PYTHON PROFILING TOOLS

Dr. Oscar Castro, Dr. Pierrick Pochelu

MANUAL PROFILING

- In a Jupyter notebook
- For profiling the cell computing time :
- %%time
- For profiling the cell memory:
  !pip install memory_profiler # installing
  %load_ext memory_profiler # activating
  %%memit # put everywhere we want to measure the memory

==> Measure the notebook mem. consumption, and the cell increment (memory consum. due to the cell).

==> Result are noisy due to interpreter optimization

Also:

Import time

st=time.time()

<your-code>

print(« Enlapsed time : », time.time()-st)

⇒ Advantage full control of what is measured

⇒ Disadvantage, when the project is large,

where put the timers ? Everywhere ???

When put every :

1) timers slowdowns the code

2) injecting “timers” impact the #lines and readbility

MEMORY MANAGEMENT IN PYTHON

- « High-water » phenomenon:
  - Long running Python jobs that consume a lot of memory while running may not return that memory to the operating system until the process actually terminates, even if everything is garbage collected properly.

OVERVIEW OF PYTHON PROFILER TECHNOLOGIES

Python introduce « decorator mechanism » allowing to choose easily which function to profile

Does not exist in Python. Python interpreter provides « hook » for plugging profilers. Thus, no profiler need to inject additional code in developper code. Python profilers use « event based approach » instead.

Four kinds of profiler :

- **Event based** : data collected when events occurs (entry/exit function, allocate/free memory).
  - Profiler may overhead
- **Statistical** : data collected periodically. The profiler does not give a overhead, data profiling are only sampled.
- **Instrumented manually** : developper put keyword to select codes to profiles (time.time(),%%time, ...)
- ~~**Instrumented automaticaly source level**: profiler code is « injected » automatically in the developper source code to collect data.~~

Profilers can offer different features :

- **Memory usage**. Different scale : (process, file, object, line).
- **Time usage**. Different scale : (process, function, line).
- **Hits count** : Count number of time line/function are reached.

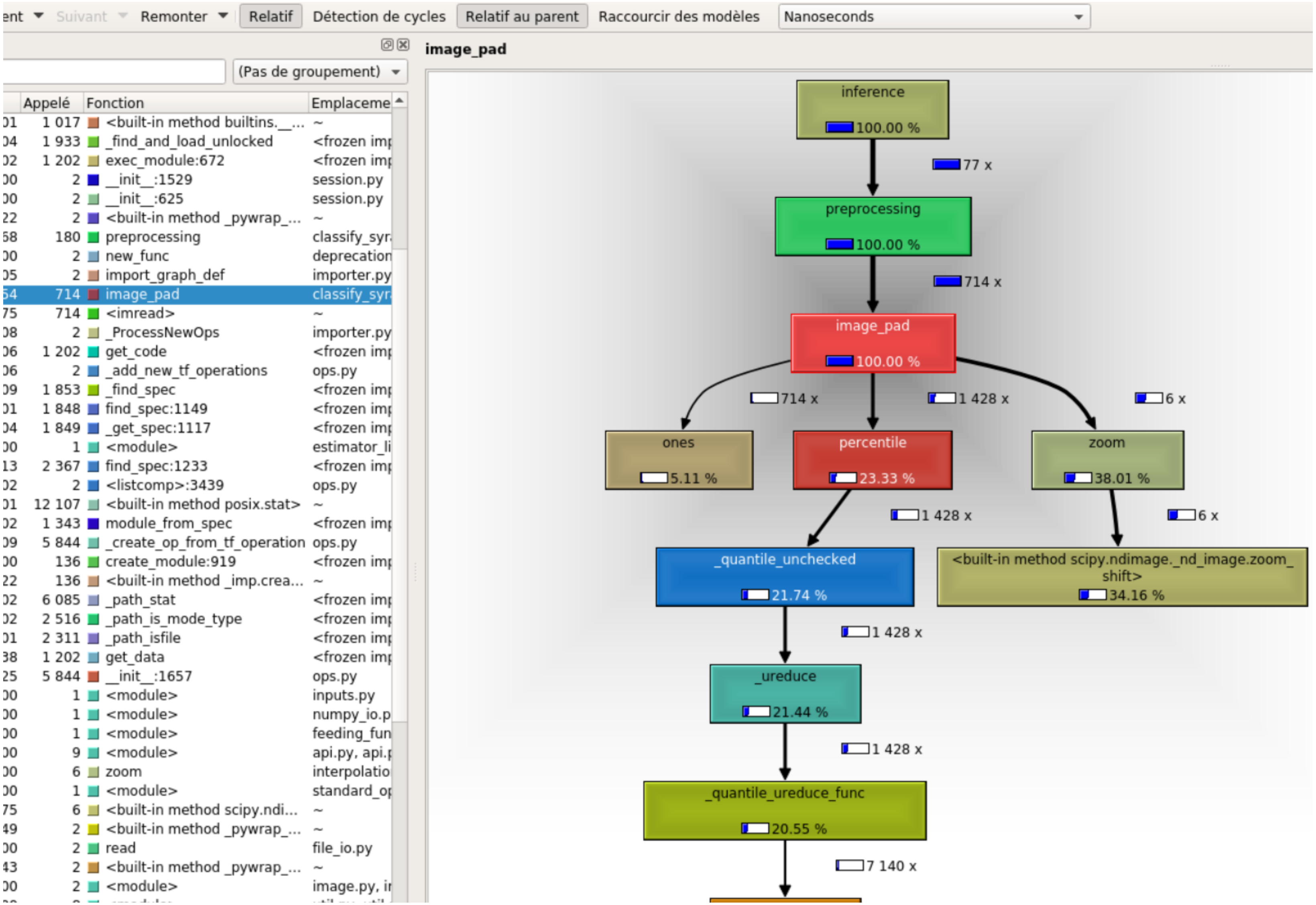
PROFILER OVERVIEW

Tested	name	kind	features							output and plotting	doc	comment	
			time prof.		hits count		memory prof.						
			funct.	line	funct.	line	all	obj.	line				
X	vprof	event based	X	X		X	X	X		display: flame, code heatmap, mem curv.	https://github.com/nvdv/vprof		memory and time
	nylas-perftools	statistical	X				X			display: flame, memory curve	https://github.com/nylas/nylas-perftools	can be used in production	
	vtune (intel)	all		X			X			GUI	https://software.intel.com/en-us/articles/profiling-python-with-intel-vtune-amplifier-a-covariance-demonstration	MPI compatible	
X	pympler	instru. manu.						X		string	https://pythonhosted.org/Pympler/	give objects size, track obj. Lifetime	memory only
	trace_malloc	instru. manu.					X		X	print	https://docs.python.org/3.4/library/tracemalloc.html		
X	memory_profiler	in. ma. or stat					X		X	global : mem curve, mem curv; func:print	https://github.com/pythonprofilers/memory_profiler	can be used globally or on function with @profile decorator	
	profile	event based	X		X					pyprof format	https://docs.python.org/3/library/profile.html		time only
X	cprofile	event based	X		X					pyprof format	https://docs.python.org/3/library/profile.html	lighter than cprofile	
	pycallgraph		X		X					graphviz format	http://pycallgraph.slowchop.com/		
	py-spy	event based	X							display : flame, top-like	https://github.com/benfred/py-spy	asynchronous to be used in production	
	pyflame	statistical	X							display: flame	https://github.com/uber/pyflame		
X	pprofile	ev. ba. or stat.		X		X				cachegrind for functions; line profiling: print	https://github.com/vpelletier/pprofile	pure python profiler so heavy	
	timeit	instru. manu.		X						python decimal number	https://docs.python.org/3/library/timeit.html		
	line_profiler	instru. manu.		X		X				pyprof format	https://github.com/rkern/line_profiler	pure python profiler so heavy	
	pycounters	instru. manu.	X							log in output file	https://pycounters.readthedocs.io/	Collect live metrics in production phase	
	Py-heat	event based	X							heatmap image	https://github.com/csurfer/pyheat/tree/master		
X	tensorflow profiler		-	-	-	-	-	-	-	log files	https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/profiler/README.md	profile tensorflow operations	tensorflow

