

A PDF format of README file which contains images is added to GitHub. Now you can follow the new instruction to run the algorithm for provided images using both CPU and GPU. Besides, we made sure that without need to change any addresses or download any new file, you can run the algorithms. Also, we added more help and error messages to the scripts. The following instruction and the images indicate step by step for running Inception-V1 for sample embryo images.

To run the pipeline please follow these steps:

Run the convert.py (it is located in the "STORK/scripts" directory) to allocate the suitable percentage of images to train and validation sets. The convert.py needs three arguments including: the address of images for training, the address of where the result will be located, and the percentage of validation images for the training step.

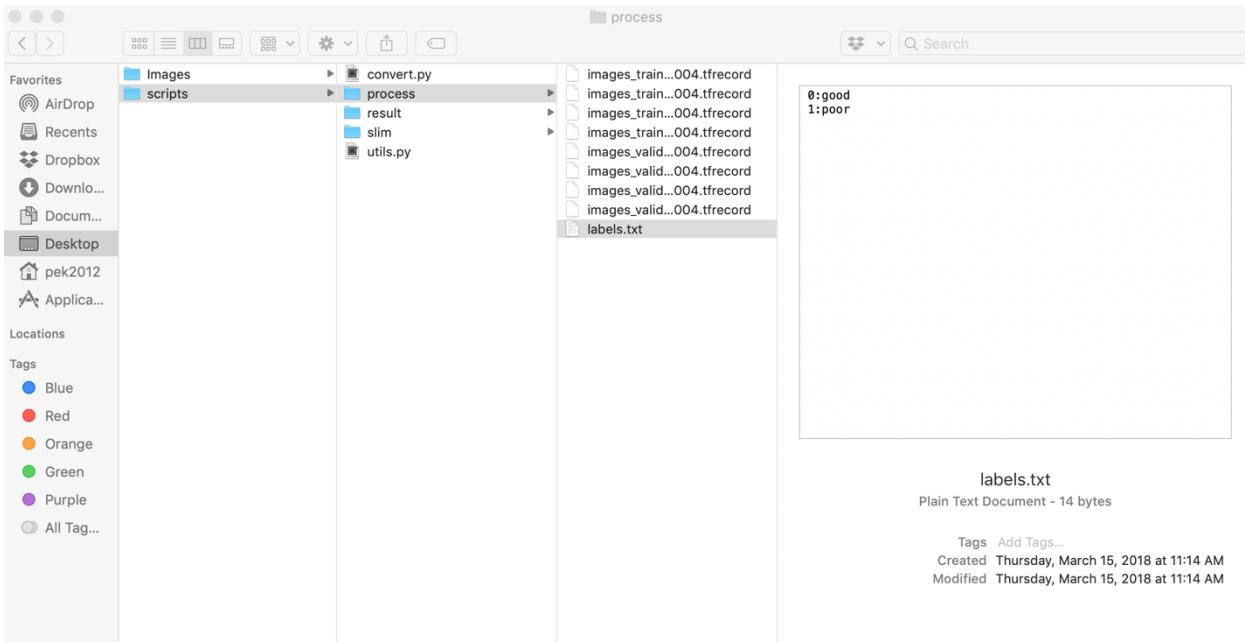
```
convert.py

1  from __future__ import absolute_import
2  from __future__ import division
3  from __future__ import print_function
4
5  import math
6  import os
7  import random
8  import sys
9
10
11 import tensorflow as tf
12 sys.path.append("slim/datasets") #add the "datasets" directory from "slim" to the system path
13 import dataset_utils
14
15
16
17 # Seed for repeatability.
18 _RANDOM_SEED = 0
19
20 # The number of shards per dataset split.
21 _NUM_SHARDS = 4
22
23
24 class ImageReader(object):
25     """Helper class that provides TensorFlow image coding utilities."""
26
27     def __init__(self):
28         """Initializes function that decodes RGB JPEG data.
29         self._decode_jpeg_data = tf.placeholder(dtype=tf.string)
30         self._decode_jpeg = tf.image.decode_jpeg(self._decode_jpeg_data, channels=3)
31
32     def read_image_dims(self, sess, image_data):
33         image = sess.run(self._decode_jpeg,
34                          {self._decode_jpeg_data: image_data})
35         return image.shape[0], image.shape[1]
36
37     def decode_jpeg(self, sess, image_data):
38         image = sess.run(self._decode_jpeg,
39                          {self._decode_jpeg_data: image_data})
40         assert len(image.shape) == 3
41         assert image.shape[2] == 3
42         return image
43
44
45     def _get_filenames_and_classes(dataset_dir):
46         """Returns a list of filenames and inferred class names.
47         Args:
48             dataset_dir: A directory containing a set of subdirectories representing
49                         class names. Each subdirectory should contain PNG or JPG encoded images.
50         Returns:
51             A list of image file paths, relative to `dataset_dir` and the list of
52             subdirectories, representing class names.
53         """
54
55         directories = []
56         class_names = []
57         for filename in os.listdir(dataset_dir):
58             print(filename)
59             path = os.path.join(dataset_dir, filename)
60             if os.path.isdir(path):
61                 directories.append(path)
62                 class_names.append(filename)
63
64         image_filenames = []
65         for directory in directories:
66             for filename in os.listdir(directory):
67                 path = os.path.join(directory, filename)
68                 if path.endswith('.jpg') != -1:
69                     image_filenames.append(path)
70
71         return image_filenames, sorted(class_names)
72
73     def _get_dataset_filename(dataset_dir, output_dir, split_name, shard_id):
74         output_filename = output_dir + '/images-%05d-of-%05d.tfrecord' % (
75             split_name, shard_id, _NUM_SHARDS)
76         return output_filename
77
78
79     def convert_dataset(split_name, filenames, class_names_to_ids, dataset_dir, output_dir):
80         """Converts the given filenames to a TFRecord dataset.
81         Args:
82             split_name: The name of the dataset, either 'train' or 'validation'.
83             filenames: A list of absolute paths to png or jpg images.
84             class_names_to_ids: A dictionary from class names (strings) to ids
85             (integers).
86
87
88 Line 1, Column 1
```

