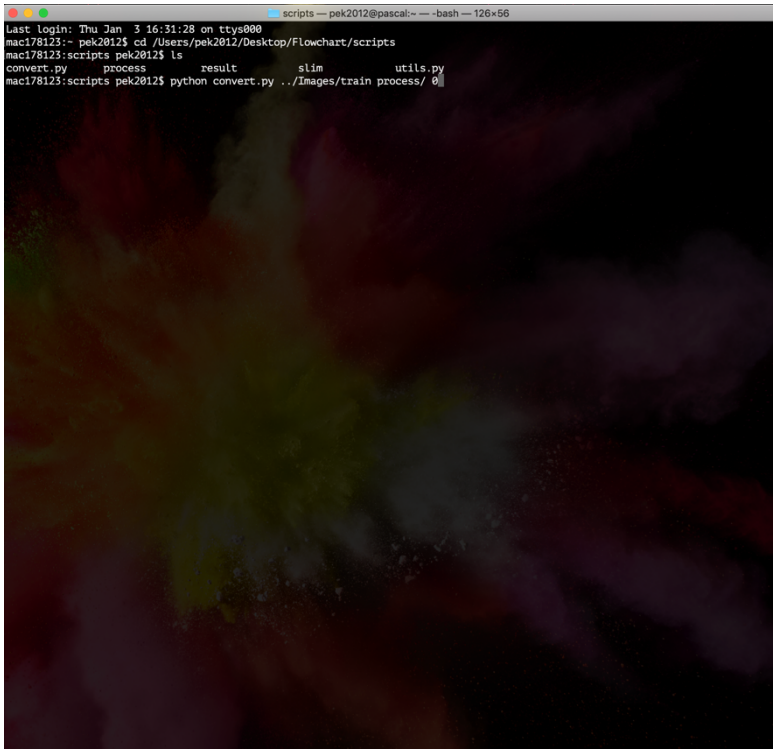
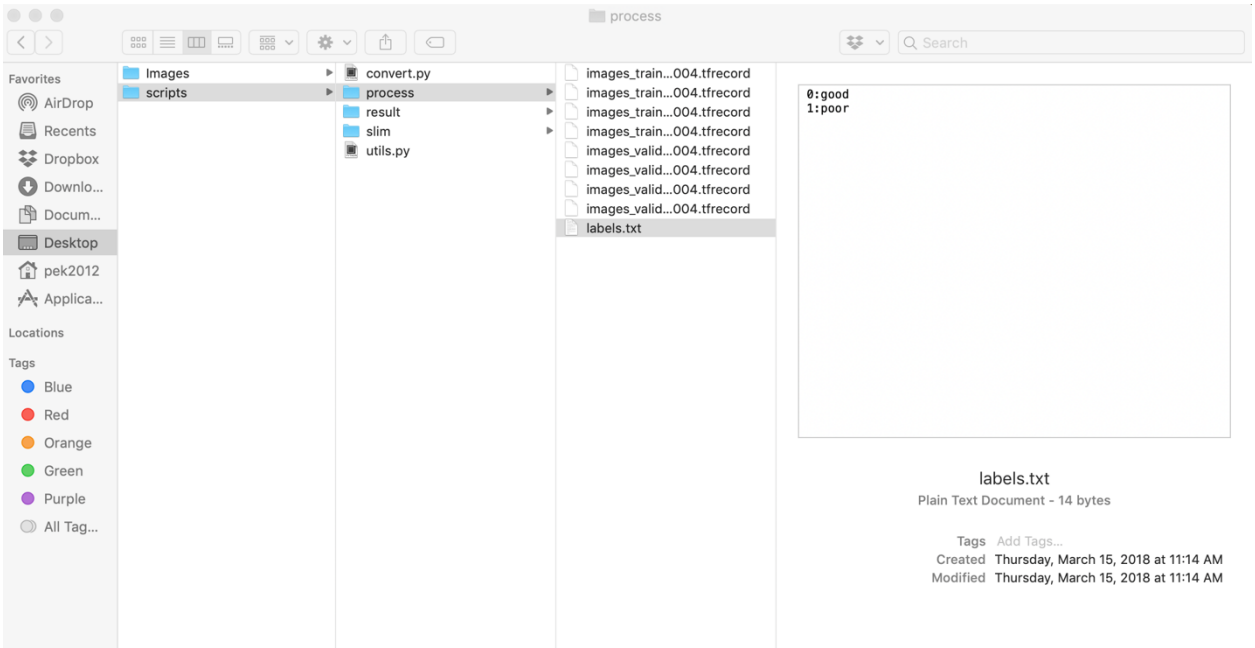
A PDF format of README file which contains images is added to GitHub. Now you can follow the new instruction to run the algorithm for provided images using both CPU and GPU. Besides, we made sure that without need to change any addresses or download any new file, you can run the algorithms. Also, we 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.