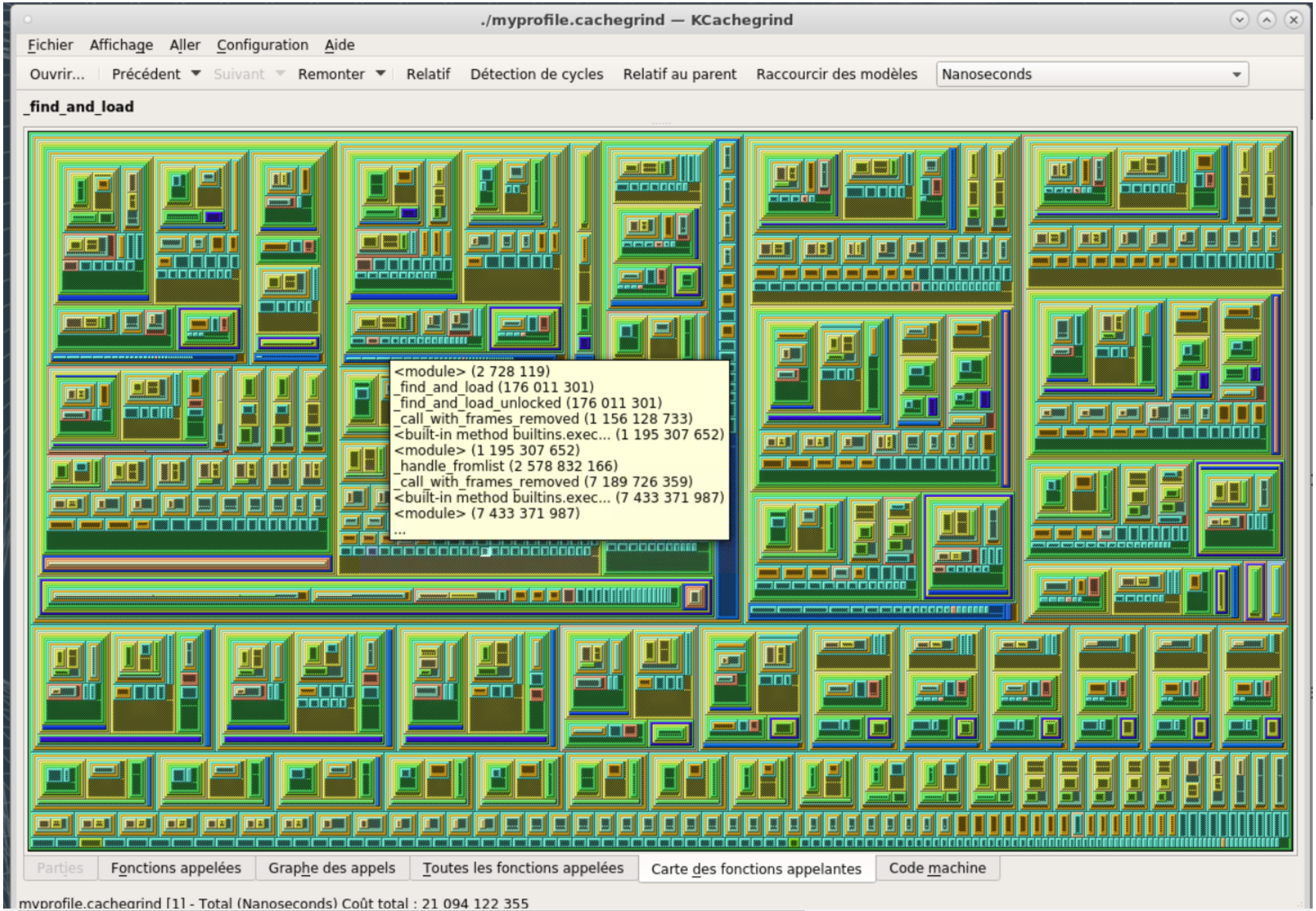
OTHER PROFILER TOOLS

	name	type	output	doc
converter (from pyprof to cachegrind format)	pyprof2calltree	python module	kcachegrind format	https://pypi.org/project/pyprof2calltree/
profiling visualisation (cachegrind format)	snakeViz	python module	display : flame, sunbust, func. Time table	https://jiffyclub.github.io/snakeviz/
	Kcachegrind	software	display : graph call, funct. Time table, func. Box	http://www.vrplumber.com/programming/runsnakerun/
framework to display pyprof info	pstats	python module	print	http://effbot.org/librarybook/pstats.htm

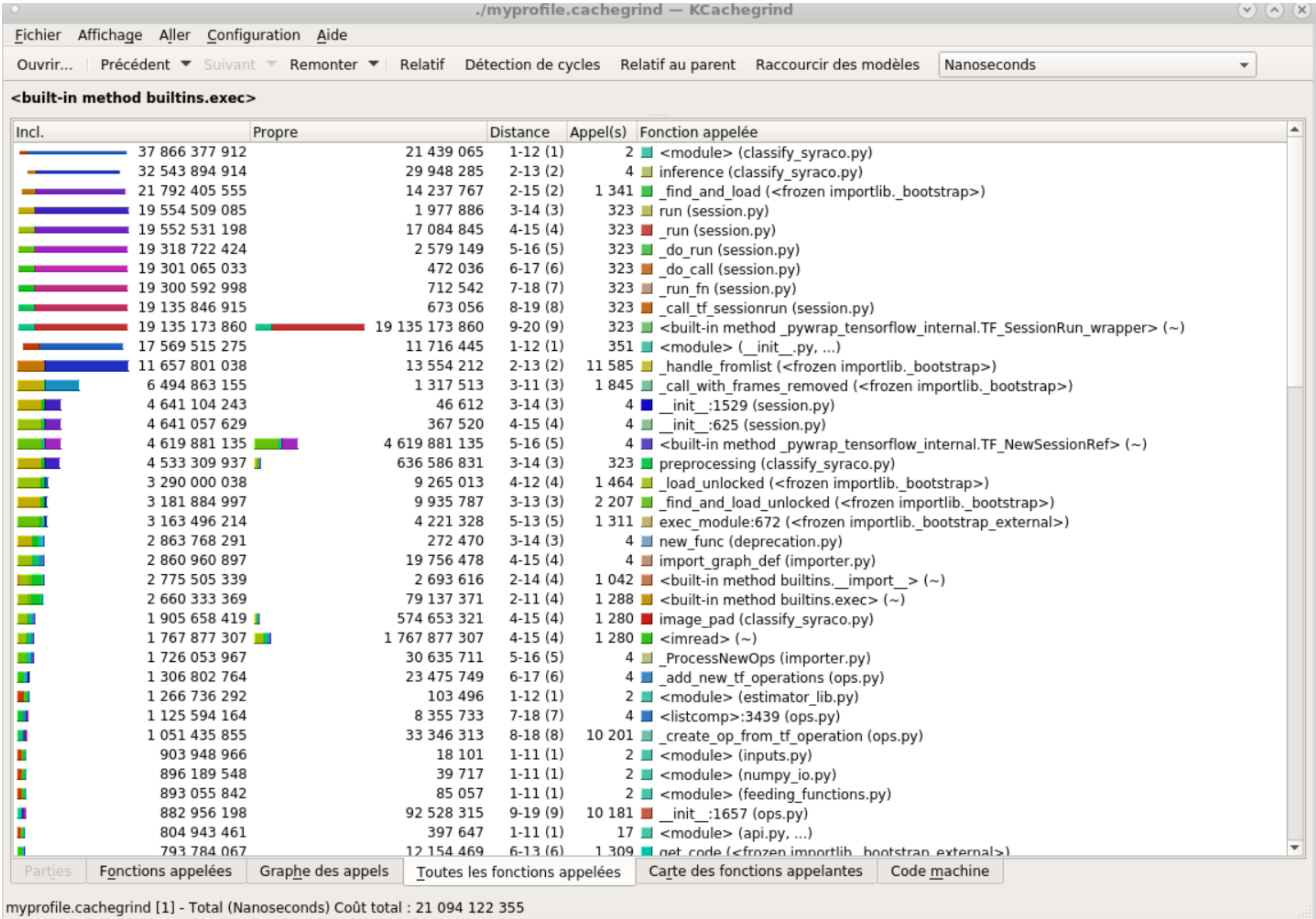
CPROFILE+KCACHEGRIND (SLIDES 1/3)