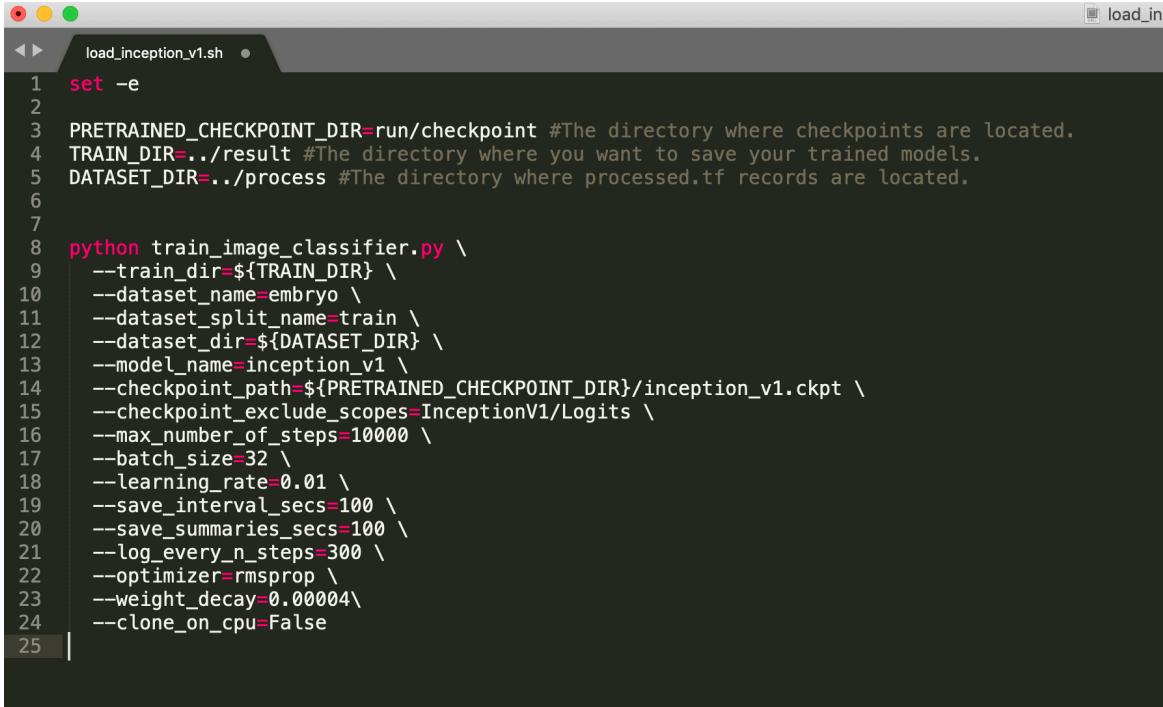
Run the following command in shell script:
`python convert.py ..\Images\train process/ 0`

```
Last login: Thu Jan  3 16:31:28 on ttys000
scripts — pek2012@pascal:~ bash — 126x56
maci78123:~ pek2012$ cd ~/Users/pek2012/Desktop/Flowchart/scripts
maci78123:scripts pek2012$ ls
convert.py      process      result      slim      utils.py
maci78123:scripts pek2012$ python convert.py ./Images/train process/ 0
```

