PROJECT REPORT
OPENFACE SOFTWARE DEPLOYMENT ON DOCKER AND UBUNTU

Suman Duvvuru, Sonal Shrivatsava, Gregor Laszewski

Table of Contents

1 ABSTRACT

The goal of the project is openface software deployment on ubuntu and docker systems. A performance study delineating the different frameworks for the various demonstrations that needs to be created for the face detection software. This performance study will be repeatable and in some fashion new results on different machines will be integrated and represented pictorially.

The two key steps are:
1. docker deployment using Dockerfiles for face comparison (openface:demo2) and face classification (openface:demo3)
2. ubuntu deployment using ansible for face comparison (openface:demo2) and face classification (openface:demo3).

2 INTRODUCTION

Openface Software description

OpenFace is a Python and Torch implementation of face recognition with deep neural networks and is based on the CVPR 2015 paper FaceNet: A Unified Embedding for Face Recognition and Clustering by Florian Schroff, Dmitry Kalenichenko, and James Philbin at Google.

The Openface software workflow involves the following steps:

1. Detect faces with a pre-trained models from dlib or OpenCV.
2. Transform the face for the neural network. This repository uses dlib's real-time pose estimation with OpenCV's affine transformation to try to make the eyes and bottom lip appear in the same location on each image.
3. Use a deep neural network to represent (or embed) the face on a 128-dimensional unit hypersphere.
4. Applies clustering or classification techniques to the features to complete the recognition task.

There are different demonstrations that are performed by software:

* Demo 1: Real time web: This demo does the full face recognition pipeline on every frame. In practice, object tracking like dlib's should be used once the face recognizer has predicted a face.

* Demo 2: Comparing two images: The comparison demo outputs the predicted similarity score of two faces by computing the squared L2 distance between their representations.

* Demo 3: Classifier demo : OpenFace's core provides a feature extraction method to obtain low-dimensional representation of any face. demos/classifier.py shows a demo of how these representations can be used to create a face classifier.

For the purpose of this project Demo 2 and Demo 3 will be used and deployed on Ubuntu and Docker systems. The performance of openface project will be studied on different machines and environment, such as docker on OSX and Ubuntu deployment using ansible. The comparative results of openface software performance is represented pictorially in the form of graphs and charts.

3 ALGORITHM

One of the most widely used methods for human detection is the HOG (Histograms of oriented gradients) [1]. For face detection one popular method is based on Harr wavelet and SVM classifier described in the paper [2]. We could use the OpenCV implementation of human and face detection for our project [3]. Mahout and/or Spark's MLlib library for machine learning consisting of common learning algorithms could be used for classification for human and face detection. (http://spark.apache.org/docs/1.2.1/mllib-guide.html) .To download the code see [4]

4 DATASET

The current models in openface project are trained with a combination of the two largest (of August 2015) publicly-available face recognition datasets based on names: FaceScrub and CASIA-WebFace[4].

The models can be found under "openface/models" folder which is downloaded while pulling bamos/openface image :

nn4.v1
nn4.v2
nn4.small1.v1
nn4.small2.v1

The performance is measured by averaging 500 forward passes with util/profile-network.lua and the following results use OpenBLAS on an 8 core 3.70 GHz CPU and a Tesla K40 GPU.

Model	Runtime (CPU)	Runtime (GPU)
nn4.v1	**75.67 ms ± 19.97 ms**	**21.96 ms ± 6.71 ms**
nn4.v2	**82.74 ms ± 19.96 ms**	**20.82 ms ± 6.03 ms**
nn4.small1.v1	**69.58 ms ± 16.17 ms**	**15.90 ms ± 5.18 ms**
nn4.small2.v1	**58.9 ms ± 15.36 ms**	**13.72 ms ± 4.64 ms**

For this project, for majority of the simulations, a subset of images from the dataset that is already being provided as part of the images directory of openface installation was utilized for the assessment of performance of ubuntu and docker runs on multiple VMs and containers.

