distribution
  
```

nn.Linear
(16*14*14, 10)

③ Fully Connected layer

* Consist of dimension changing, Linear layer & activation function

⇒ change the dimensions is important to prepare images for Linear.

```

1 How to handle images after with dimensions
2 class Flatten(nn.Module):
3     def __init__(self):
4         super(Flatten, self).__init__()
5     def forward(self, x):
6         x = x.view(x.size(0), -1)
7         return x
8
9 class Squeeze(nn.Module):
10    def __init__(self, dim):
11        super(Squeeze, self).__init__()
12        self.dim = dim
13    def forward(self, x):
14        x = x.squeeze(dim)
15        return x
16
17 class Unsqueeze(nn.Module):
18    def __init__(self, dim):
19        super(Unsqueeze, self).__init__()
20        self.dim = dim
21    def forward(self, x):
22        x = x.unsqueeze(dim)
23        return x
24
25 class Permute(nn.Module):
26    def __init__(self, perm):
27        super(Permute, self).__init__()
28        self.perm = perm
29    def forward(self, x):
30        x = x.permute(*self.perm)
31        return x
32
33 class View(nn.Module):
34    def __init__(self, shape):
35        super(View, self).__init__()
36        self.shape = shape
37    def forward(self, x):
38        x = x.view(*self.shape)
39        return x
  
```

⇒ Linear layer is typically used as the final layer in CNNs

for Classification: (nn.Linear)

- Consist of (channels * height * width, classes)
- if you don't know the dimensions → MNIST classes 0-9 (10)
use trial-and-error, or the formula.
- CIFAR10 classes (10)
CIFAR100 classes (100)

Maintaining tensor shapes is essential in deep learning. PyTorch provides several functions to modify tensor dimensions.

- ◆ **Flatten**
 - Converts any shape to (batch_size, features).
 - Example:


```
(batch_size, channels, height, width) → (batch_size, features)
```
- ◆ **Squeeze**
 - Removes dimensions with size 1.
 - Example:


```
(1, 32, 3, 28, 28) → (32, 3, 28, 28)
```
- ◆ **Unsqueeze**
 - Adds a dimension with size 1 at a specified position.
 - Example:


```
(3, 28, 28) → (1, 3, 28, 28)
```
- ◆ **View (works similar to reshape)**
 - Reshapes a tensor freely while maintaining the same number of elements.
 - Example:


```
(32, 28, 28, 3) → view(-1, 28*28*3) → (32, 28*28*3)
```

	Operations	Parameters	Outputs	Sample Transformation
Flatten	→ (batch_size, ...)	Convert tensor to (batch, features)	(batch_size, ...)	(32, 3, 28, 28) → (32, 3*28*28)
Squeeze	→ (..., 1)	Remove dim of size 1	(..., 1)	(1, 3, 28, 28) → (3, 28, 28)
Unsqueeze	→ (..., 1)	Add dim of size 1	(..., 1)	(3, 28, 28) → (3, 1, 28, 28)
Permute	→ (permuted)	Change order of dimensions	(..., 1, ..., 2)	(32, 28, 28, 3) → (32, 1, 28, 28)
View	→ (shape)	Reshape freely	(..., 1, ..., 2)	(32, 28, 28, 3) → (32, 28*28*3)

might be different based on the goal

- ⇒ Activation functions
 - use Softmax for multi-class classification, converts logits into a probability.
 - typically paired with CrossEntropyLoss (which often softmax applied it internally)
- use Sigmoid for binary-class classification, or multi-label classification (image have multi tags).
- Commonly used with Binary CrossEntropy Loss
- use Tanh for Regression; its range (-1, 1)
non-probabilistic output.

⇒ Full:

