



```
1 # set tf 1.x for colab
2 %tensorflow_version 1.x
```

## Fine-tuning InceptionV3 for flowers classification

In this task you will fine-tune InceptionV3 architecture for flowers classification task.

InceptionV3 architecture (<https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html>): 

Flowers classification dataset (<http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html>) consists of 102 flower categories commonly occurring in the United Kingdom. Each class contains between 40 and 258 images: 

### Running on Google Colab

Copy all files of intro-to-dl-master in root\_path

```
1 ! shred -u setup_google_colab.py
2 ! wget https://raw.githubusercontent.com/hse-aml/intro-to-dl/master/setup_google_colab.
```

```
shred: setup_google_colab.py: failed to open for writing: No such file or directory
--2021-01-17 22:56:44-- https://raw.githubusercontent.com/hse-aml/intro-to-dl/master/
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 3636 (3.6K) [text/plain]
Saving to: 'setup_google_colab.py'
```

```
setup_google_colab. 100%[=====>] 3.55K --.-KB/s in 0s
```

```
2021-01-17 22:56:45 (36.9 MB/s) - 'setup_google_colab.py' saved [3636/3636]
```



```
1 import setup_google_colab
2 setup_google_colab.setup_week3()
```

```
*****
102flowers.tgz
*****
imagelabels.mat
*****
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
*****
cifar-10-batches-py.tar.gz
*****
mnist.npz
```

## ▼ Import stuff

```

1 root_path = '/content/gdrive/MyDrive/HSE_Introduction to Deep Learning'

1 import sys
2 sys.path.append(root_path)
3 import grading_utils
4 import download_utils

1 # !!! remember to clear session/graph if you rebuild your graph to avoid out-of-memory

1 download_utils.link_all_keras_resources()

1 import tensorflow as tf
2 import keras
3 from keras import backend as K
4 import numpy as np
5 %matplotlib inline
6 import matplotlib.pyplot as plt
7 print(tf.__version__)
8 print(keras.__version__)
9 import cv2 # for image processing
10 from sklearn.model_selection import train_test_split
11 import scipy.io
12 import os
13 import tarfile
14 import keras_utils
15 from keras_utils import reset_tf_session
16 import grading

    1.15.2
    2.3.1

```

## ▼ Fill in your Coursera token and email

To successfully submit your answers to our grader, please fill in your Coursera submission token and email

```

1 grader = grading.Grader(assignment_key="2v-uxpD7EeeMxQ6Fwsz5LA",
2                          all_parts=["wuwWC", "a4FK1", "qRsZ1"])

1 # token expires every 30 min
2 COURSERA_TOKEN = 'Rd1AjChnFdHTNvS8'      ### YOUR TOKEN HERE
3 COURSERA_EMAIL = 'knowtech94@gmail.com'   ### YOUR EMAIL HERE

```

## ▼ Load dataset

Dataset was downloaded for you, it takes 12 min and 400mb. Relevant links (just in case):

- <http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html>
- <http://www.robots.ox.ac.uk/~vgg/data/flowers/102/102flowers.tgz>
- <http://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat>

```
1 # we downloaded them for you, just link them here
2 download_utils.link_week_3_resources()
```

## ▼ Prepare images for model

```
1 # we will crop and resize input images to IMG_SIZE x IMG_SIZE
2 IMG_SIZE = 250

1 def decode_image_from_raw_bytes(raw_bytes):
2     img = cv2.imdecode(np.asarray(bytearray(raw_bytes), dtype=np.uint8), 1)
3     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4     return img
```

We will take a center crop from each image like this: 

```
1 def image_center_crop(img):
2     """
3     Makes a square center crop of an img, which is a [h, w, 3] numpy array.
4     Returns [min(h, w), min(h, w), 3] output with same width and height.
5     For cropping use numpy slicing.
6     """
7
8     #cropped_img = ### YOUR CODE HERE
9     w, h, c = img.shape
10
11     if w > h:
12         cropped_img = img[int((w-h)/2):int((w+h)/2), 0:h, 0:c]
13     elif h > w:
14         cropped_img = img[0:w, int((h-w)/2):int((h+w)/2), 0:c]
15     else:
16         cropped_img = img[0:w, 0:h, 0:c]
17
18     return cropped_img
19

1 def prepare_raw_bytes_for_model(raw_bytes, normalize_for_model=True):
2     img = decode_image_from_raw_bytes(raw_bytes) # decode image raw bytes to matrix
```

```

1  img = image_center_crop(img, (img.shape[0], img.shape[1])) # take squared center crop
3  img = cv2.resize(img, (IMG_SIZE, IMG_SIZE)) # resize for our model
4  if normalize_for_model:
5      img = img.astype("float32") # prepare for normalization
6      img = keras.applications.inception_v3.preprocess_input(img) # normalize for mo
8  return img