MUCT (Milborrow / University of Cape Town) dataset: In addition, images from MUCT database [5] was used for a quick evaluation of the Ubuntu performance on a single VM and on DOCKER on single container. The MUCT database consists of 3755 images from 276 unique subjects. The main motivation for the creation of the database was to provide more variety than the existing publicly available landmarked databases — variety in terms of lighting, age, and ethnicity. The MUCT landmarks are the 68 points defined by the popular FGnet [3] markup of the XM2VTS database [2], plus four extra points for each eye. This dataset is available for download via github at https://github.com/StephenMilborrow/muct.git

Possible Development Tools

1. *Big-Data* → Apache Hadoop, Apache Spark, OpenCV, Apache Mahout, MLlib - Machine Learning Library, DataMPI
2. *Languages* → Java, Python, Scala, R

Pre-Requisites

1. Operating Systems: Linux or OSX
2. VirtualBox: https://www.virtualbox.org/wiki/Downloads
3. Docker: Download and install docker Toolbox: https://www.docker.com/toolbox
4. Ubuntu 14.04: Chameleon Openstack

5 IMPLEMENTATION

5.1 DOCKER

All THE COMMANDS SHOULD BE EXECUTED ON THE TERMINAL ON WHICH DOCKER IS LAUNCHED !!!

MULTI-SERVER REPLICATION STEPS ::

==========

These steps will execute openface project on multiple docker swarm nodes and collect their optputs for graph plots.

1. Clone the ansible-cloudmesh-face github respository ::

$ mkdir -p ansible-cloudmesh-face
$ git clone https://github.com/cloudmesh/ansible-cloudmesh-face.git
$ cd ansible-cloudmesh-face/docker/

To check Dokcer is installed properly ::

$ source openface_dep.sh

2. Create the docker swarm cluster with openface containers ::

docker$ source openface-multiserver.sh <Number of swarm nodes to be run>

This command will create required number of nodes in docker swarm cluster. In the above command 2nd argument takes number of node that you want to run.

Note: Please be aware that in addition to the swarm nodes you specified there will always a Master-node and Machine-node created to enable the process. The name of the nodes will be openface-node<number of the node>.Master node can be identified as openface-master
and key-store as openface-machine.

WARNING : If you get an error saying "openface" container already exists or "openface" name has been given to another container, then you could kill the existing openface container using commands - docker-machine rm $(docker-machine ls -q).

3. Container will be created for nodes in the swarm one-by-one. First node will create the conatiner and it will pull the bamos/openface image. Upon image pull the command prompt will change from docker $ to root1111111# , i.e. prompt control changes from host to
container. Once on container change directory to Docker folder by ::

root1111111# cd /root/openface/docker

4. Verify if the required scripts are present in container ::

docker# ls -l

demo2.sh and demo3.sh should be present in the current directory.

5. To run Face Comparison demo ::

docker# source demo2.sh <Number of times script to be run>

This command will create files “docker_compare_<container-id>.csv" and “docker_compare_<container-id>.txt" as output in the current directory.

Verify these output files ::

docker# cat docker_compare_$CID.csv

docker# cat docker_compare_$CID.txt

Note: CID is the id of the container.

6. To run Face Recognition demo ::

docker# source demo3.sh <Number of times script to be run>

This will carete files “docker_classifier_<container-id>.csv" and “docker_classifier_<container-id>.txt" as output in the current directory.

Verify these output files ::
docker# cat docker_classifier_$CID.csv
docker# cat docker_classifier_$CID.txt

Note: CID is the id of the container.

7. Exit from the container of node1 ::
docker# exit

8. As soon as node1 is exited new container for next node will open and it will pull the bamos/openface image. Upon image pull the command prompt will change from docker $ to root1111111# and this will be repeated for all the nodes in the swarm cluster ::

Repeat step 3 to 7 for all the nodes

9. The results from all the containers will be saved in mounted folder i.e /ansible-cloudmesh-face/docker on host. On host machine under docker folder verify the output files generated by multiple containers ::

docker$ ls -l

10. Gather csv files for graph plot ::
This will gather all the output files to ansible-cloudmesh-face/performance folder

docker$ source gather-csv.sh

11. Get a pictorial presentation of docker and ubuntu time comaprision ::

cd ../performance
performance$ Rscript plot_demo2.R
performance$ Rscript plot_demo3.R

Graphs saved by the name demo2_real_plot.png , demo2_sys_plot.png and demo2_user_plot.png for Demo2 Face comparision and demo3_real_plot.png , demo3_sys_plot.png and demo3_user_plot.png for Demo3 Face classifier , under
"ansible-cloudmesh-face/performance" folder.

