

Deep learning using neural networks.

author: bhavesh patel

we will use images and its related tags to predict what the image is about.

we will use neural network for deep learning. Deep features is very interesting and see how it improves prediction.

```
In [1]: import graphlab

In [2]: # Limit number of worker processes. This preserves system memory, which prevents hosted notebooks from crashing.
graphlab.set_runtime_config('GRAPHLAB_DEFAULT_NUM_PYLAMBDAS', 4)





This non-commercial license of GraphLab Create for academic use is assigned to bhaveshhk8@gmail.com and will expire on October 17, 2017.
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: /tmp/graphlab_server_1481406886.log

In [5]: # now let's load the images from CIFAR-10 dataset, but its reduced to four categories: cat, bird, automobile, dog.
# it is already split into training dataset and test dataset.

image_train = graphlab.SFrame('image_train_data/')
image_test = graphlab.SFrame('image_test_data/')

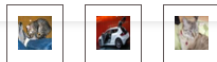
In [6]: # set output local to here.
graphlab.canvas.set_target('ipynb')

In [8]: # let's view the data.
image_train.show()
```

	image	label	deep_features	image_array
dtype: int	dtype: Image	dtype: str	dtype: array	dtype: array
num_unique (est.): 1,999	First 4 images: 	num_unique (est.): 4 num_undefined: 0 frequent items: automobile dog cat bird	num_unique (est.): 2,336,740 num_undefined: 0 min: 0 max: 15.345 median: 0 mean: 0.386 std: 0.905	num_unique (est.): 255 num_undefined: 0 min: 0 max: 255 median: 113 mean: 116.985 std: 64.312
num_undefined: 0			distribution of values (all sub-columns): 	distribution of values (all sub-columns): 
nin: 24				
nax: 49,970				
median: 23,969				
mean: 24,828.162				
std: 14,682.469				
distribution of values: 				

```
In [9]: # before we build model to predict, let's see what first three images are.
image_test[0:3]['image'].show()
```

All 3 images in <SArray>



```
In [10]: # its cat, car, cat. noted.
# to confirm, here are the labels.
image_test[0:3]['label']
```

```
Out[10]: dtype: str
Rows: 3
['cat', 'automobile', 'cat']
```

```
In [12]: # now let's build ML model to train classifier.
```

```
image_classifier_model = graphlab.logistic_classifier.create(image_train, target='label',
                                                             features=['image_array'])
```

PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.

You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Unless an appropriate regularization value is set, this model may not provide accurate predictions for a validation/test set.

Logistic regression:

Number of examples : 1910
Number of classes : 4
Number of feature columns : 1
Number of unpacked features : 3072
Number of coefficients : 9219

Starting L-BFGS

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	6	0.000016	2.888895	0.331414	0.389474
2	8	1.000000	3.842132	0.383246	0.421053
3	9	1.000000	4.346528	0.408377	0.378947
4	10	1.000000	4.912695	0.445550	0.368421
5	11	1.000000	5.510324	0.447120	0.368421
6	12	1.000000	6.038188	0.465969	0.431579
10	16	1.000000	8.001951	0.521990	0.526316

TERMINATED: Iteration limit reached.

This model may not be optimal. To improve it, consider increasing ``max_iterations``.

In [13]: # now use the model to predict.

```
image_classifier_model.predict(image_test[0:3])
```

Out[13]: dtype: str

Rows: 3
['bird', 'cat', 'bird']

In [14]: # well, that's horrible accuracy. :(
let's evaluate to find out.

```
image_classifier_model.evaluate(image_test)
```

Out[14]: {'accuracy': 0.48075, 'auc': 0.7235272916666664, 'confusion_matrix': Columns:
target_label str
predicted_label str
count int

Rows: 16

Data:

target_label	predicted_label	count
bird	dog	198
dog	cat	239
bird	automobile	112
automobile	automobile	607
cat	dog	303
dog	dog	431
dog	automobile	88
bird	bird	529
automobile	bird	118
bird	cat	161

[16 rows x 3 columns]

Note: Only the head of the SFrame is printed.

You can use `print_rows(num_rows=m, num_columns=n)` to print more rows and columns., 'f1_score': 0.4807160516374978,
'log_loss': 1.2065411828057908, 'precision': 0.48193676238170613, 'recall': 0.48075, 'roc_curve': Columns:

threshold float
fpr float
tpr float
p int
n int
class int

Rows: 400004

Data:

threshold	fpr	tpr	p	n	class
0.0	1.0	1.0	1000	3000	0
1e-05	1.0	1.0	1000	3000	0
2e-05	1.0	1.0	1000	3000	0
3e-05	1.0	1.0	1000	3000	0
4e-05	1.0	1.0	1000	3000	0
5e-05	1.0	1.0	1000	3000	0
6e-05	1.0	1.0	1000	3000	0
7e-05	1.0	1.0	1000	3000	0
8e-05	1.0	1.0	1000	3000	0
9e-05	1.0	1.0	1000	3000	0

```

+-----+-----+-----+-----+-----+
[400004 rows x 6 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.)