CPROFILE+KCACHEGRIND (SLIDES 2/3)



CPROFILE+KCACHEGRIND (SLIDES 3/3)



./myprofile.cachegrind — KCachegrind

Fichier Affichage Aller Configuration Aide

Ouvrir... Précédent Suivant Remonter Relatif Détection de cycles Relatif au parent Raccourcir des modèles Nanoseconds

<built-in method builtins.exec>

Incl.	Propre	Distance	Appel(s)	Fonction appelée
37 866 377 912	21 439 065	1-12 (1)	2	<module> (classify_syraco.py)
32 543 894 914	29 948 285	2-13 (2)	4	inference (classify_syraco.py)
21 792 405 555	14 237 767	2-15 (2)	1 341	_find_and_load (<frozen importlib._bootstrap>)
19 554 509 085	1 977 886	3-14 (3)	323	run (session.py)
19 552 531 198	17 084 845	4-15 (4)	323	_run (session.py)
19 318 722 424	2 579 149	5-16 (5)	323	_do_run (session.py)
19 301 065 033	472 036	6-17 (6)	323	_do_call (session.py)
19 300 592 998	712 542	7-18 (7)	323	_run_fn (session.py)
19 135 846 915	673 056	8-19 (8)	323	_call_tf_sessionrun (session.py)
19 135 173 860	19 135 173 860	9-20 (9)	323	<built-in method _pywrap_tensorflow_internal.TF_SessionRun_wrapper> (~)
17 569 515 275	11 716 445	1-12 (1)	351	<module> (__init__.py, ...)
11 657 801 038	13 554 212	2-13 (2)	11 585	_handle_fromlist (<frozen importlib._bootstrap>)
6 494 863 155	1 317 513	3-11 (3)	1 845	_call_with_frames_removed (<frozen importlib._bootstrap>)
4 641 104 243	46 612	3-14 (3)	4	__init__:1529 (session.py)
4 641 057 629	367 520	4-15 (4)	4	__init__:625 (session.py)
4 619 881 135	4 619 881 135	5-16 (5)	4	<built-in method _pywrap_tensorflow_internal.TF_NewSessionRef> (~)
4 533 309 937	636 586 831	3-14 (3)	323	preprocessing (classify_syraco.py)
3 290 000 038	9 265 013	4-12 (4)	1 464	_load_unlocked (<frozen importlib._bootstrap>)
3 181 884 997	9 935 787	3-13 (3)	2 207	_find_and_load_unlocked (<frozen importlib._bootstrap>)
3 163 496 214	4 221 328	5-13 (5)	1 311	exec_module:672 (<frozen importlib._bootstrap_external>)
2 863 768 291	272 470	3-14 (3)	4	new_func (deprecation.py)
2 860 960 897	19 756 478	4-15 (4)	4	import_graph_def (importer.py)
2 775 505 339	2 693 616	2-14 (4)	1 042	<built-in method builtins.__import__> (~)
2 660 333 369	79 137 371	2-11 (4)	1 288	<built-in method builtins.exec> (~)
1 905 658 419	574 653 321	4-15 (4)	1 280	image_pad (classify_syraco.py)
1 767 877 307	1 767 877 307	4-15 (4)	1 280	<imread> (~)
1 726 053 967	30 635 711	5-16 (5)	4	_ProcessNewOps (importer.py)
1 306 802 764	23 475 749	6-17 (6)	4	_add_new_tf_operations (ops.py)
1 266 736 292	103 496	1-12 (1)	2	<module> (estimator_lib.py)
1 125 594 164	8 355 733	7-18 (7)	4	<listcomp>:3439 (ops.py)
1 051 435 855	33 346 313	8-18 (8)	10 201	_create_op_from_tf_operation (ops.py)
903 948 966	18 101	1-11 (1)	2	<module> (inputs.py)
896 189 548	39 717	1-11 (1)	2	<module> (numpy_io.py)
893 055 842	85 057	1-11 (1)	2	<module> (feeding_functions.py)
882 956 198	92 528 315	9-19 (9)	10 181	__init__:1657 (ops.py)
804 943 461	397 647	1-11 (1)	17	<module> (api.py, ...)
793 784 067	12 154 469	6-13 (6)	1 309	get_code (<frozen importlib._bootstrap_external>)

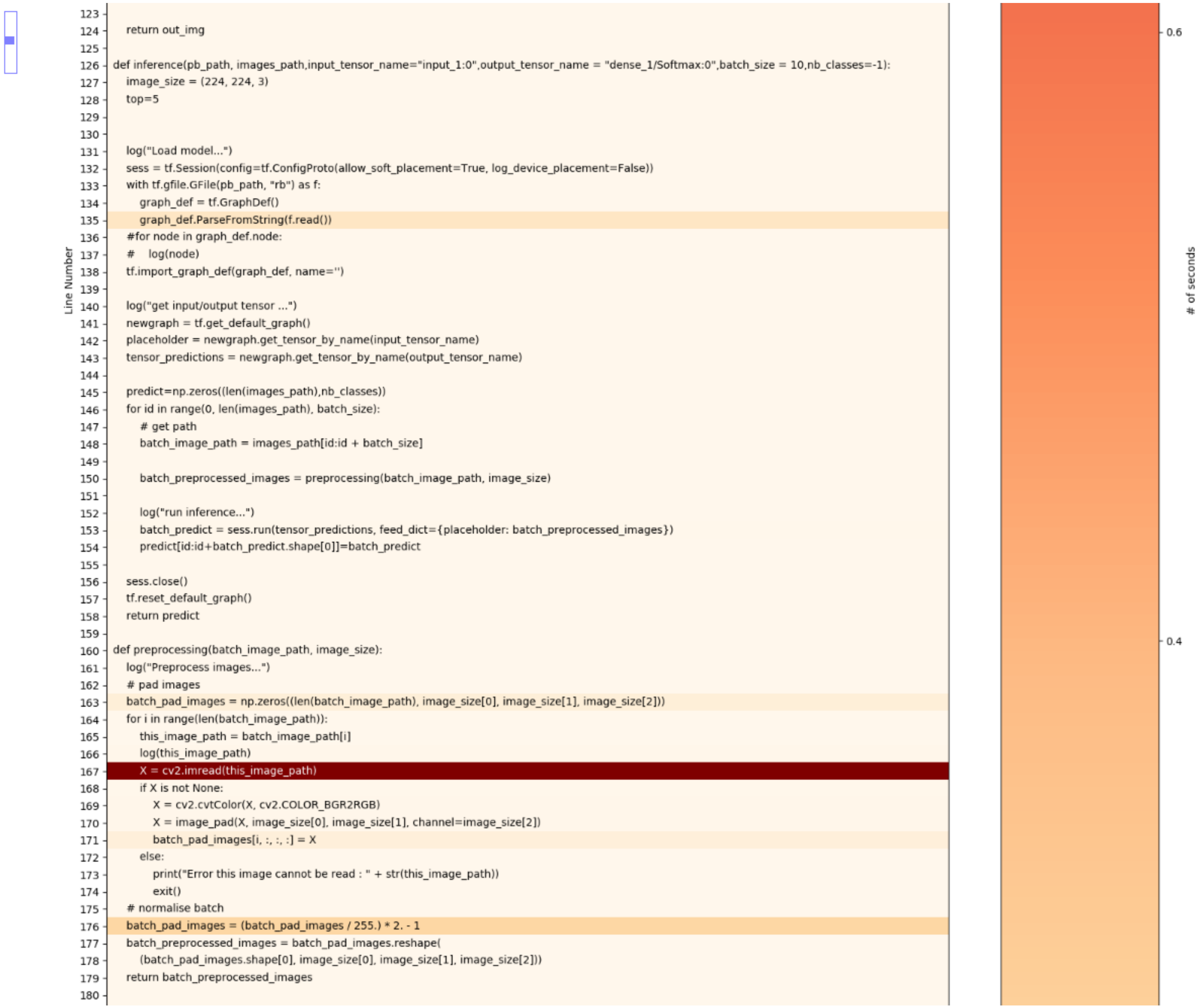
Parties | Fonctions appelées | Graphe des appels | Toutes les fonctions appelées | Carte des fonctions appelantes | Code machine