12. The swarm nodes will remain on the host in detached mode. To get attached to any of these nodes run following command ::

$ eval $(docker-machine env --swarm openface-node<node_number>)

To check the swarm node information ::

$ docker-machine ls

13. To kill all the swarm nodes ::

$ docker-machine rm $(docker-machine ls -q)

NOTE: This command will kill all the swarm nodes from the host and they have to be recreated if required , using step 2.

14. BIG DATA: Test Openface on big dataset (MUCT)

- Download the MUCT dataset via git ::

cd ansible-cloudmesh-face

git clone https://github.com/StephenMilborrow/muct.git

Start the docker swarm container using step 2 and 3 and run the following commands inside the containers:

- Run the demos using a big dataset ::

./demo2big.sh N

./demo3big.sh N

5.2 UBUNTU

VM Replication steps (VMs run on Chameleon Openstack)
==========
Step 1: Install Openface

(i) using the ansible script (ubuntu_openface.yml) that using ansible methods to install all the dependencies and the openface software

```
ansible-playbook  ubuntu_openface.yml -i inventory.txt -u cc
```

OR
(ii) Run the shell script directly on the VMs.

```
./openface_ubuntu.install.sh
```

Step 2: Copy Scripts for running demo2 (demo2b.sh) and demo3 (demo3b.sh) to VMs

Once the installation is complete, run a script to copy the demo2, demo3 scripts to run on the example data and MUCT data

```
./democopy.sh
```

Step 3: Execute the demo2 and demo 3 for a certain number of iterations on VMs (used N=50)

```
./demo2b.sh  N
./demo3b.sh  N
```

The results files (ubuntu_compare_uid.csv and ubuntu_classifier_uid.csv) are being generated

Step 4: Copy the results to the local git directory (ansible-cloudmesh-face/performance folder) for analysis

```
scp cc@vm-ip:openface/ubuntu* .csv .
```
Repeat this for all VMs

Step 5: Run analysis to generate descriptives and box plots

Once the docker files were generated then run the Rscripts to generate 3 plots for demo2 and 3 plots for demo3 corresponding to user, real and sys times and further generate the means and SDs for comparison. This script needs to be run from the local directory ((ansible-cloudmesh-face/performance folder) containing all the results csv files

Rscript demo2_summaryPlots.R
Rscript demo3_summaryPlots.R
Rscript demo_mean_sd.R

Step 6: Test Openface on big dataset (MUCT)

- Download the MUCT dataset via git
  ssh cc@vm-ip
  git clone https://github.com/StephenMilborrow/muct.git

- Run the demos using a big dataset
  ./demo2big.sh N
  ./demo3big.sh N

Script locations

-YML and inventory file
  cloudmesh-ansible-face/ubuntu/ubuntu_openface.yml
  cloudmesh-ansible-face/ubuntu/inventory.yml

-Demos running scripts using sample data from openface installation
  Demo2: cloudmesh-ansible-face/ubuntu/demo2b.sh
  Demo3: cloudmesh-ansible-face/ubuntu/demo3b.sh
  Copy: cloudmesh-ansible-face/ubuntu/democopy.sh

-Demos running scripts using publicly available MUCT data
  Demo2: cloudmesh-ansible-face/ubuntu/demo2big.sh
  Demo3: cloudmesh-ansible-face/ubuntu/demo3big.sh

-Analysis Scripts
  cloudmesh-ansible-face/performace/summaryPlots_demo2.R
  cloudmesh-ansible-face/performace/summaryPlots_demo3.R
  cloudmesh-ansible-face/performace/demos_mean_sd.R