```

```

1 # reads bytes directly from tar by filename (slow, but ok for testing, takes ~6 sec)
2 def read_raw_from_tar(tar_fn, fn):
3     with tarfile.open(tar_fn) as f:
4         m = f.getmember(fn)
5         return f.extractfile(m).read()

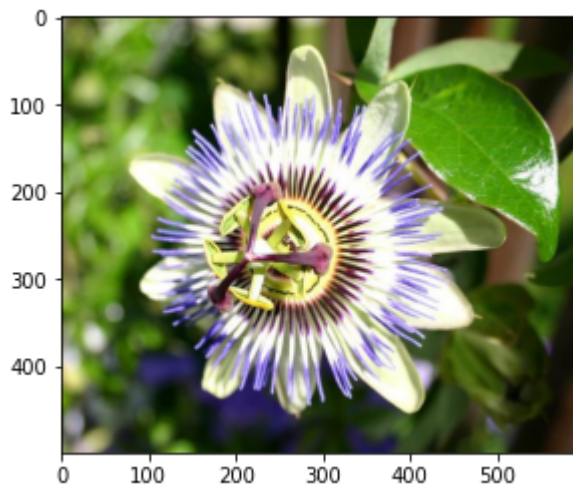
```

```

1 # test cropping
2 raw_bytes = read_raw_from_tar("102flowers.tgz", "jpg/image_00001.jpg")
3
4 img = decode_image_from_raw_bytes(raw_bytes)
5 print(img.shape)
6 plt.imshow(img)
7 plt.show()
8

```

(500, 591, 3)

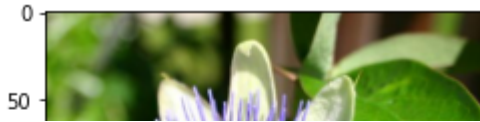


```

1 img = prepare_raw_bytes_for_model(raw_bytes, normalize_for_model=False)
2 print(img.shape)
3 plt.imshow(img)
4 plt.show()

```

(250, 250, 3)



```
1 ## GRADED PART, DO NOT CHANGE!
2 # Test image preparation for model
3 prepared_img = prepare_raw_bytes_for_model(read_raw_from_tar("102flowers.tgz", "jpg/ima
4 grader.set_answer("qRsZ1", list(prepared_img.shape) + [np.mean(prepared_img), np.std(pr
```



```
1 # you can make submission with answers so far to check yourself at this stage
2 grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

Submitted to Coursera platform. See results on assignment page!

## ▼ Prepare for training

```
1 # read all filenames and labels for them
2
3 # read filenames firectly from tar
4 def get_all_filenames(tar_fn):
5     with tarfile.open(tar_fn) as f:
6         return [m.name for m in f.getmembers() if m.isfile()]
7
8 all_files = sorted(get_all_filenames("102flowers.tgz")) # list all files in tar sorted
9 all_labels = scipy.io.loadmat('imagelabels.mat')['labels'][0] - 1 # read class labels
10 # all_files and all_labels are aligned now
11 N_CLASSES = len(np.unique(all_labels))
12 print(N_CLASSES)
```

102

```
1 # split into train/test
2 tr_files, te_files, tr_labels, te_labels = \
3     train_test_split(all_files, all_labels, test_size=0.2, random_state=42, stratify=all_labels)
```

```
1 # will yield raw image bytes from tar with corresponding label
2 def raw_generator_with_label_from_tar(tar_fn, files, labels):
3     label_by_fn = dict(zip(files, labels))
4     with tarfile.open(tar_fn) as f:
5         while True:
6             m = f.next()
7             if m is None:
8                 break
9             if m.name in label_by_fn:
10                 yield f.extractfile(m).read(), label_by_fn[m.name]
```