```

1 import torch
2 import torch.nn as nn
3
4 class CustomModel(nn.Module):
5     def __init__(self):
6         super(CustomModel, self).__init__()
7         # Define all layers in the model.
8         self.conv1 = nn.Conv2d(in_channels=2, out_channels=16, kernel_size=3, stride=1, padding=1) #out_channels usually you choose it 16 or 32 so 28*28
9         self.relu1 = nn.ReLU()
10        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
11
12        # Convolutional Layer + Activation + Pooling
13        self.conv2 = nn.Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding=1) #out_channels usually you choose it 16 or 32 so 14*14
14        self.relu2 = nn.ReLU()
15
16        self.fc1 = nn.Linear(16*14*14, 10) # Output 10 classes, 16*14*14 using the formula
17
18        # Fully Connected Layer
19        self.fc2 = nn.Linear(10, 10) # What is the expected size after doing the mapping? Check above to know the original size
20
21        # ?? Layer
22        self.softmax = nn.Softmax(dim=1)
23
24
  
```

And we can use pretrained model
(AlexNet, ResNet, UNet....)

③ Training Loop

- ① Create a function Consist of model, dataloader, criterion, optimizer, device:
- ⇒ Set the model to training mode & Set loss, correct, total to Zero
 - ⇒ Begin the Loop for images, labels in `tqdm(dataloader)`:
 - Transfer images, labels to the GPU &&
 - Forward pass & Compute loss
 - Update weights (Backpropagation)
 - Calculate Total Loss for accuracy
 - Track accuracy (use softmax here if you use CrossEntropyLoss)
 - ⇒ Complete accuracy outside the for loop but under the function

```

1 from tqdm import tqdm    # Shows progress bar
2
3 # ◆ Training Loop
4 def train_one_epoch(model, dataloader, criterion, optimizer, device):
5     model.train()          # Set model to training mode
6     total_loss = 0
7     correct = 0
8     total = 0
9
10    for images, labels in tqdm(dataloader):
11        images, labels = images.to(device), labels.to(device)
12
13
14        outputs = model(images)    # Forward pass
15        loss = criterion(outputs, labels)    # Compute loss
16
17        optimizer.zero_grad()    # Reset gradients
18        loss.backward()          # Backpropagation
19        optimizer.step()         # Update weights
20
21        total_loss += loss.item()
22
23        # Track accuracy
24        outputs = torch.softmax(outputs, dim=1)
25        predictions = outputs.argmax(dim=1)    # Get class with highest probability
26        correct += (predictions == labels).sum().item()
27        total += labels.size(0)
28
29    avg_loss = total_loss / len(dataloader)
30    accuracy = 100 * correct / total    # Compute accuracy in percentage
31    return avg_loss, accuracy
  
```

→ In the function we have
to specify which criterion,
optimizer we have to use
for each goal.

multi-class classification ⇒ `CrossEntropyLoss()`

Binary-class

- with sigmoid ⇒ `BCELoss()`
- without sigmoid ⇒ `BCEWithLogitsLoss()`

Regression or AutoEncoder ⇒ `MSELoss()`

③ Validation Loop

* Same as Training Loop, except updating weights *

① Create a function Consist of model, dataloader, criterion, optimizer, device:

⇒ set the model to Validation mode & Set loss, correct, total to Zero

⇒ Disable gradient Computation (with torch.no_grad()) above the loop

⇒ Begin the Loop for images, Labels in (dataloader):

• Transfer images, Labels to the GPU **

• Forward pass & Compute loss.

• Calculate Total Loss for accuracy

• Track accuracy (use softmax here if you use CrossEntropyLoss)

⇒ Compute accuracy outside the for loop but under the function

```
33 # ◆ Validation Loop
34 def validate(model, dataloader, criterion, device):
35     model.eval() # Set model to evaluation mode
36     total_loss = 0
37     correct = 0
38     total = 0
39
40     with torch.no_grad(): # Disable gradient computation
41         for images, labels in dataloader:
42             images, labels = images.to(device), labels.to(device)
43
44             outputs = model(images) # Forward pass
45             loss = criterion(outputs, labels) # Compute loss
46             total_loss += loss.item()
47
48             # Compute accuracy
49             outputs = torch.softmax(outputs, dim=1)
50             predictions = outputs.argmax(dim=1) # Get predicted class
51             correct += (predictions == labels).sum().item()
52             total += labels.size(0)
53
54     avg_loss = total_loss / len(dataloader)
55     accuracy = 100 * correct / total # Compute accuracy in percentage
56     return avg_loss, accuracy
```

Running Training + store of plot metrics

① Initialize the model as object not a class, then Transfer it to GPU.

② Define Loss function & optimizer & # of epochs

③ Training process: Loop for epoch

④ Lists to store metrics, then store them & print

⑤ plot Loss curve, Plot accuracy curve

⑥ You can show predicted labels on images as function