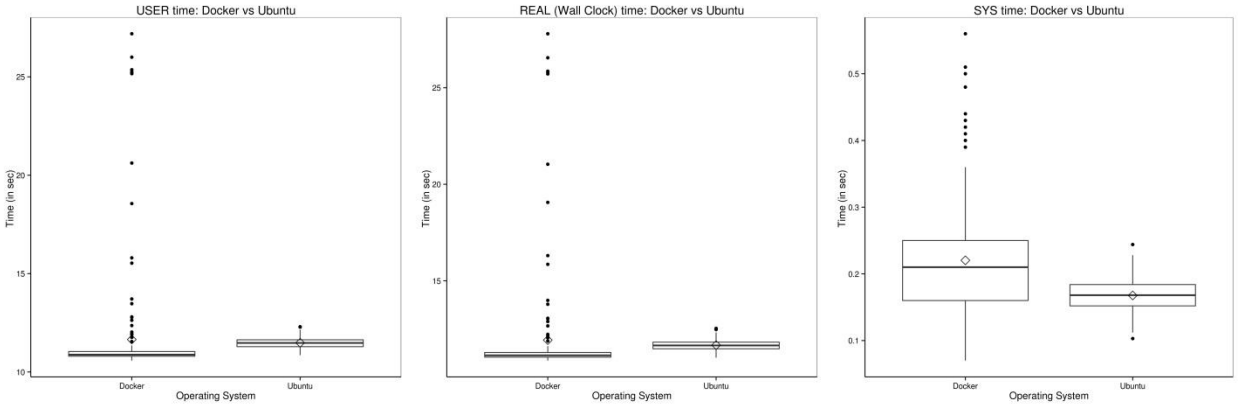
More details can be found in the README file at
  cloudmesh-ansible-face/ubuntu/README.rst

6 RESULTS

Following is the Box-whisker graph plot between performances of Docker (on Mac OSX) and Ubuntu (on Windows 10). The graph compares real time (wall clock time) , system time and user time performance of running openface face comparison and face detection models

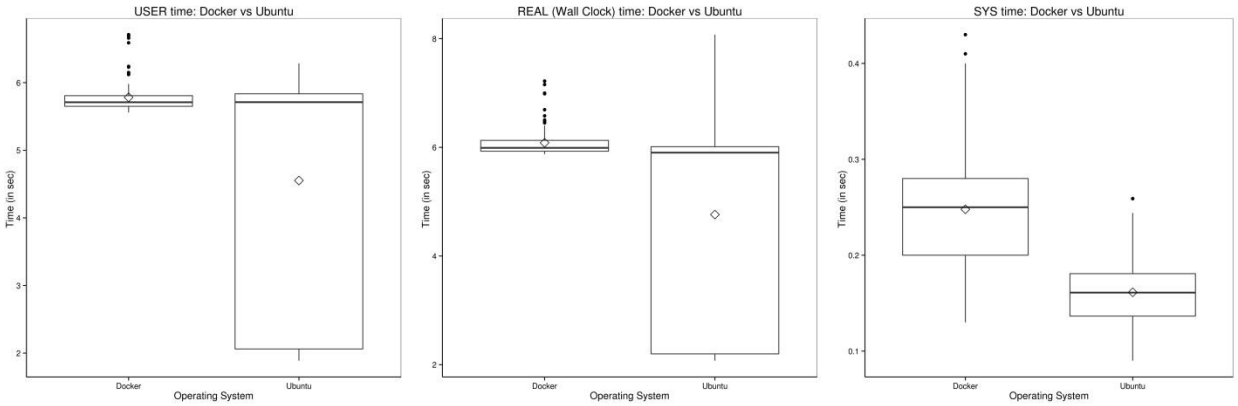
on multiservers using docker swarm containers and ubuntu vms (ansible). Please note that the Ubuntu version 14.04 running on the computer with the following hardware: processor: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz with 993MiB System memory. Docker Server Version: 1.11.1 running on the computer with the following hardware: Processor Name: Intel Core i5; Processor Speed: 2.7 GHz; Memory: 8 GB.