myprofile.cachegrind [1] - Total (Nanoseconds) Coût total : 21 094 122 355

PPROFILE

```
Command line: syraco_deployment/classify_syraco.py syraco_deployment//deep_model/fon_pol_model.pb syraco_deployment//deep_model/morpho_pol_model.pb
syraco_deployment//deep_model/fon_pol_class_name.npy syraco_deployment//deep_model/morpho_pol_class_name.npy
syraco_deployment//Test_Proto-COLOR-A-0000v0n0/Test_Proto-COLOR-A-0000v0n0-06_POL/ 4
Total duration: 70.0431s
File: syraco_deployment/classify_syraco.py
File duration: 2.4714s (3.53%)
Line #|      Hits|         Time| Time per hit|      %|Source code
------+----------+-------------+-------------+-------+-----------
     1|         0|            0|            0|  0.00%|
     2|         2|  9.65595e-05|  4.82798e-05|  0.00%|VERBOSE=True
     3|         1|  7.86781e-06|  7.86781e-06|  0.00%|DELETE_IMG=False
     4|         1|  1.23978e-05|  1.23978e-05|  0.00%|import time
     5|         1|  2.76566e-05|  2.76566e-05|  0.00%|import sys
     6|         1|  1.04904e-05|  1.04904e-05|  0.00%|import os
     7|         1|  9.77516e-06|  9.77516e-06|  0.00%|import argparse
     8|         1|  5.48363e-05|  5.48363e-05|  0.00%|import cv2
(call)|         1|      1.29204|      1.29204|  1.84%|# <frozen importlib._bootstrap>:966 _find_and_load
     9|         1|  1.14441e-05|  1.14441e-05|  0.00%|import numpy as np
    10|         1|  5.79357e-05|  5.79357e-05|  0.00%|import tensorflow as tf
(call)|         1|      25.2203|      25.2203| 36.01%|# <frozen importlib._bootstrap>:966 _find_and_load
    11|         1|  6.65188e-05|  6.65188e-05|  0.00%|from scipy.ndimage import zoom
(call)|         1|  7.82013e-05|  7.82013e-05|  0.00%|# <frozen importlib._bootstrap>:997 _handle_fromlist
    12|         0|            0|            0|  0.00%|
    13|         0|            0|            0|  0.00%|
    14|      1089|   0.00442934|  4.06735e-06|  0.01%|def log(txt):
    15|         0|            0|            0|  0.00%|    global VERBOSE
    16|      1088|   0.00468659|  4.30753e-06|  0.01%|    if VERBOSE:
    17|      1088|    0.0249534|  2.29351e-05|  0.04%|        print(txt)
    18|         0|            0|            0|  0.00%|
    19|         1|  8.60691e-05|  8.60691e-05|  0.00%|from memory_profiler import profile
(call)|         1|     0.294606|     0.294606|  0.42%|# <frozen importlib._bootstrap>:966 _find_and_load
(call)|         1|  4.05312e-05|  4.05312e-05|  0.00%|# <frozen importlib._bootstrap>:997 _handle_fromlist
    20|         2|   2.0504e-05|   1.0252e-05|  0.00%|def get_files(folder_image_path):
    21|         1|  4.52995e-06|  4.52995e-06|  0.00%|    list_files=[]
    22|       364|   0.00229287|   6.2991e-06|  0.00%|    for f in os.listdir(folder_image_path):
    23|       363|     0.004318|  1.18953e-05|  0.01%|        path=os.path.join(folder_image_path, f)
(call)|       363|    0.0231106|  6.36656e-05|  0.03%|# /data/appli_PITSI/users/pochelu/syraco_dl//env/install/Python-3.6.7/lib/python3.6/posixpath.py:75
join
    24|       363|   0.00149465|  4.11748e-06|  0.00%|        if path.endswith(('.tif','.tiff','.TIF','.TIFF')):
    25|       357|   0.00134945|  3.77997e-06|  0.00%|            list_files.append(path)
    26|         1|  3.57628e-06|  3.57628e-06|  0.00%|    return list_files
    27|         0|            0|            0|  0.00%|
    28|         0|            0|            0|  0.00%|
    29|       309|   0.00121617|  3.93584e-06|  0.00%|def not_available_line(predict_img,top,NA_string="N/A"):
    30|       308|    0.0039475|  1.28165e-05|  0.01%|    nb_score = np.min((predict_img.shape[0], top))
(call)|       308|    0.0273321|  8.87405e-05|  0.04%|#
/data/appli_PITSI/users/pochelu/syraco_dl/env/install/Python-3.6.7/lib/python3.6/site-packages/numpy/core/fromnumeric.py:2337 amin
    31|       308|   0.00129962|  4.21954e-06|  0.00%|    predict_img_txt = ""
    32|      1848|   0.00706983|  3.82566e-06|  0.01%|    for i in range(nb_score):
    33|      1540|   0.00632644|  4.10808e-06|  0.01%|        predict_img_txt += "\t" + NA_string + "\t" + NA_string
    34|       308|   0.00111604|   3.6235e-06|  0.00%|    return predict_img_txt
    35|         0|            0|            0|  0.00%|
    36|         0|            0|            0|  0.00%|
    37|         2|  1.95503e-05|  9.77516e-06|  0.00%|def from_score_to_txt(list_predict_fon, list_predict_morpho, fon_class_name, morpho_class_name,
    38|         0|            0|            0|  0.00%|                      top, list_images,enable_morpho_classif):
    39|       407|   0.00153971|  3.78306e-06|  0.00%|    def img_from_score_to_txt(predict, class_name, top):
    40|       406|   0.00555468|  1.36815e-05|  0.01%|        nb_score = np.min((predict.shape[0], top))
(call)|       406|    0.0361755|  8.91022e-05|  0.05%|#
```