```
1 # batch generator
2 BATCH_SIZE = 32
```

```

3
4 def batch_generator(items, batch_size):
5     """
6     Implement batch generator that yields items in batches of size batch_size.
7     There's no need to shuffle input items, just chop them into batches.
8     Remember about the last batch that can be smaller than batch_size!
9     Input: any iterable (list, generator, ...). You should do `for item in items: ...`
10    In case of generator you can pass through your items only once!
11    Output: In output yield each batch as a list of items.
12    """
13
14    ### YOUR CODE HERE
15    batch_generator = []
16
17    for item in items:
18        batch_generator.append(item)
19        if len(batch_generator) == batch_size:
20            yield batch_generator
21            batch_generator = []
22    if batch_generator:
23        yield batch_generator
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

```

Submitted to Coursera platform. See results on assignment page!

```

1 def train_generator(files, labels):
2     while True: # so that Keras can loop through this as long as it wants
3         for batch in batch_generator(raw_generator_with_label_from_tar(
4             "102flowers.tgz", files, labels), BATCH_SIZE):
5             # prepare batch images
6             batch_imgs = []
7             batch_targets = []
8             for raw, label in batch:
9                 img = prepare_raw_bytes_for_model(raw)
10                batch_imgs.append(img)
11                batch_targets.append(label)
12            # stack images into 4D tensor [batch_size, img_size, img_size, 3]
13            batch_imgs = np.stack(batch_imgs, axis=0)
14            # convert targets into 2D tensor [batch_size, num_classes]
15            batch_targets = keras.utils.np_utils.to_categorical(batch_targets, N_CLASSES)
16            yield batch_imgs, batch_targets

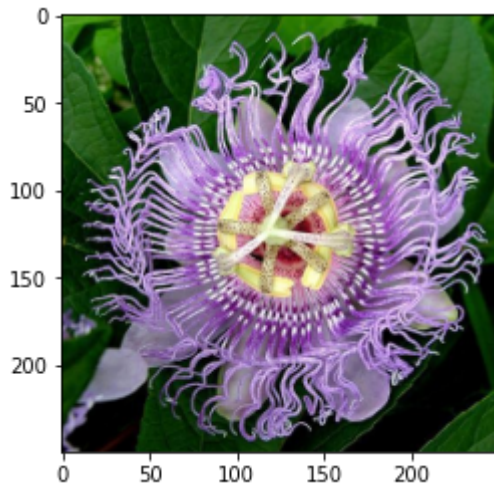
```

```

1 # test training generator
2 for _ in train_generator(tr_files, tr_labels):
3     print(_[0].shape, _[1].shape)
4     plt.imshow(np.clip(_[0][0] / 2. + 0.5, 0, 1))
5     break

```

```
(32, 250, 250, 3) (32, 102)
```



## ▼ Training

You cannot train such a huge architecture from scratch with such a small dataset.

But using fine-tuning of last layers of pre-trained network you can get a pretty good classifier very quickly.

```

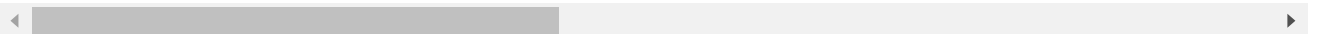
1 # remember to clear session if you start building graph from scratch!
2 s = reset_tf_session()
3 # don't call K.set_learning_phase() !!! (otherwise will enable dropout in train/test si

```

```
WARNING:tensorflow:From /content/gdrive/MyDrive/HSE_Introduction to Deep Learning/ker
```

```
WARNING:tensorflow:From /content/gdrive/MyDrive/HSE_Introduction to Deep Learning/ker
```

```
WARNING:tensorflow:From /content/gdrive/MyDrive/HSE_Introduction to Deep Learning/ker
```



```

1 def inception(use_imagenet=True):
2     # load pre-trained model graph, don't add final layer
3     model = keras.applications.InceptionV3(include_top=False, input_shape=(IMG_SIZE, IM
4                                         weights='imagenet' if use_imagenet else None)
5     # add global pooling just like in InceptionV3
6     new_output = keras.layers.GlobalAveragePooling2D()(model.output)
7     # add new dense layer for our labels
8     new_output = keras.layers.Dense(N_CLASSES, activation='softmax')(new_output)
9     model = keras.engine.training.Model(model.inputs, new_output)
10    return model

```