The convert.py script saves converted .tf records in the "process" directory.



The CNN algorithm (e.g., Inception architecture) should be run on the training set images from the "STORK/scripts/slim" directory.

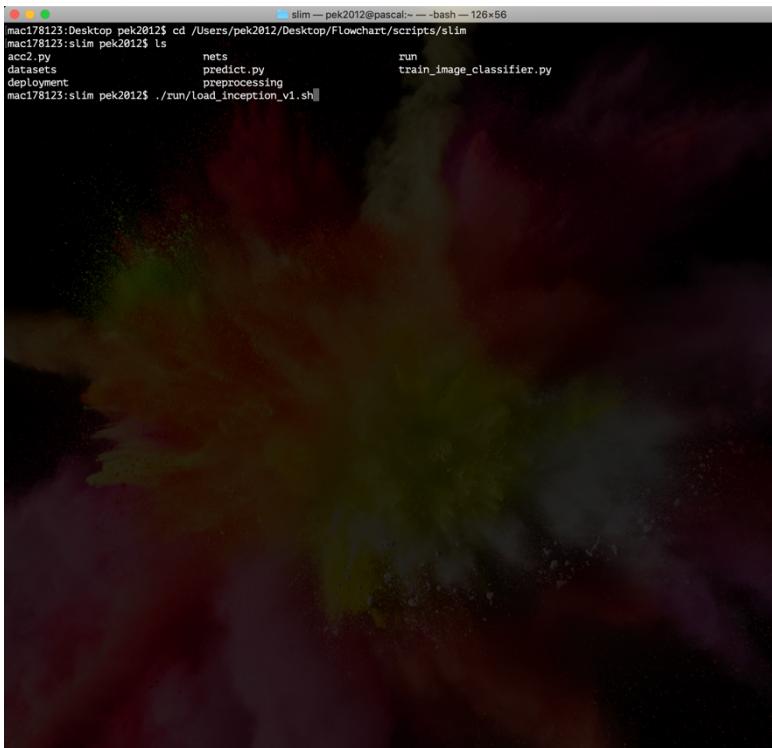


```
load_inception_v1.sh
set -e
PRETRAINED_CHECKPOINT_DIR=run/checkpoint #The directory where checkpoints are located.
TRAIN_DIR=../result #The directory where you want to save your trained models.
DATASET_DIR=../process #The directory where processed.tf records are located.

python train_image_classifier.py \
--train_dir=${TRAIN_DIR} \
--dataset_name=embryo \
--dataset_split_name=train \
--dataset_dir=${DATASET_DIR} \
--model_name=inception_v1 \
--checkpoint_path=${PRETRAINED_CHECKPOINT_DIR}/inception_v1.ckpt \
--checkpoint_exclude_scopes=InceptionV1/Logits \
--max_number_of_steps=10000 \
--batch_size=32 \
--learning_rate=0.01 \
--save_interval_secs=100 \
--save_summaries_secs=100 \
--log_every_n_steps=300 \
--optimizer=rmsprop \
--weight_decay=0.00004 \
--clone_on_cpu=False
25
```