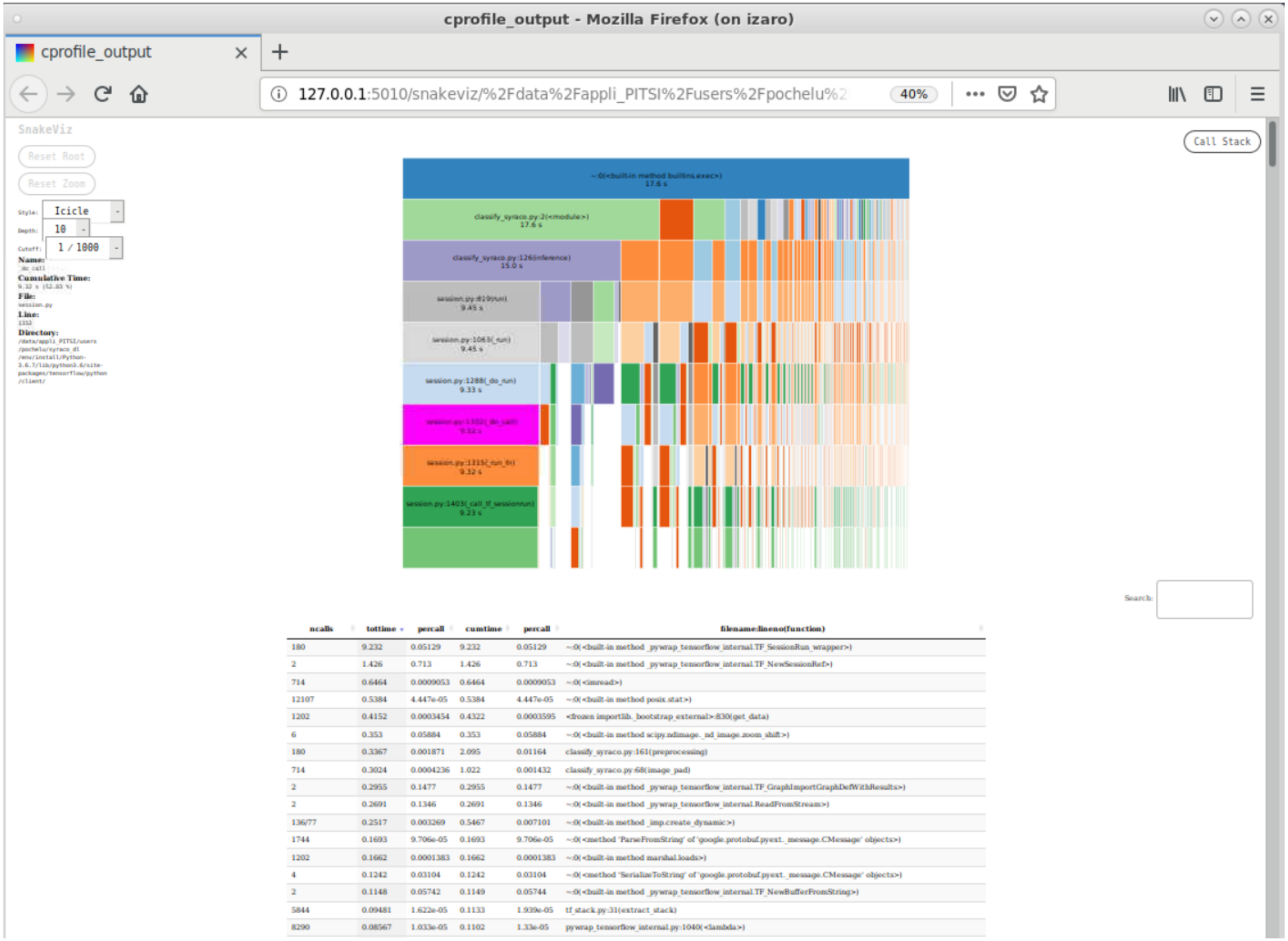
PYHEAT


Line Number
return out_img
def inference(pb_path, images_path,input_tensor_name="input_1:0",output_tensor_name = "dense_1/Softmax:0",batch_size = 10,nb_classes=-1):
image_size = (224, 224, 3)
top=5
log("Load model...")
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True, log_device_placement=False))
with tf.gfile.GFile(pb_path, "rb") as f:
graph_def = tf.GraphDef()
graph_def.ParseFromString(f.read())
#for node in graph_def.node:
# log(node)
tf.import_graph_def(graph_def, name='')
log("get input/output tensor ...")
newgraph = tf.get_default_graph()
placeholder = newgraph.get_tensor_by_name(input_tensor_name)
tensor_predictions = newgraph.get_tensor_by_name(output_tensor_name)
predict=np.zeros((len(images_path),nb_classes))
for id in range(0, len(images_path), batch_size):
# get path
batch_image_path = images_path[id:id + batch_size]
batch_preprocessed_images = preprocessing(batch_image_path, image_size)
log("run inference...")
batch_predict = sess.run(tensor_predictions, feed_dict={placeholder: batch_preprocessed_images})
predict[id:id+batch_predict.shape[0]]=batch_predict
sess.close()
tf.reset_default_graph()
return predict
def preprocessing(batch_image_path, image_size):
log("Preprocess images...")
# pad images
batch_pad_images = np.zeros((len(batch_image_path), image_size[0], image_size[1], image_size[2]))
for i in range(len(batch_image_path)):
this_image_path = batch_image_path[i]
log(this_image_path)
X = cv2.imread(this_image_path)
if X is not None:
X = cv2.cvtColor(X, cv2.COLOR_BGR2RGB)
X = image_pad(X, image_size[0], image_size[1], channel=image_size[2])
batch_pad_images[i, :, :, :] = X
else:
print("Error this image cannot be read : " + str(this_image_path))
exit()
# normalise batch
batch_pad_images = (batch_pad_images / 255.) * 2. - 1
batch_preprocessed_images = batch_pad_images.reshape(
(batch_pad_images.shape[0], image_size[0], image_size[1], image_size[2]))
return batch_preprocessed_images
0.6
0.4
# of seconds

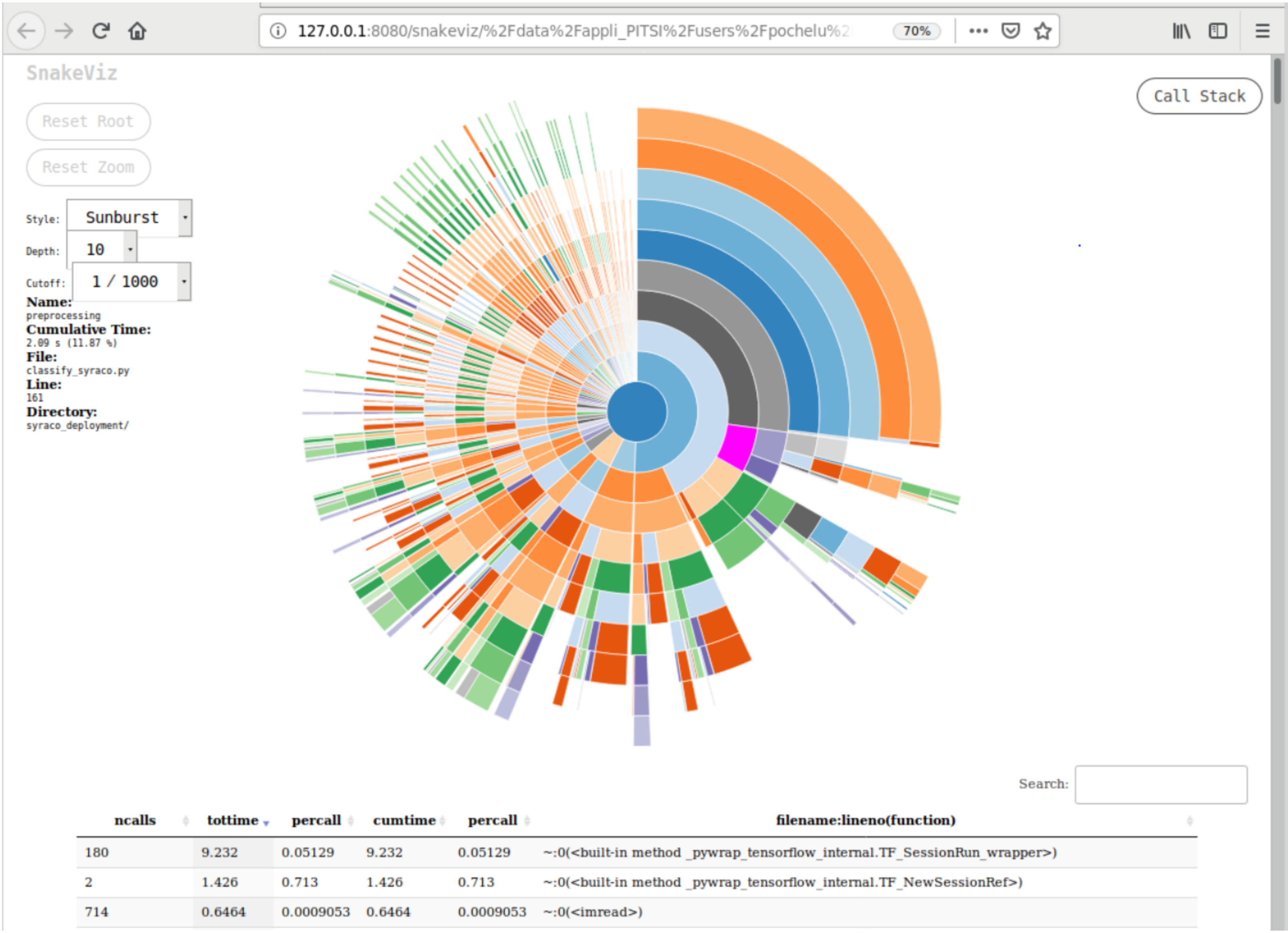
SNAKEVIZ (SLIDES 1/3)