For demo 2, though the mean/median real and user times are slightly higher for ubuntu compared to docker as seen in the figure below, the time variability is higher for docker with several outliers with much higher run times compared to Ubuntu. So Ubuntu has a better consistency compared to Docker for demo2 runs. For the SYS time, ubuntu run's had much lower run time compared to docker and hence ubuntu clearly outperformed docker in this case.



Time_OS	Mean	SD
real_docker	11.21717	0.454894
real_ubuntu	12.07338	0.845406
user_docker	10.96413	0.438258
user_ubuntu	11.78477	0.599313
sys_docker	0.200375	0.062151
sys_ubuntu	0.16882	0.032894

For demo 3, as seen in the figure below, the mean/median real,user and sys times are clearly lower for ubuntu compared to docker and it can be concluded that ubuntu clearly outperformed docker for demo 3 performance.



Time_OS	Mean	SD
real_docker	6.084225	0.265752
real_ubuntu	4.763067	1.828127
user_docker	5.782941	0.234468
user_ubuntu	4.552927	1.790613
sys_docker	0.247843	0.058676
sys_ubuntu	0.16128	0.032198

Clearly the comparison demo takes much longer to run than the classifier demo based on the above tables and plots.

Since the comparison is made on different computer hardware, the differences we observed may be due to the differences in the Operating system or the system hardware and caution needs to be taken during the interpretation of the results.

DOCKER Performance on MUCT dataset on Container for 5 runs

Demo2

real (sec)	**user (sec)**	**sys (sec)**
1.309	1.230	0.060
1.284	1.180	0.090
1.284	1.220	0.050
1.303	1.190	0.100
1.284	1.150	0.120

Demo3

real (sec)	**user (sec)**	**sys (sec)**
1.704	1.550	0.120

1.673	1.480	0.160
1.678	1.550	0.100
1.680	1.480	0.170
1.675	1.510	0.140

UBUNTU Performance on MUCT dataset on VM for 5 runs

Demo2

real (ms)	**user (ms)**	**sys (ms)**
1929.759	1920.629	6.342
1860.003	1850.934	6.4
1810.215	1801.176	6.51
1825.297	1816.27	6.447
1808.802	1800.191	6.096

Demo3

real (ms)	**user (ms)**	**sys (ms)**
954.778	950.244	3.144
935.221	930.625	3.238
909.851	905.548	3.047
914.797	910.339	3.172
911.077	906.591	3.223

Clearly the as MUCT is a much bigger dataset, it takes longer time to run and the comparison demo takes much longer than the classifer demo in all the runs as you can see from the tables above.

Limitations:

Docker Swarm instead of pulling private image automatically on all the swarm nodes simultaneously performs a one-by-one pull on each swarm node container.

Future directions

Future enhancements could include developing ansible scripts to distribute the big datasets such as MUCT or LFW facial image datasets and run the analysis on multiple servers and combine them to get the integrated results. Native and docker installations on Ubuntu can be further explored.

Future enhancement on Docker swarm would be to add script to distribute data and instructions to independently run on all the containers.

7 ACKNOWLEDGMENTS

Gregor von Laszewski
Badi Abdul-Wahid

8 REFERENCES

[1]. Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." In Computer Vision and Pattern
Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 886-893. IEEE, 2005.
[2] . Viola, Paul, and Michael J. Jones. "Robust real-time face detection." International journal of computer vision 57.2 (2004): 137-154.
[3]. Bradski, Gary, and Adrian Kaehler. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.
http://opencv.org/
[4]. Brandon Amos, Bartosz Ludwiczuk, Jan Harkes, Padmanabhan Pillai,Khalid Elgazzar, and Mahadev Satyanarayanan.OpenFace: Face Recognition with Deep Neural Networks.http://github.com/cmusatyalab/openface.Accessed: 2016-01-1
[5] S. Milborrow and J. Morkel and F. Nicolls, The MUCT Landmarked Face Database. Pattern Recognition Association of South Africa. (2015)
[6] FGnet, Face and Gesture Recognition Working Group. www-prima. imag.fr/FGnet, 2004.
[7] K. Messer, J. Matas, J. Kittler, J. Luettin, and G. Maitre, XM2VTS: The Extended M2VTS Database. AVPBA, 1999.