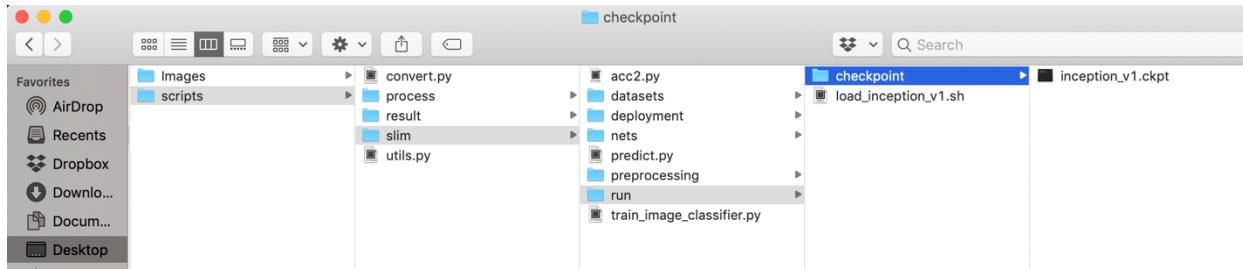
Then, run the following command in shell script:

[./run/load_inception_v1.sh](#)

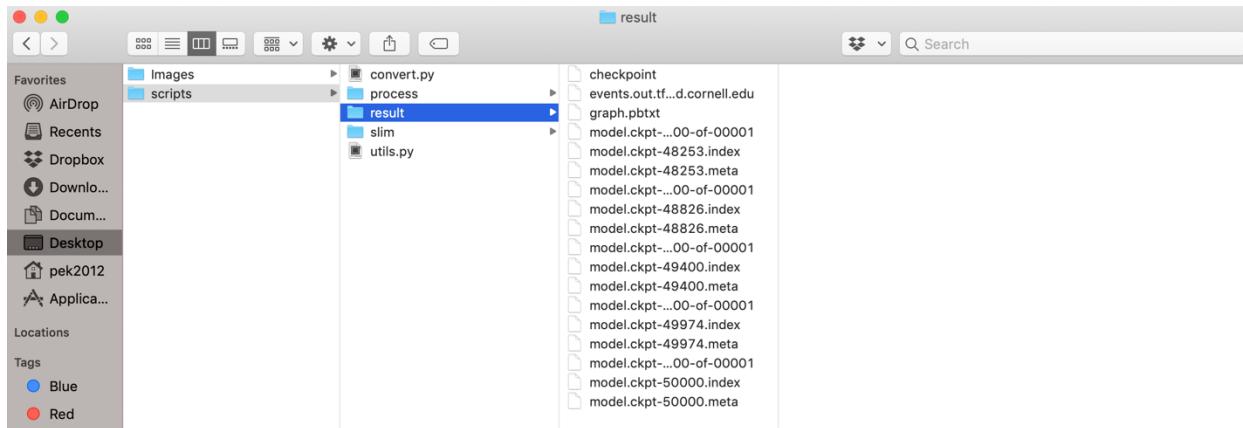


```
slim -- pek2012@pascal:~ - bash - 126x56
mac178123:Desktop pek2012$ cd /Users/pek2012/Desktop/Flowchart/scripts/slim
mac178123:slim pek2012$ ls
acc2.py           nets          run          train_image_classifier.py
datasets         predict.py    preprocessing
deployment
mac178123:slim pek2012$ ./run/load_inception_v1.sh
```

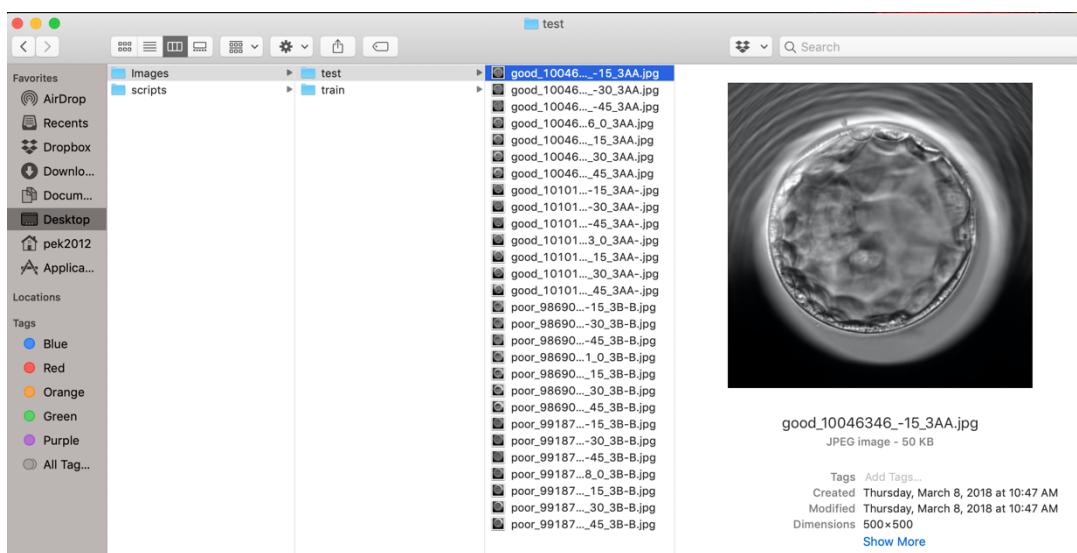
This script loads parameters of an inception v1 model which is trained over ImageNet dataset and saved in checkpoint. It excludes the last layer and then updates whole parameters of the model based on our provided images.



