**OPENFACE SOFTWARE DEPLOYMENT ON DOCKER AND UBUNTU**

**Suman Duvvuru, Sonal Srivatsava, Gregor Laszewski**

**[laszewski@gmail.com](mailto:Laszewski@gmail.com)**

Table of Contents

1 ABSTRACT 1

2 INTRODUCTION 1

3 ALGORITHM 2

4 DATASET 2

5 IMPLEMENTATION 4

5.1 DOCKER 4

5.2 UBUNTU 6

6 RESULTS 8

7 ACKNOWLEDGMENTS 9

8 REFERENCES 9

# ABSTRACT

We summarize some initial efforts to deploy and execute the 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 achievements are:

1. A docker deployment using Dockerfiles for face comparison (openface:demo2) and face classification (openface:demo3)
2. An ubuntu deployment using ansible for face comparison (openface:demo2) and face classification (openface:demo3).

# INTRODUCTION

OpenFace is a Python and Torch implementation of a face recognition framework utilizing deep neural networks. It is described in a paper published in CVPR 2015” ”*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 included as part of the software distribution:

* **Demo 1. Real time web:** This demo conducts 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. The program *demos/classifier.py* shows 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. With the help of the included demonstration programs we will measure the performance for each of them and compare the performance on different machines and environments. This includes docker on OSX and Ubuntu deployments using ansible. The comparative results of openface software performance is represented as box whisker diagrams.

# 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 [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>) .The code can be downloaded from [4].

UNCLEAR

# DATASET

The current models in openface project are trained with a combination of the two largest publicly-available face recognition datasets based on names (as of August 2015): 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** |

THIS IS A FIGURE AND MUST BE REFERNCESD SO IN THE TEXT . If it’s a list than it should be bulleted

|  |
| --- |
| **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** |

Forthe performance study carried out by us  a subset of images from the dataset that is  provided as part of the images directory of openface installation was utilized. As previously mentioned, we conduct performance studies using ubuntu and docker runs on multiple VMs.

**MUCT (Milborrow / University of Cape Town) dataset**: In addition, images from MUCT database was used for a quick evaluation of the Ubuntu performance on a single VM. The MUCT database consists of 3755 images from 276 unique subjects. The motivation for the creation of the database was to provide more variety in terms of lighting, age, and ethnicity than originally available landmarked databases. This dataset is available for download via github at <https://github.com/StephenMilborrow/muct.git>.

**Possible Development Tools**

1. *Big-Data*
2. Apache Hadoop, Apache Spark, OpenCV, Apache Mahout, MLlib -Machine Learning Library, DataMPI
3. *Languages*
4. 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

# IMPLEMENTATION

## DOCKER

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

<1> MULTI-SERVER REPLICATION STEPS ::

===============================

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

1. Clone theansible-cloudmesh-facegithubrespository::

$mkdir-pansible-cloudmesh-face

$ git clonehttps://github.com/cloudmesh/ansible-cloudmesh-face.git

$ cd ansible-cloudmesh-face/docker/

To checkDokceris installed properly ::

$ source openface\_dep.sh

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

docker$ sourceopenface-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 beopenface-node<number of the node>.Master node can be identified asopenface-master and key-store asopenface-machine.

NOTE : Ifyou 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 in step:13 for fresh installation OR you could attach to this existing container using commands in step:12.

3. Container will be created for nodes in the swarm one-by-one. First node will create the container and it will pull thebamos/openface image. Upon image pull the command prompt will change from docker$ to root1111111# , i.e.promtcontrol changes from host to container. Once on container change directory todcokerfolder 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 willcaretefiles “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 ofnode1::

docker# exit

8. As soon asnode1is exited new container for next node will open and it will pull thebamos/openfaceimage. Upon image pull the

command prompt will change fromdocker$ toroot1111111# 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 ::

docker$ source gather-csv.sh

11. Get a pictorial presentation of docker and Ubuntu time comparison::

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 forDemo2Facecomparisonand

demo3\_real\_plot.png , demo3\_sys\_plot.png and demo3\_user\_plot.png forDemo3Face 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-machineenv--swarmopenface-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.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## UBUNTU

VM Replication steps (VMs run on Chameleon Openstack)

**Step 1:  Install Openface**

* Step 1.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

* Step1.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

# RESULTS

Our results are visualized with Box-whisker graph plots while showcaseing the performance 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).

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.

TABLES

|  |  |  |
| --- | --- | --- |
| 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.

**UBUNTU Performance on MUCT dataset on VM for 5 runs**

Demo2:

|  |  |  |
| --- | --- | --- |
| **real** | **user** | **sys** |
| **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** | **user** | **sys** |
| **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 classifier demo in all the runs as you can see from the tables above.

UNFORTUNATELY YOU MISSED BIG OPORTUNITY. THERE IS NO DESCRIPTION OF SWARM VS MULTI SERVER ENVIRONMENT. THERE IS NO DISCUSSION ON HOW YOU DISTRIUTED THE DATA. INFACT THE DOCUMENTATION PROVIDED AS PART OF DOCKER INDICATES THAT THE PARALELISM HAS NOT BEEN ACHIEVED AND THAT MULTIPLE EXPERIMENTS ARE CARRIED OUT SEQUENTIALLY RATHER THAN IN PARALLEL.

# ACKNOWLEDGMENTS

      Gregor von Laszewski   
      Badi Abdul-Wahid 

# 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)