```
1 model = inception()
```

WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow\_core/python/ops/resol  
Instructions for updating:  
If using Keras pass \*\_constraint arguments to layers.  
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/keras/backend/tensorflow\_backend  
  
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/keras/backend/tensorflow\_backend

```
1 model.summary()
```

batch_normalization_87 (BatchNormalizatio	(None, 6, 6, 384)	1152	conv2d_87[0][0]
batch_normalization_91 (BatchNormalizatio	(None, 6, 6, 384)	1152	conv2d_91[0][0]
activation_87 (Activation)	(None, 6, 6, 384)	0	batch_normalization_91[0][0]
activation_91 (Activation)	(None, 6, 6, 384)	0	batch_normalization_91[0][0]
conv2d_88 (Conv2D)	(None, 6, 6, 384)	442368	activation_87[0][0]
conv2d_89 (Conv2D)	(None, 6, 6, 384)	442368	activation_87[0][0]
conv2d_92 (Conv2D)	(None, 6, 6, 384)	442368	activation_91[0][0]
conv2d_93 (Conv2D)	(None, 6, 6, 384)	442368	activation_91[0][0]
average_pooling2d_9 (AveragePooling2D)	(None, 6, 6, 2048)	0	mixed9[0][0]
conv2d_86 (Conv2D)	(None, 6, 6, 320)	655360	mixed9[0][0]
batch_normalization_88 (BatchNormalizatio	(None, 6, 6, 384)	1152	conv2d_88[0][0]
batch_normalization_89 (BatchNormalizatio	(None, 6, 6, 384)	1152	conv2d_89[0][0]
batch_normalization_92 (BatchNormalizatio	(None, 6, 6, 384)	1152	conv2d_92[0][0]
batch_normalization_93 (BatchNormalizatio	(None, 6, 6, 384)	1152	conv2d_93[0][0]
conv2d_94 (Conv2D)	(None, 6, 6, 192)	393216	average_pooling2d_9[0][0]
batch_normalization_86 (BatchNormalizatio	(None, 6, 6, 320)	960	conv2d_86[0][0]
activation_88 (Activation)	(None, 6, 6, 384)	0	batch_normalization_88[0][0]
activation_89 (Activation)	(None, 6, 6, 384)	0	batch_normalization_89[0][0]
activation_92 (Activation)	(None, 6, 6, 384)	0	batch_normalization_92[0][0]
activation_93 (Activation)	(None, 6, 6, 384)	0	batch_normalization_93[0][0]
batch_normalization_94 (BatchNormalizatio	(None, 6, 6, 192)	576	conv2d_94[0][0]
activation_86 (Activation)	(None, 6, 6, 320)	0	batch_normalization_86[0][0]
mixed9_1 (Concatenate)	(None, 6, 6, 768)	0	activation_88[0][0] activation_89[0][0]
concatenate_2 (Concatenate)	(None, 6, 6, 768)	0	activation_92[0][0] activation_93[0][0]



concatenate_2 (Concatenate)	(None, 6, 6, 168)	0	activation_94[0][0] activation_93[0][0]
activation_94 (Activation)	(None, 6, 6, 192)	0	batch_normalization
mixed10 (Concatenate)	(None, 6, 6, 2048)	0	activation_86[0][0] mixed9_1[0][0] concatenate_2[0][0] activation_94[0][0]
global_average_pooling2d_1 (Global Average Pooling)	(None, 2048)	0	mixed10[0][0]

```
1 # how many layers our model has
2 print(len(model.layers))
```

```
313
```

```
1 # set all layers trainable by default
2 for layer in model.layers:
3     layer.trainable = True
4     if isinstance(layer, keras.layers.BatchNormalization):
5         # we do aggressive exponential smoothing of batch norm
6         # parameters to faster adjust to our new dataset
7         layer.momentum = 0.9
8
9 # fix deep layers (fine-tuning only last 50)
10 for layer in model.layers[:-50]:
11     # fix all but batch norm layers, because we needed to update moving averages for a n
12     if not isinstance(layer, keras.layers.BatchNormalization):
13         layer.trainable = False
```

```
1 # compile new model
2 model.compile(
3     loss='categorical_crossentropy', # we train 102-way classification
4     optimizer=keras.optimizers.adamax(lr=1e-2), # we can take big lr here because we f
5     metrics=['accuracy'] # report accuracy during training
6 )
```

```
1 # we will save model checkpoints to continue training in case of kernel death
2 model_filename = 'flowers.{0:03d}.hdf5'
3 last_finished_epoch = None
4
5 ##### uncomment below to continue training from model checkpoint
6 ##### fill `last_finished_epoch` with your latest finished epoch
7 # from keras.models import load_model
8 # s = reset_tf_session()
9 # last_finished_epoch = 10
10 # model = load_model(model_filename.format(last_finished_epoch))
```

Training takes **2 hours**. You're aiming for ~0.93 validation accuracy.