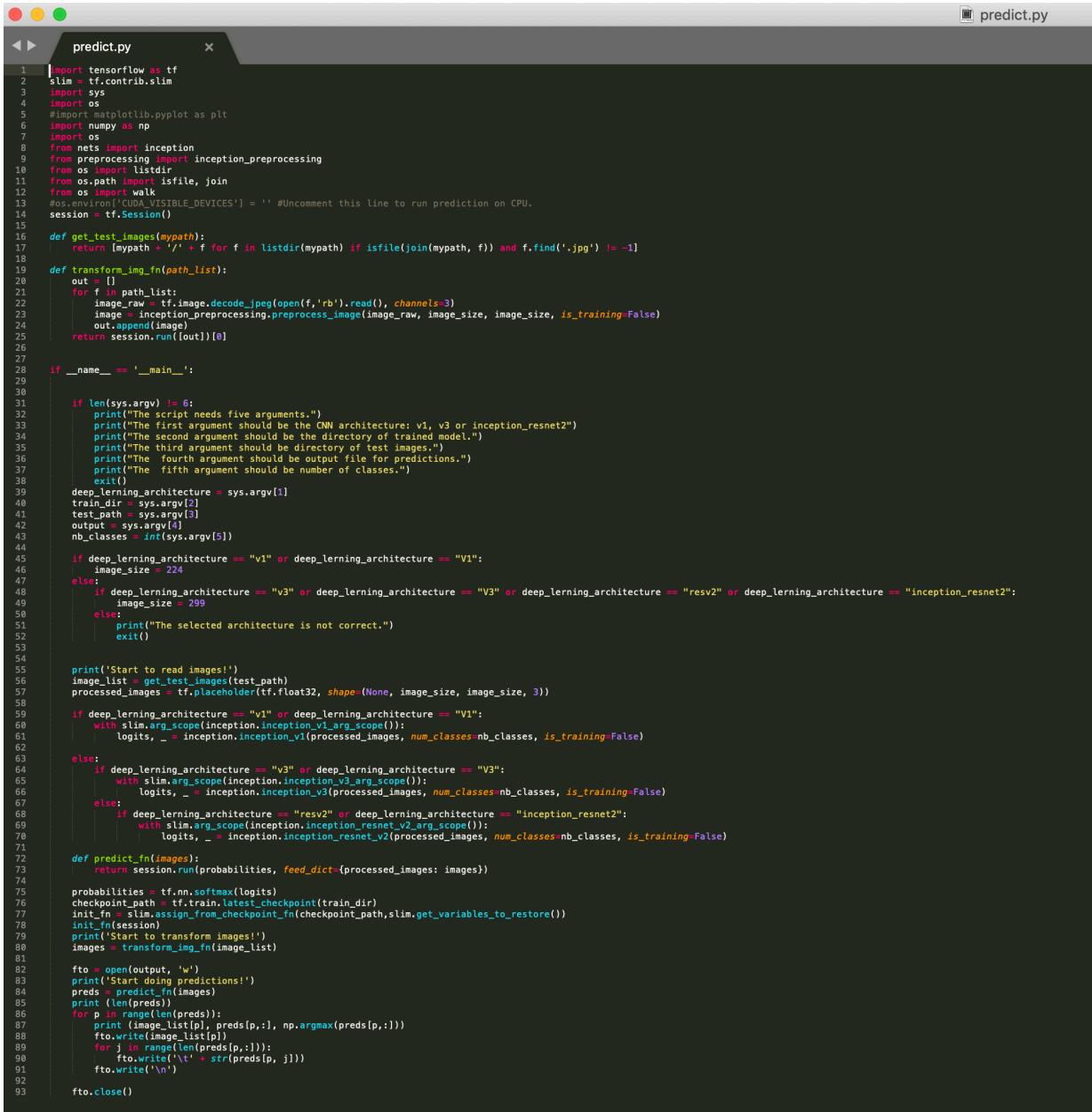
The results will be saved the "result" directory.