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
180	17.39	0.09661	17.39	0.09661	~:0(<built-in method _pywrap_tensorflow_internal.TF_SessionRun_wrapper>)
714	0.6617	0.0009267	0.6617	0.0009267	~:0(<imread>)
12098	0.5189	4.289e-05	0.5189	4.289e-05	~:0(<built-in method posix.stat>)
2	0.5001	0.2501	0.5001	0.2501	~:0(<built-in method _pywrap_tensorflow_internal.TF_NewSessionRef>)
6	0.4143	0.06905	0.4143	0.06905	~:0(<built-in method scipy.ndimage._nd_image.zoom_shift>)
180	0.4027	0.002237	2.535	0.01408	classify_syraco.py:161(preprocessing)
1202	0.3923	0.0003264	0.4095	0.0003406	<frozen importlib._bootstrap_external>:830(get_data)
714	0.3784	0.00053	1.259	0.001764	classify_syraco.py:68(image_pad)
2	0.3171	0.1586	0.3171	0.1586	~:0(<built-in method _pywrap_tensorflow_internal.TF_GraphImportGraphDefWithResults>)
2	0.2959	0.148	0.2959	0.148	~:0(<built-in method _pywrap_tensorflow_internal.ReadFromStream>)
136/77	0.2431	0.003157	0.5298	0.00688	~:0(<built-in method _imp.create_dynamic>)
1744	0.2013	0.0001154	0.2013	0.0001154	~:0(<method 'ParseFromString' of 'google.protobuf.pyext._message.CMessage' objects>)
933	0.1843	0.0001975	0.1843	0.0001975	~:0(<built-in method numpy.core.multiarray.zeros>)
1202	0.1685	0.0001402	0.1685	0.0001402	~:0(<built-in method marshal.loads>)
4	0.1377	0.03442	0.1377	0.03442	~:0(<method 'SerializeToString' of 'google.protobuf.pyext._message.CMessage' objects>)
2	0.1202	0.06009	0.1202	0.06012	~:0(<built-in method _pywrap_tensorflow_internal.TF_NewBufferFromString>)
5844	0.1085	1.857e-05	0.1296	2.217e-05	tf_stack.py:31(extract_stack)
8290	0.09339	1.127e-05	0.1226	1.478e-05	pywrap_tensorflow_internal.py:1040(<lambda>)
5341	0.0929	1.739e-05	0.0929	1.739e-05	~:0(<built-in method numpy.core.multiarray.array>)
180	0.08665	0.0004814	0.08665	0.0004814	~:0(<built-in method _pywrap_tensorflow_internal.ExtendSession>)
1454/1	0.08345	0.08345	25.64	25.64	~:0(<built-in method builtins.exec>)
721	0.07459	0.0001035	0.07459	0.0001035	~:0(<built-in method numpy.core.multiarray.copyto>)
257	0.07434	0.0002892	0.07434	0.0002892	~:0(<built-in method posix.listdir>)

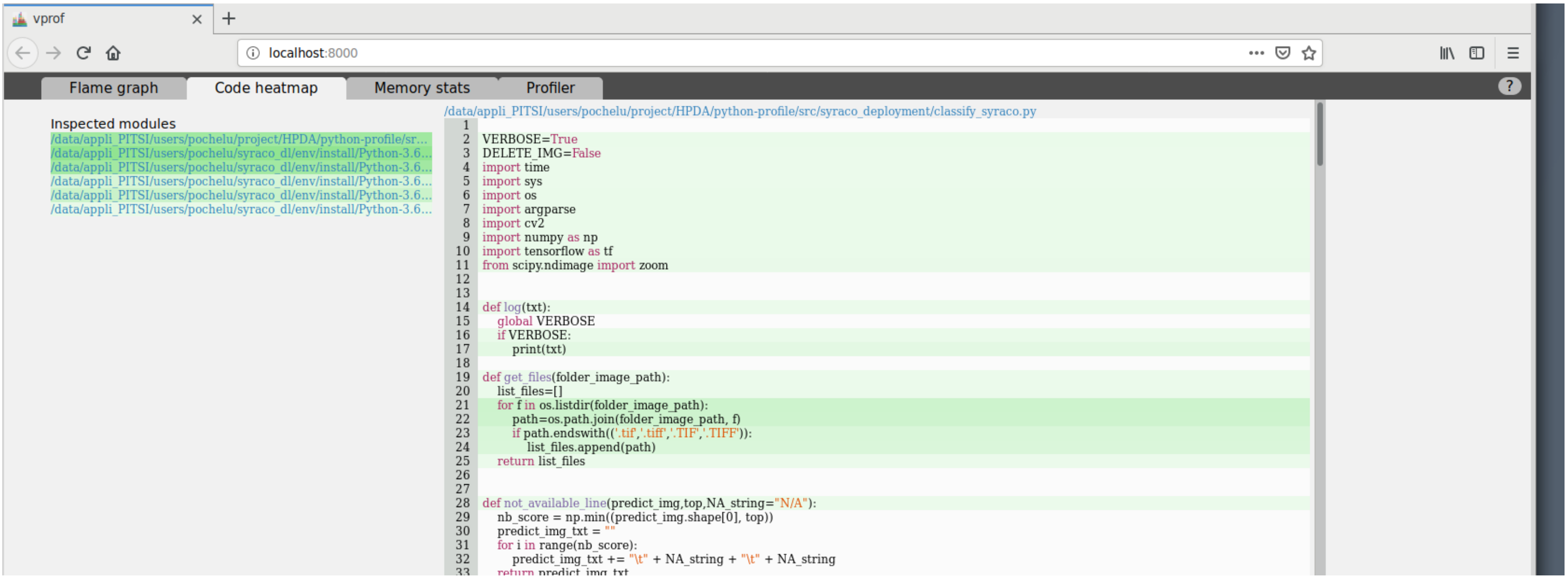
SNAKEVIZ (SLIDES 2/3)



SNAKEVIZ (SLIDES 3/3)