```

1 # fine tune for 2 epochs (full passes through all training data)
2 # we make 2*8 epochs, where epoch is 1/8 of our training data to see progress more ofte
3 model.fit_generator(
4     train_generator(tr_files, tr_labels),
5     steps_per_epoch=len(tr_files) // BATCH_SIZE // 8,
6     epochs= 10,
7     validation_data=train_generator(te_files, te_labels),
8     validation_steps=len(te_files) // BATCH_SIZE // 4,
9     callbacks=[keras_utils.TqdmProgressCallback(),
10               keras_utils.ModelSaveCallback(model_filename)],
11     verbose=0,
12     initial_epoch=last_finished_epoch or 0
13 )

```

WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/keras/backend/tensorflow\_backend

Epoch 1/10

\*\*\*\*\*

loss: 4.4671; accuracy: 0.1904; val\_loss: 21.5944; val\_accuracy: 0.1432

Model saved in flowers.000.hdf5

Epoch 2/10

\*\*\*\*\*

loss: 2.5581; accuracy: 0.4232; val\_loss: 6.8255; val\_accuracy: 0.3698

Model saved in flowers.001.hdf5

Epoch 3/10

\*\*\*\*\*

loss: 1.5520; accuracy: 0.6190; val\_loss: 3.0175; val\_accuracy: 0.5339

Model saved in flowers.002.hdf5

Epoch 4/10

\*\*\*\*\*

loss: 1.0951; accuracy: 0.7233; val\_loss: 0.5756; val\_accuracy: 0.7292

Model saved in flowers.003.hdf5

Epoch 5/10

\*\*\*\*\*

loss: 0.7477; accuracy: 0.8038; val\_loss: 1.2126; val\_accuracy: 0.8128

Model saved in flowers.004.hdf5

Epoch 6/10

\*\*\*\*\*

loss: 0.6746; accuracy: 0.8394; val\_loss: 1.0167; val\_accuracy: 0.8802

Model saved in flowers.005.hdf5

Epoch 7/10

\*\*\*\*\*

loss: 0.5160; accuracy: 0.8741; val\_loss: 0.1965; val\_accuracy: 0.9167

Model saved in flowers.006.hdf5

Epoch 8/10

\*\*\*\*\*

loss: 0.4151; accuracy: 0.8800; val\_loss: 0.2613; val\_accuracy: 0.8828

Model saved in flowers.007.hdf5

Epoch 9/10

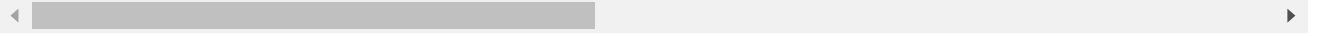
\*\*\*\*\*

```
loss: 0.2868; accuracy: 0.9304; val_loss: 0.2643; val_accuracy: 0.9022  
Model saved in flowers.008.hdf5
```

```
Epoch 10/10
```

```
*****
```

```
loss: 0.2178; accuracy: 0.9549; val_loss: 0.1252; val_accuracy: 0.9271  
Model saved in flowers.009.hdf5  
<keras.callbacks.callbacks.History at 0x7fbea1169eb8>
```



```
1 ## GRADED PART, DO NOT CHANGE!  
2 # Accuracy on validation set  
3 test_accuracy = model.evaluate_generator(  
4     train_generator(te_files, te_labels),  
5     len(te_files) // BATCH_SIZE // 2  
6 )[1]  
7 grader.set_answer("wuwWC", test_accuracy)  
8 print(test_accuracy)
```

```
0.9137499928474426
```

```
1 # you can make submission with answers so far to check yourself at this stage  
2 grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

Submitted to Coursera platform. See results on assignment page!

That's it! Congratulations!

What you've done:

- prepared images for the model
- implemented your own batch generator
- fine-tuned the pre-trained model