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




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



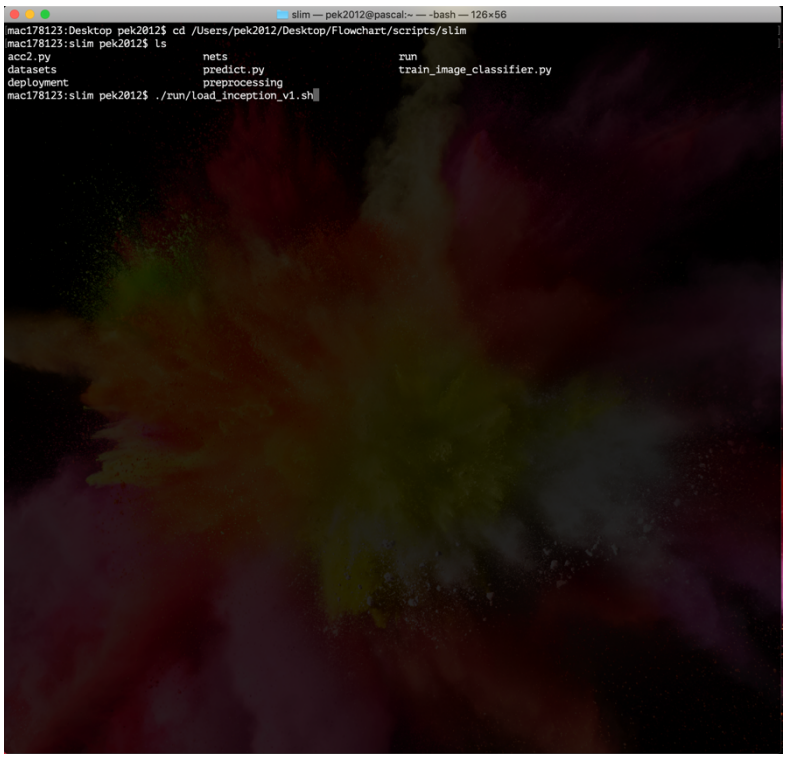
The CNN algorithm (e.g., Inception architecture) should be run on the training set images from the "STORK/scripts/slim" directory.

load_inc

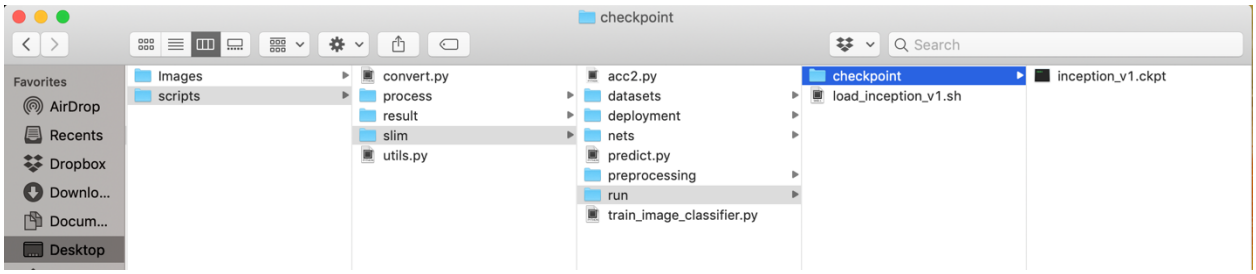
load_inception_v1.sh

```
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