```

In [15]: `# ok so only 48% accuracy! Not good.`

In [16]: `# now let's use deep features. Borrow it!
first load the model.
deep_learning_model = graphlab.load_model('http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45`









Downloading http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45/dir_archive.ini to /var/tmp/graphlab-admin/6030/eelal6f0-618c-48c7-a5ec-80a8f621dcbe.ini

Downloading http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45/objects.bin to /var/tmp/graphlab-admin/6030/6e5de8fd-5303-4706-bddf-fc5cd6d192fc.bin

In [17]: `# now let's extract the features for our data based on this model.
image_train['bp_deep_features'] = deep_learning_model.extract_features(image_train)`

Images being resized.

In [18]: `image_train.show()`

image		label		deep_features		image_array		bp_deep_features	
dtype:	Image	dtype:	str	dtype:	array	dtype:	array	dtype:	array
First 4 images:		num_unique (est.):	4	num_unique (est.):	2,336,740	num_unique (est.):	255	num_unique (est.):	2,343,203
		num_undefined:	0	num_undefined:	0	num_undefined:	0	num_undefined:	0
frequent items:		min:	0	min:	0	min:	0	min:	0
		max:	15.345	max:	15.345	max:	255	max:	15.345
		median:	0	median:	0	median:	113	median:	0
		mean:	0.386	mean:	0.386	mean:	116.985	mean:	0.386
		std:	0.905	std:	0.905	std:	64.312	std:	0.905
		distribution of values (all sub-columns):		distribution of values (all sub-columns):		distribution of values (all sub-columns):		distribution of values (all sub-columns):	
									

In [20]: `# It took long time to process the new model. But finally it did!
Let's use this deep featuers, which are borrowed from other model.`

`deep_feature_model=graphlab.logistic_classifier.create(image_train,
features=['bp_deep_features'],
target='label')`

PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Unless an appropriate regularization value is set, this model may not provide accurate predictions for a validation/test set.

WARNING: Detected extremely low variance for feature(s) 'bp_deep_features' because all entries are nearly the same. Proceeding with model training using all features. If the model does not provide results of adequate quality, exclude the above mentioned feature(s) from the input dataset.

Logistic regression:

```

-----
Number of examples      : 1918
Number of classes       : 4
Number of feature columns : 1
Number of unpacked features : 4096
Number of coefficients   : 12291
Starting L-BFGS

```

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	5	0.000130	2.699571	0.757039	0.678161
2	9	0.250000	5.461219	0.765902	0.724138
3	10	0.250000	6.176370	0.773723	0.701149
4	11	0.250000	6.907332	0.778936	0.712644

5	12	0.250000	7.658626	0.789364	0.712644
6	13	0.250000	8.439804	0.800313	0.712644
7	14	0.250000	9.485471	0.819082	0.701149
8	15	0.250000	10.499534	0.842544	0.735632
9	16	0.250000	11.402847	0.873827	0.747126
10	17	0.250000	12.264933	0.895203	0.735632

```
+-----+-----+-----+-----+-----+-----+
```

TERMINATED: Iteration limit reached.

This model may not be optimal. To improve it, consider increasing `max_iterations`.

```
In [21]: deep_feature_model.predict(image_test[0:3])
```

```
Out[21]: dtype: str
Rows: 3
['cat', 'cat', 'cat']
```

```
In [22]: # ok good improvement. Let's compare all three.
# real values are: cat, car, cat
# our training data predicted it as: bird, cat, bird
# our deep feature model predicted, cat, cat, cat -> not bad, but I was hoping better!
```

```
In [24]: # now let's find accuracy of this model.
```

```
deep_feature_model.evaluate(image_test)
```

```
Out[24]: {'accuracy': 0.25, 'auc': 0.5, 'confusion_matrix': Columns:
target_label str
predicted_label str
count int
```

Rows: 4

Data:

target_label	predicted_label	count
dog	cat	1000
bird	cat	1000
automobile	cat	1000
cat	cat	1000

```
[4 rows x 3 columns], 'f1_score': 0.1, 'log_loss': 1.466788500595805, 'precision': 0.25, 'recall': 0.25, 'roc_curve': Columns:
```

```
threshold float
fpr float
tpr float
p int
n int
class int
```

Rows: 400004

Data:

threshold	fpr	tpr	p	n	class
0.0	1.0	1.0	1000	3000	0
1e-05	1.0	1.0	1000	3000	0
2e-05	1.0	1.0	1000	3000	0
3e-05	1.0	1.0	1000	3000	0
4e-05	1.0	1.0	1000	3000	0
5e-05	1.0	1.0	1000	3000	0
6e-05	1.0	1.0	1000	3000	0
7e-05	1.0	1.0	1000	3000	0
8e-05	1.0	1.0	1000	3000	0
9e-05	1.0	1.0	1000	3000	0

[400004 rows x 6 columns]

Note: Only the head of the SFrame is printed.