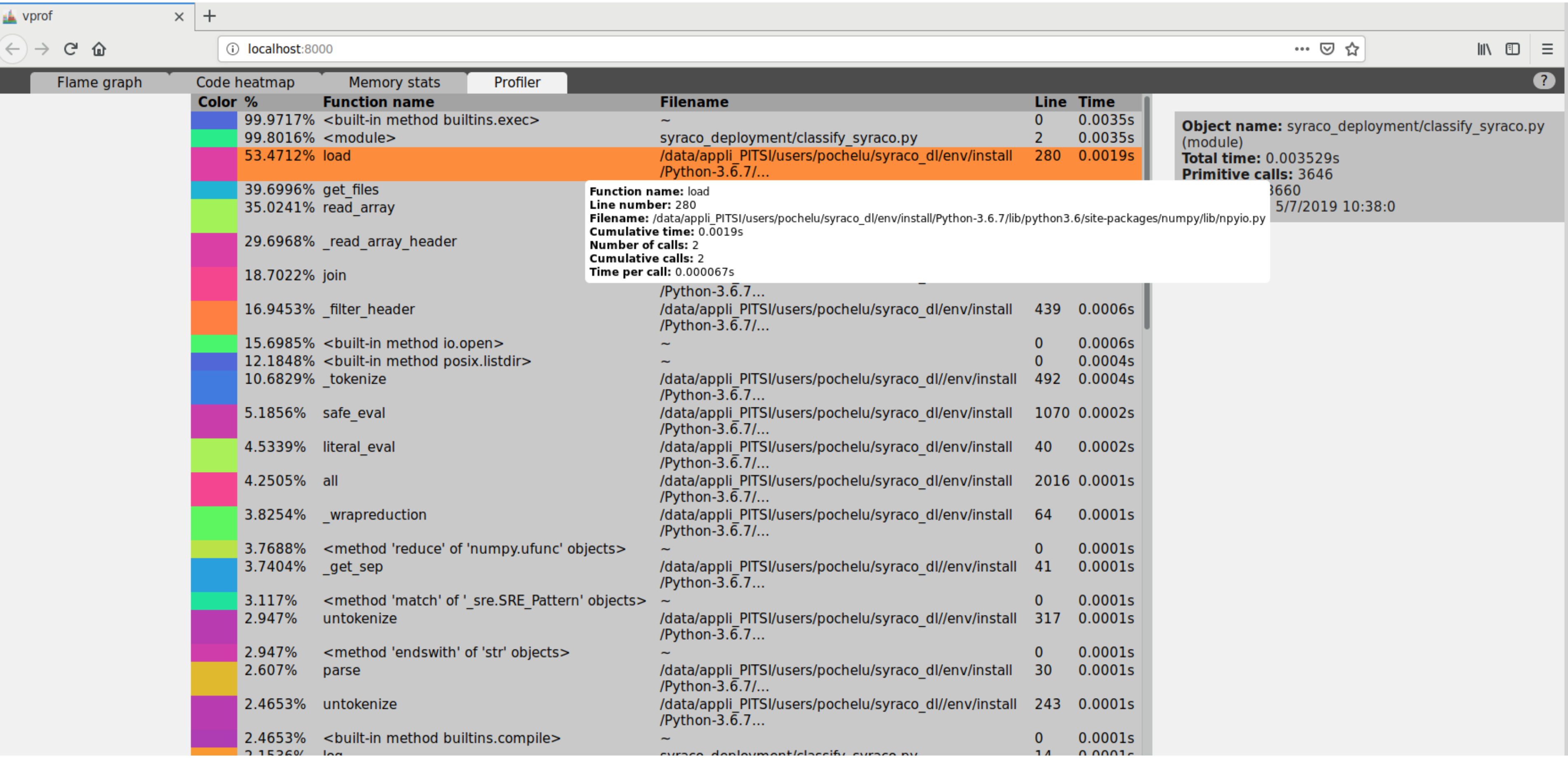


VPROF (SLIDES 1/3)



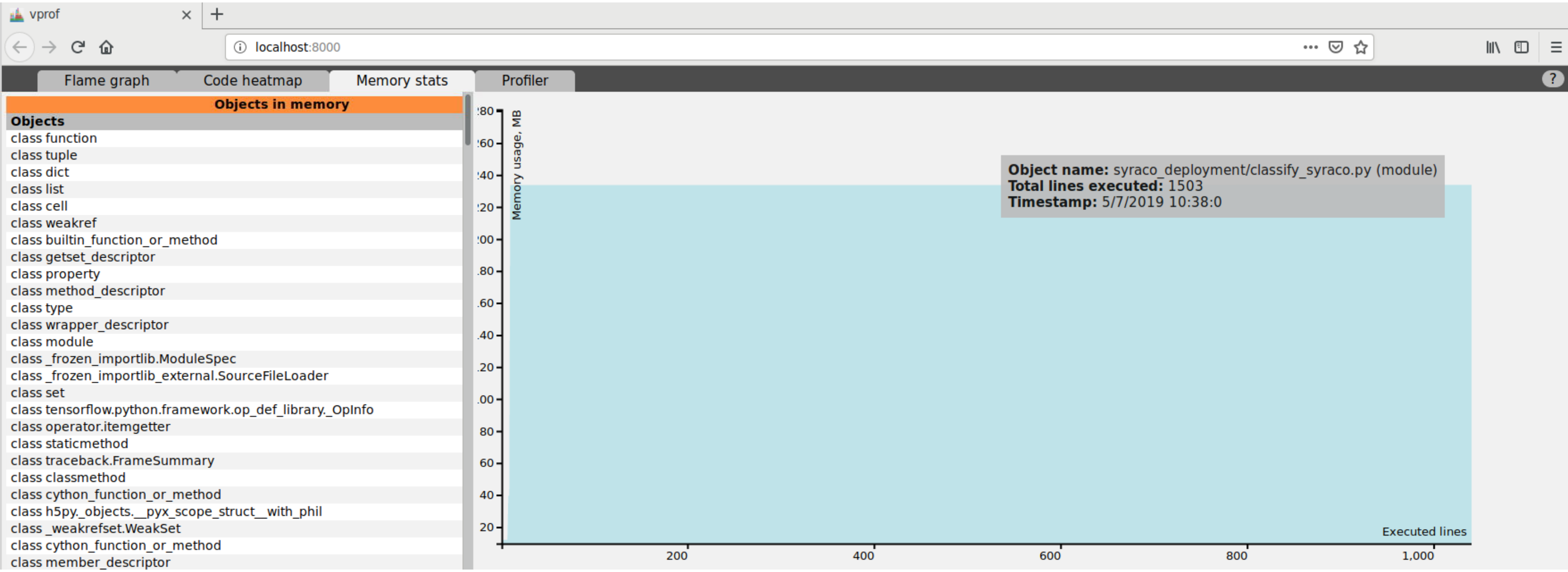
- Warning : Tensorflow run line « block » vprof

VPROF (SLIDES 2/3)



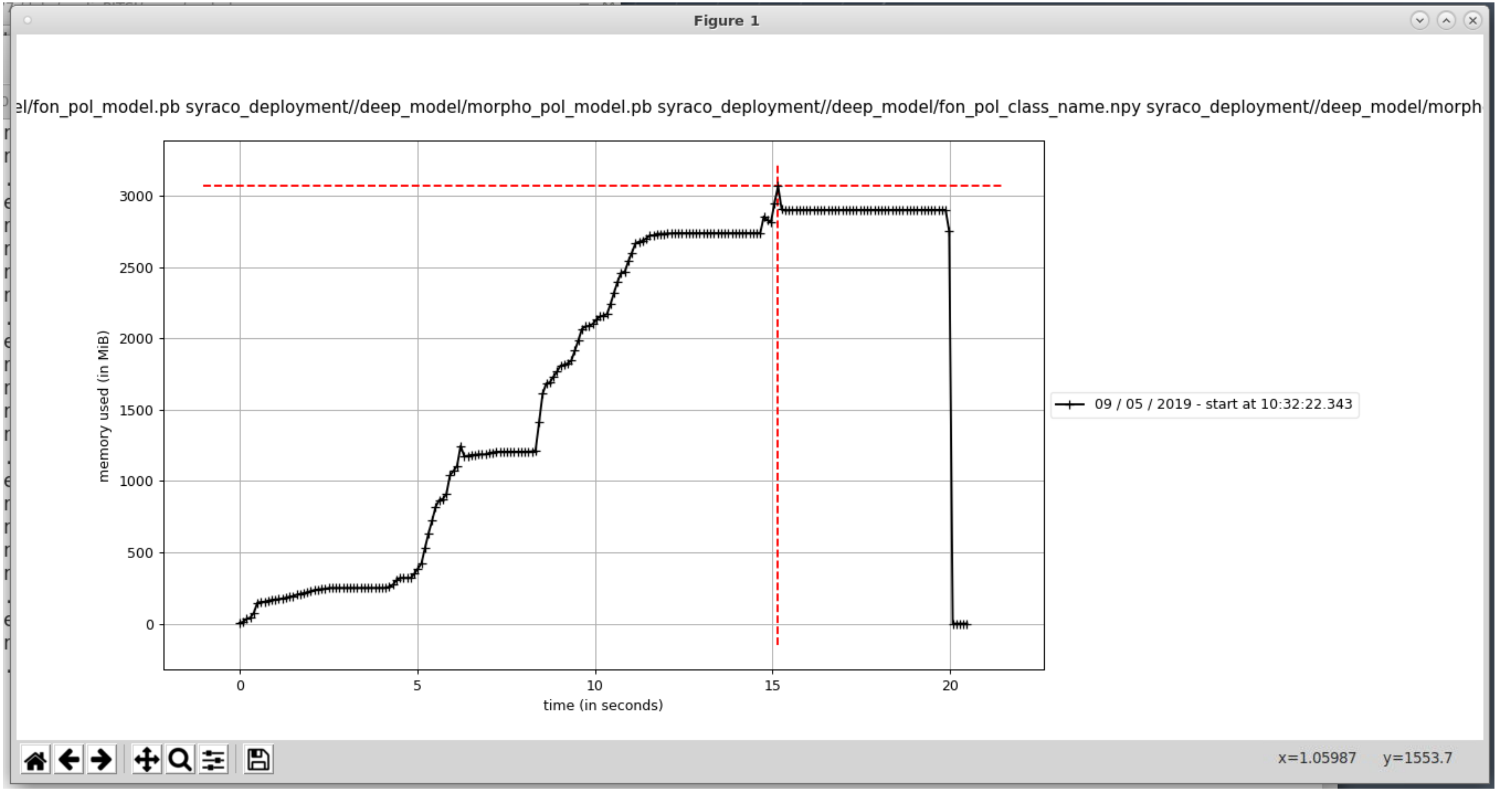
- Warning : Tensorflow run line « block » vprof