```

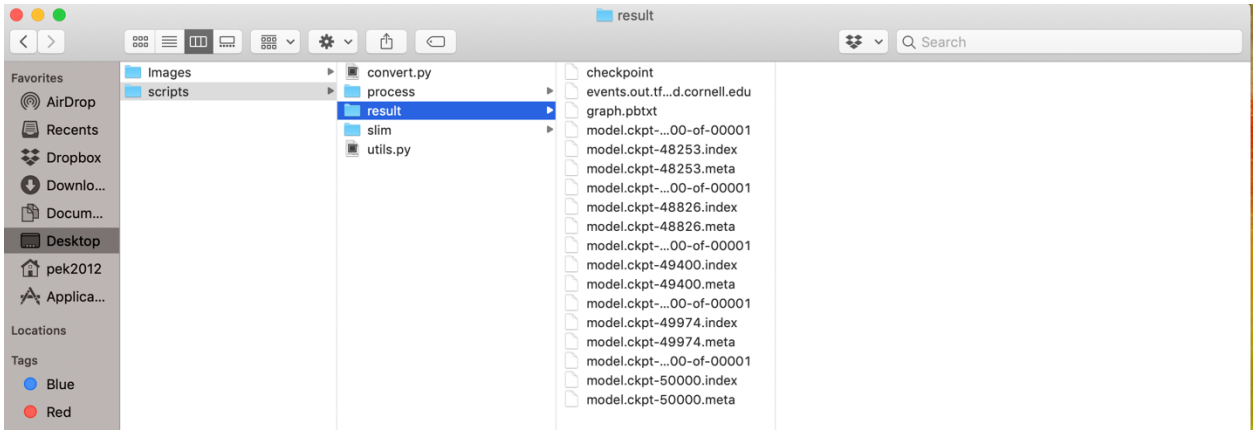
Then, run the following command in shell script:
./run/load_inception_v1.sh



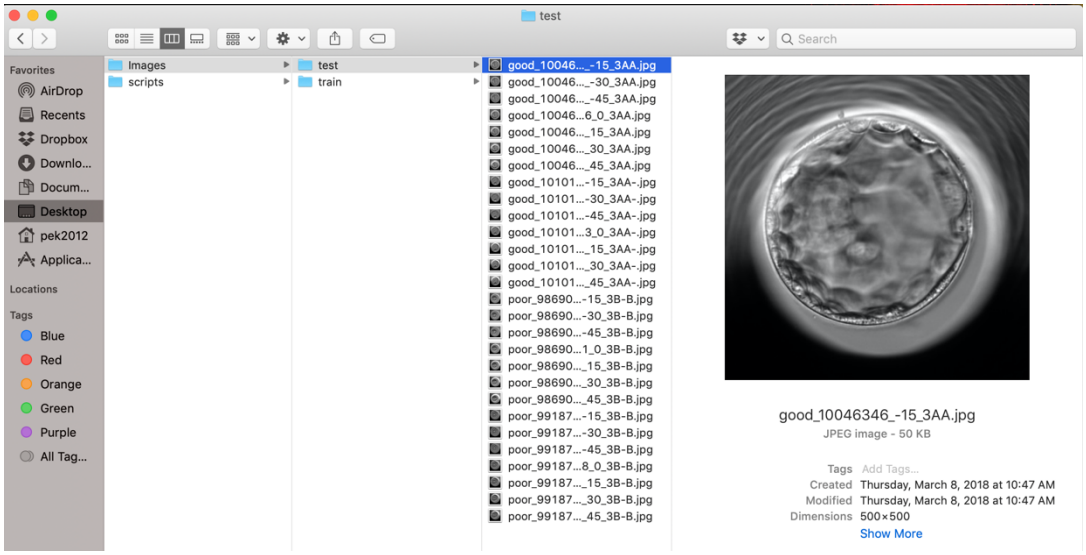
This script loads parameters of an inception v1 model which is trained over ImageNet dataset and saved in checkpoint. It excludes the last layer and then updates whole parameters of the model based on our provided images.