The trained algorithms should be tested using test set images which are located in STORK/Images/test folder.



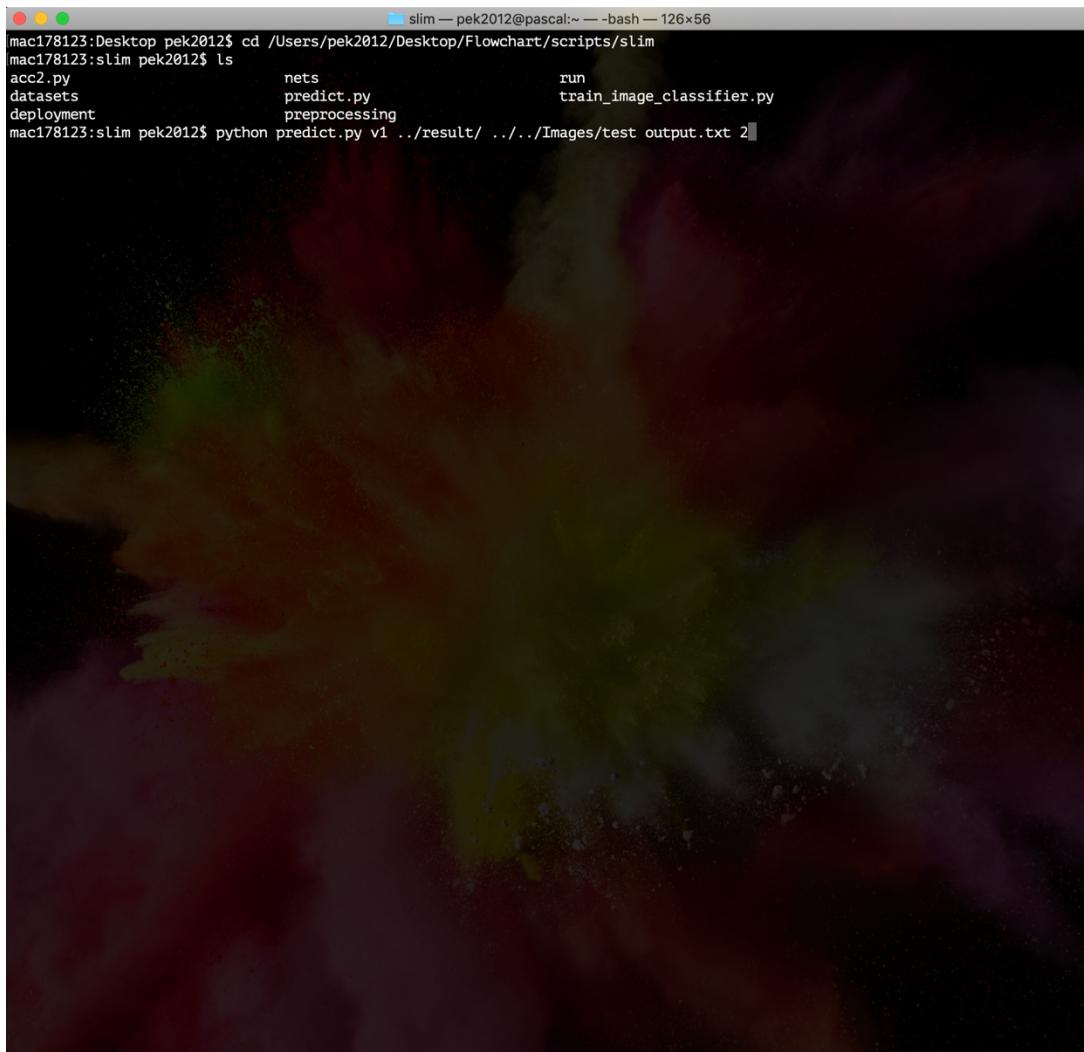
In folder "STORK/scripts/slim", predict.py loads a trained model on provided images.



```
predict.py
1 import tensorflow as tf
2 slim = tf.contrib.slim
3 import sys
4 import os
5 #import matplotlib.pyplot as plt
6 import numpy as np
7 import os
8 from nets import inception
9 from preprocess import inception_preprocessing
10 from os import listdir
11 from os.path import isfile, join
12 from os import walk
13 #os.environ['CUDA_VISIBLE_DEVICES'] = '' # Uncomment this line to run prediction on CPU.
14 session = tf.Session()
15
16 def get_test_images(mypath):
17     return [mypath + '/' + f for f in listdir(mypath) if isfile(join(mypath, f)) and f.find('.jpg') != -1]
18
19 def transform_img_fn(path_list):
20     out = []
21     for f in path_list:
22         image_raw = tf.image.decode_jpeg(open(f,'rb').read(), channels=3)
23         image = inception_preprocessing.preprocess_image(image_raw, image_size, image_size, is_training=False)
24         out.append(image)
25     return session.run([out])[0]
26
27
28 if __name__ == '__main__':
29
30
31     if len(sys.argv) != 6:
32         print("The script needs five arguments.")
33         print("The first argument should be the CNN architecture: v1, v3 or inception_resnet2")
34         print("The second argument should be the directory of trained model.")
35         print("The third argument should be directory of test images.")
36         print("The fourth argument should be output file for predictions.")
37         print("The fifth argument should be number of classes.")
38         exit()
39     deep_learning_architecture = sys.argv[1]
40     train_dir = sys.argv[2]
41     test_path = sys.argv[3]
42     output = sys.argv[4]
43     nb_classes = int(sys.argv[5])
44
45     if deep_learning_architecture == "v1" or deep_learning_architecture == "V1":
46         image_size = 224
47     else:
48         if deep_learning_architecture == "v3" or deep_learning_architecture == "V3" or deep_learning_architecture == "resv2" or deep_learning_architecture == "inception_resnet2":
49             image_size = 299
50         else:
51             print("The selected architecture is not correct.")
52             exit()
53
54
55     print('Start to read images!')