```

1 import torch.optim as optim
2
3 # Initialize the model
4 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
5 model = CNNModel().to(device)
6
7 # Define loss function and optimizer
8 criterion = nn.CrossEntropyLoss() # Multi-class Classification loss (Input: Logits, not probabilities)
9 optimizer = optim.Adam(model.parameters(), lr=0.001) # Adam optimizer
10 num_epochs = 10 # Number of epochs
11
12
13 # Lists to store metrics
14 train_losses = []
15 val_losses = []
16 train_accuracies = []
17 val_accuracies = []
18
19 # Training process
20 for epoch in range(num_epochs):
21     train_loss, train_accuracy = train_one_epoch(model, train_loader, criterion, optimizer, device)
22     val_loss, val_accuracy = validate(model, test_loader, criterion, device)
23
24     # Store metrics
25     train_losses.append(train_loss)
26     val_losses.append(val_loss)
27     train_accuracies.append(train_accuracy)
28     val_accuracies.append(val_accuracy)
29
30     print(f"Epoch {epoch+1}/{num_epochs}: "
31           f"Train Loss={train_loss:.4f}, Train Accuracy={train_accuracy:.2f}%, "
32           f"Val Loss={val_loss:.4f}, Val Accuracy={val_accuracy:.2f}%")


    
```

```

1 import matplotlib.pyplot as plt
2
3 # Plot loss curve
4 plt.figure(figsize=(12, 5))
5 plt.subplot(1, 2, 1)
6 plt.plot(range(1, num_epochs+1), train_losses, label="Train Loss", marker='o')
7 plt.plot(range(1, num_epochs+1), val_losses, label="Validation Loss", marker='o')
8 plt.xlabel("Epochs")
9 plt.ylabel("Loss")
10 plt.title("Loss Curve")
11 plt.legend()
12
13 # Plot accuracy curve
14 plt.subplot(1, 2, 2)
15 plt.plot(range(1, num_epochs+1), train_accuracies, label="Train Accuracy", marker='o')
16 plt.plot(range(1, num_epochs+1), val_accuracies, label="Validation Accuracy", marker='o')
17 plt.xlabel("Epochs")
18 plt.ylabel("Accuracy (%)")
19 plt.title("Accuracy Curve")
20 plt.legend()
21
22 plt.show()
23
    
```



```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Function to display images with predicted labels
5 def show_predictions(model, dataloader, device, num_images=10):
6     model.eval() # Set to evaluation mode
7     images, labels = next(iter(dataloader)) # Get a batch
8     images, labels = images.to(device), labels.to(device)
9
10    with torch.no_grad(): # Disable gradient computation
11        outputs = model(images)
12        predictions = outputs.argmax(dim=1) # Get predicted class
13
14    # CIFAR-10 class names
15    classes = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
16               'Dog', 'Frog', 'Horse', 'Ship', 'Truck']
17
18    # Plot images with predictions
19    fig, axes = plt.subplots(2, 5, figsize=(10, 5))
20    for i, ax in enumerate(axes.flat[:num_images]):
21        img = images[i]
22        img = np.transpose(img.cpu().numpy(), (1, 2, 0)) # Convert to (H, W, C)
23
24        ax.imshow(img)
25        ax.set_title(f"Pred: {classes[predictions[i].item()]} \nTrue: {classes[labels[i].item()]}")
26        ax.axis("off")
27
28    plt.show()
29
30 # Show predictions
31 show_predictions(model, test_loader, device)
    
```

The labs included:

- ① Pytorch Basics
- ② Image Classification

From scratch (CustomModel, CNNModel)
- ③ Image Generation using AutoEncoder

image $\xrightarrow{\text{encoder}} \text{compressed image (dimensionality reduction)}$
 $\xrightarrow{\text{decoder}} \text{image generator (numbers} \rightarrow \text{image)}$
- ④ Custom Dataset class & Image Augmentation
- ⑤ Object detection
- ⑥ Image Segmentation

Image Search

→ might be:

- image Corralization

- image Reconstruction as example



(might use Segmentation)