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

```python
import tensorflow as tf
slim = tf.contrib.slim
import sys
import os
#import matplotlib.pyplot as plt
import numpy as np
import os
from nets import inception
from preprocessing import inception_preprocessing
from os import listdir
from os.path import isfile, join
from os import walk
#os.environ['CUDA_VISIBLE_DEVICES'] = '' #Uncomment this line to run prediction on CPU.
session = tf.Session()

def get_test_images(mypath):
    return [mypath + '/' + f for f in listdir(mypath) if isfile(join(mypath, f)) and f.find('.jpg') != -1]

def transform_img_fn(path_list):
    out = []
    for f in path_list:
        image_raw = tf.image.decode_jpeg(open(f,'rb').read(), channels=3)
        image = inception_preprocessing.preprocess_image(image_raw, image_size, image_size, is_training=False)
        out.append(image)
    return session.run([out])[0]


if __name__ == '__main__':


    if len(sys.argv) != 6:
        print("The script needs five arguments.")
        print("The first argument should be the CNN architecture: v1, v3 or inception_resnet2")
        print("The second argument should be the directory of trained model.")
        print("The third argument should be directory of test images.")
        print("The  fourth argument should be output file for predictions.")
        print("The  fifth argument should be number of classes.")
        exit()
    deep_lerning_architecture = sys.argv[1]
    train_dir = sys.argv[2]
    test_path = sys.argv[3]
    output = sys.argv[4]
    nb_classes = int(sys.argv[5])

    if deep_lerning_architecture == "v1" or deep_lerning_architecture == "V1":
        image_size = 224
    else:
        if deep_lerning_architecture == "v3" or deep_lerning_architecture == "V3" or deep_lerning_architecture == "resv2" or deep_lerning_architecture == "inception_resnet2":
            image_size = 299
        else:
            print("The selected architecture is not correct.")
            exit()


    print('Start to read images!')
    image_list = get_test_images(test_path)
    processed_images = tf.placeholder(tf.float32, shape=(None, image_size, image_size, 3))

    if deep_lerning_architecture == "v1" or deep_lerning_architecture == "V1":
        with slim.arg_scope(inception.inception_v1_arg_scope()):
            logits, _ = inception.inception_v1(processed_images, num_classes=nb_classes, is_training=False)

    else:
        if deep_lerning_architecture == "v3" or deep_lerning_architecture == "V3":
            with slim.arg_scope(inception.inception_v3_arg_scope()):
                logits, _ = inception.inception_v3(processed_images, num_classes=nb_classes, is_training=False)
        else:
            if deep_lerning_architecture == "resv2" or deep_lerning_architecture == "inception_resnet2":
                with slim.arg_scope(inception.inception_resnet_v2_arg_scope()):
                    logits, _ = inception.inception_resnet_v2(processed_images, num_classes=nb_classes, is_training=False)

    def predict_fn(images):
        return session.run(probabilities, feed_dict={processed_images: images})

    probabilities = tf.nn.softmax(logits)
    checkpoint_path = tf.train.latest_checkpoint(train_dir)
    init_fn = slim.assign_from_checkpoint_fn(checkpoint_path,slim.get_variables_to_restore())
    init_fn(session)
    print('Start to transform images!')
    images = transform_img_fn(image_list)

    fto = open(output, 'w')
    print('Start doing predictions!')
    preds = predict_fn(images)
    print (len(preds))
    for p in range(len(preds)):
        print (image_list[p], preds[p,:], np.argmax(preds[p,:]))
        fto.write(image_list[p])
        for j in range(len(preds[p,:])):
            fto.write('\t' + str(preds[p, j]))
        fto.write('\n')

    fto.close()
```

This code get 5 arguments:
python predict.py v1 ../result/ ../../Images/test output.txt 2