You can use `print_rows(num_rows=m, num_columns=n)` to print more rows and columns.)

```
In [25]: # only 25%. That's not good. That's because my computer didn't continue to iterate.
```

```
In [ ]: # let's use the deep features, which were part of the model and see what we get.
```

```
In [31]: deep_feature_precalculated_model = graphlab.logistic_classifier.create(image_train,
features=['deep_features'],
target='label')
```

PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Unless an appropriate regularization value is set, this model may not provide accurate predictions for a validation/test set.

WARNING: Detected extremely low variance for feature(s) 'deep_features' because all entries are nearly the same. Proceeding with model training using all features. If the model does not provide results of adequate quality, exclude the above mentioned feature(s) from the input dataset.

Logistic regression:

```
-----
```

Number of examples : 1889

Number of classes : 4

```
.. . . . .
```

```
Number of feature columns : 1
Number of unpacked features : 4096
Number of coefficients : 12291
Starting L-BFGS
```

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	5	0.000132	2.434330	0.741133	0.715517
2	9	0.250000	5.056429	0.772896	0.775862
3	10	0.250000	5.836083	0.775543	0.775862
4	11	0.250000	6.608528	0.779778	0.767241
5	12	0.250000	7.481547	0.790895	0.775862
6	13	0.250000	8.274079	0.801482	0.784483
7	14	0.250000	8.980571	0.824246	0.758621
8	15	0.250000	9.986029	0.838539	0.793103
9	16	0.250000	10.892873	0.852832	0.793103
10	17	0.250000	11.719468	0.872949	0.793103

TERMINATED: Iteration limit reached.

This model may not be optimal. To improve it, consider increasing `max_iterations`.

```
In [32]: deep_feature_precalculated_model.predict(image_test[0:3])
```

```
Out[32]: dtype: str
Rows: 3
['cat', 'automobile', 'cat']
```

```
In [ ]: # wow-> finally ML got it.
```

```
In [33]: # let's see accuracy of this model.
```

```
In [35]: deep_feature_precalculated_model.evaluate(image_test)
```

```
Out[35]: {'accuracy': 0.784, 'auc': 0.9384483749999979, 'confusion_matrix': Columns:
        target_label  str
        predicted_label str
        count      int
```

Rows: 16

Data:

target_label	predicted_label	count
automobile	cat	14
bird	dog	58
cat	bird	69
automobile	dog	7
cat	automobile	33
dog	bird	44
bird	cat	130
dog	automobile	20
dog	dog	716
cat	dog	222

[16 rows x 3 columns]

Note: Only the head of the SFrame is printed.

You can use `print_rows(num_rows=m, num_columns=n)` to print more rows and columns., 'f1_score': 0.7841899124404468, 'log_loss': 0.6112328671477912, 'precision': 0.78548595574037, 'recall': 0.784, 'roc_curve': Columns:

```
threshold      float
fpr            float
tpr            float
p              int
n              int
class          int
```

Rows: 400004

Data:

threshold	fpr	tpr	p	n	class
0.0	1.0	1.0	1000	3000	0
1e-05	0.966333333333	1.0	1000	3000	0
2e-05	0.954333333333	1.0	1000	3000	0
3e-05	0.946	1.0	1000	3000	0
4e-05	0.94	1.0	1000	3000	0
5e-05	0.931666666667	1.0	1000	3000	0
6e-05	0.928333333333	1.0	1000	3000	0
7e-05	0.925333333333	1.0	1000	3000	0
8e-05	0.919666666667	1.0	1000	3000	0
9e-05	0.918666666667	1.0	1000	3000	0

[400004 rows x 6 columns]

Note: Only the head of the SFrame is printed.

You can use `print_rows(num_rows=m, num_columns=n)` to print more rows and columns.)

```
In [36]: # wow 78% accuracy. That's very good.
```

```
To [ ]: # To summarize:
```

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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 10

```
#Model 1: Using limited set of data without using deep features from other model. Accuracy: 48%
#Model 2: My model with deep features, but not going through all iteration. Accuracy: 25% -> feeling bad.
#Model 3: Pre calculated deep feature model. Accuracy: 78% -> aka we need to have bigger computer for more iteration.
```