56     image_list = get_test_images(test_path)
57     processed_images = tf.placeholder(tf.float32, shape=(None, image_size, image_size, 3))
58
59     if deep_learning_architecture == "v1" or deep_learning_architecture == "V1":
60         with slim.arg_scope(inception.inception_v1_arg_scope()):
61             logits, _ = inception.inception_v1(processed_images, num_classes=nb_classes, is_training=False)
62
63     else:
64         if deep_learning_architecture == "v3" or deep_learning_architecture == "V3":
65             with slim.arg_scope(inception.inception_v3_arg_scope()):
66                 logits, _ = inception.inception_v3(processed_images, num_classes=nb_classes, is_training=False)
67
68         else:
69             if deep_learning_architecture == "resv2" or deep_learning_architecture == "inception_resnet2":
70                 with slim.arg_scope(inception.inception_resnet_v2_arg_scope()):
71                     logits, _ = inception.inception_resnet_v2(processed_images, num_classes=nb_classes, is_training=False)
72
73     def predict_fn(images):
74         return session.run(probabilities, feed_dict={processed_images: images})
75
76     probabilities = tf.nn.softmax(logits)
77     checkpoint_path = tf.train.latest_checkpoint(train_dir)
78     init_fn = slim.assign_from_checkpoint_fn(checkpoint_path, slim.get_variables_to_restore())
79     init_fn(session)
80     print('Start to transform images!')
81     images = transform_img_fn(image_list)
82
83     fto = open(output, 'w')
84     print('Start doing predictions!')
85     preds = predict_fn(images)
86     print(len(preds))
87     for p in range(len(preds)):
88         print(image_list[p], preds[p,:], np.argmax(preds[p,:]))
89         fto.write(image_list[p])
90         for j in range(len(preds[p,:])):
91             fto.write('\t' + str(preds[p, j]))
92             fto.write('\n')
93     fto.close()
```

This code get 5 arguments:

[python predict.py v1/result/ ../../Images/test output.txt 2](#)



```
slim — pek2012@pascal:~ — bash — 126x56
mac178123:Desktop pek2012$ cd /Users/pek2012/Desktop/Flowchart/scripts/slim
mac178123:slim pek2012$ ls
acc2.py          nets          run
datasets         predict.py    train_image_classifier.py
deployment       preprocessing
mac178123:slim pek2012$ python predict.py v1 ..../result/ ../../Images/test output.txt 2
```