VPROF (SLIDES 3/3)



- Warning : Tensorflow run line « block » vprof

MEMORY_PROFILER (SLIDES 1/2)



MEMORY_PROFILER (SLIDES 2/2)

```
Line #    Mem usage    Increment   Line Contents
================================================
   126    243.2 MiB    243.2 MiB   @profile
   127                             def inference(pb_path, images_path,input_tensor_name="input_1:0",output_tensor_name = "dense_1/Softmax:0",batch_size =
   10,nb_classes=-1):
   128    243.2 MiB      0.0 MiB       image_size = (224, 224, 3)
   129    243.2 MiB      0.0 MiB       top=5
   130
   131
   132    243.2 MiB      0.0 MiB       log("Load model...")
   133    883.6 MiB    640.4 MiB       sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True, log_device_placement=False))
   134    883.6 MiB      0.0 MiB       with tf.gfile.GFile(pb_path, "rb") as f:
   135    883.6 MiB      0.0 MiB           graph_def = tf.GraphDef()
   136    997.4 MiB    113.8 MiB           graph_def.ParseFromString(f.read())
   137                                 #for node in graph_def.node:
   138                                 #    log(node)
   139   1186.6 MiB    189.3 MiB       tf.import_graph_def(graph_def, name='')
   140
   141   1186.6 MiB      0.0 MiB       log("get input/output tensor ...")
   142   1186.6 MiB      0.0 MiB       newgraph = tf.get_default_graph()
   143   1186.6 MiB      0.0 MiB       placeholder = newgraph.get_tensor_by_name(input_tensor_name)
   144   1186.6 MiB      0.0 MiB       tensor_predictions = newgraph.get_tensor_by_name(output_tensor_name)
   145
   146   1186.6 MiB      0.0 MiB       predict=np.zeros((len(images_path),nb_classes))
   147   2741.3 MiB      0.0 MiB       for id in range(0, len(images_path), batch_size):
   148                                     # get path
   149   2741.3 MiB      0.0 MiB           batch_image_path = images_path[id:id + batch_size]
   150
   151   2741.3 MiB      8.9 MiB           batch_preprocessed_images = preprocessing(batch_image_path, image_size)
   152
   153   2741.3 MiB      0.0 MiB           log("run inference...")
   154   2741.3 MiB   1538.0 MiB           batch_predict = sess.run(tensor_predictions, feed_dict={placeholder: batch_preprocessed_images})
   155   2741.3 MiB      0.0 MiB           predict[id:id+batch_predict.shape[0]]=batch_predict
   156
   157   2741.5 MiB      0.2 MiB       sess.close()
   158   2741.5 MiB      0.0 MiB       tf.reset_default_graph()
   159   2741.5 MiB      0.0 MiB       return predict
```

PYMPLER

```
                                                  types |   # objects |   total size
======================================================= | =========== | ============
                                           <class 'list |       16727 |      1.54 MB
                                            <class 'str |       17726 |      1.35 MB
                                  <class 'numpy.ndarray |           4 |    265.19 KB
                                            <class 'int |        3279 |     89.70 KB
                                           <class 'dict |          15 |      5.55 KB
                                        <class 'weakref |          55 |      4.30 KB
                                           <class 'type |           0 |      2.43 KB
                             <class 'wrapper_descriptor |          20 |      1.56 KB
                              <class 'method_descriptor |          17 |      1.20 KB
                        <class 'collections.OrderedDict |           2 |    736     B
                                            <class 'set |           2 |    448     B
                     <class 'builtin_function_or_method |           2 |    144     B
                                  function (store_info) |           1 |    136     B
                                     function (_remove) |           1 |    136     B
  <class 'tensorflow.python.eager.context._EagerTensorCache |           2 |    112     B
```

TENSORFLOW PROFILER (SLIDES 1/2)

```
Doc:
op: The nodes are operation kernel type, such as MatMul, Conv2D. Graph nodes belonging to the same type are aggregated together.
requested bytes: The memory requested by the operation, accumulatively.
total execution time: Sum of accelerator execution time and cpu execution time.
cpu execution time: The time from the start to the end of the operation. It's the sum of actual cpu run time plus the time that it spends waiting if par
computation is launched asynchronously.
accelerator execution time: Time spent executing on the accelerator. This is normally measured by the actual hardware library.
occurrence: The number of times it occurs

Profile:
node name | requested bytes | total execution time | accelerator execution time | cpu execution time | op occurrence (run|defined)
Conv2D                    40069.14MB (100.00%, 99.40%),      1.63sec (100.00%, 95.56%),     150.07ms (100.00%, 96.73%),      1.48sec (100.00%,
95.46%),      94|94
Sub                                 0B (0.00%, 0.00%),        44.57ms (4.44%, 2.62%),          275us (3.27%, 0.18%),         44.28ms (4.54%,
2.87%),      94|96
FusedBatchNorm                  97.23MB (0.60%, 0.24%),        7.27ms (1.82%, 0.43%),         1.39ms (3.09%, 0.90%),          5.86ms (1.68%,
0.38%),      94|94
Mul                             72.19KB (0.36%, 0.00%),        5.28ms (1.39%, 0.31%),         1.18ms (2.19%, 0.76%),          4.07ms (1.30%, 0.26%),
188|194
Add                             72.19KB (0.36%, 0.00%),        5.16ms (1.08%, 0.30%),         1.19ms (1.43%, 0.76%),          3.93ms (1.04%, 0.25%),
188|192
Const                           95.63MB (0.36%, 0.24%),        5.05ms (0.77%, 0.30%),           0us (0.67%, 0.00%),          5.05ms (0.78%, 0.33%),
394|781
Rsqrt                               0B (0.00%, 0.00%),         1.78ms (0.48%, 0.10%),          249us (0.67%, 0.16%),          1.52ms (0.46%,
0.10%),      94|94
Switch                              0B (0.00%, 0.00%),         1.75ms (0.37%, 0.10%),           0us (0.51%, 0.00%),          1.75ms (0.36%, 0.11%),
473|570
Relu                                0B (0.00%, 0.00%),         1.68ms (0.27%, 0.10%),          288us (0.51%, 0.19%),          1.38ms (0.24%,
0.09%),      64|95
ConcatV2                        21.62MB (0.12%, 0.05%),         916us (0.17%, 0.05%),          170us (0.32%, 0.11%),           745us (0.15%,
0.05%),      11|15
AvgPool                         17.59MB (0.06%, 0.04%),         545us (0.12%, 0.03%),          178us (0.21%, 0.11%),           364us (0.11%,
0.02%),       9|9
Merge                             384B (0.02%, 0.00%),          497us (0.08%, 0.03%),            0us (0.10%, 0.00%),           497us (0.08%,
0.03%),      96|96
MatMul                          16.64KB (0.02%, 0.00%),         292us (0.05%, 0.02%),           52us (0.10%, 0.03%),           239us (0.05%,
0.02%),       2|2
MaxPool                          5.88MB (0.02%, 0.01%),         238us (0.04%, 0.01%),           71us (0.06%, 0.05%),           167us (0.03%,
0.01%),       4|4
conv2d_1/convolution-0-TransposeNHWCToNCHW-LayoutOptimizer       2.41MB (0.01%, 0.01%),        184us (0.02%, 0.01%),         12us (0.02%,
0.01%),          172us (0.02%, 0.01%),        1|1
Softmax                          1.02KB (0.00%, 0.00%),         129us (0.01%, 0.01%),           11us (0.01%, 0.01%),           117us (0.01%,
0.01%),       1|1
Mean                            32.77KB (0.00%, 0.00%),          91us (0.01%, 0.01%),            6us (0.00%, 0.00%),            84us (0.01%,
0.01%),       1|1
```

TENSORFLOW PROFILER (2/2)