# AI Face Recognition with OpenCV DNN module

--------------------------------------------------------------------------------------------------

|  |  |
|---|---|
| **Name**: | Thanh Trung Dinh |
| **University:** | Université de Technologie de Compiègne (UTC) |
| **Email address**: | dinhthanhtrung1996@gmail.com |
| **Skype ID**: | dinhthanhtrung1996 |

**Location** (Country, Time Zone):
France, GMT/UTC +1 (until 19 July)
Vietnam, GMT/UTC +7 (from 20 July)

--------------------------------------------------------------------------------------------------

## I. Introduction

digiKam is a well-known desktop application for photos management. In digiKam, tags on photos are strongly supported for the sake of providing users with a natural workflow of searching and arranging photos in their collections. Since many of our photos contain faces, face tag has apparently emerged as an essential property for any photos management software. With a huge ambition from digiKam team, face engine, which can scan photos and suggest face tags automatically basing on a few pre-tagged photos by users, has been developed in digiKam for a long time. However, being computationally expensive while not adequately accurate, that functionality is currently deactivated. Thus, this project aims to improve the performance and accuracy of facial recognition in digiKam, in order to bring this wonderful functionality back to users in a very soon release.

Efforts to improve facial recognition began in GSoC 2017 with great work of Yingjie Liu. According to the final work report, he finished implementing 4 algorithms for facial recognition, including: Eigen Face, Fisher Face, Local Binary Patterns Histograms (LBPH), Deep Neural Network (DNN) based algorithm. In addition, he also reported that DNN based algorithm achieved an outstanding result with 99% of accuracy after learning only from 1 photo pre-tagged. Despite this very promising result, facial recognition with DNN is extremely computational-costly. Moreover, current implementation with dlib is very difficult for maintenance and introduces a significant dependency to digiKam.

Currently, OpenCV has been already used in digiKam for image processing, as well as for implementing Eigen Face, Fisher Face and LBPH algorithms. Since OpenCV 3.1, DNN module, which is highly optimized for DNN based algorithms, has been introduced. Hence, using OpenCV DNN to replace dlib codes is hopefully promising to reduce computational cost while keeping the same accuracy level of performance as dlib-based implementation. In addition, it will reduce digiKam dependencies and code-base complexity.

In this document, I will represent firstly the deliverables. Then I will explain in details my planned implementation and finally my tentative timeline for each stage.

## II. Deliverables

For this project, the final work is expected to:

- Implement DNN based approach and unit tests with OpenCV DNN module
- Complete integration tests on computational and accuracy benchmark for face engine
- Study performance metrics and decide which algorithm and which kind of neural network architecture to use for facial recognition in digiKam
- (Optionally) Implement facial detection with OpenCV DNN module to replace current method using Haar cascade algorithm

## III. Implementation details

### A. Current implementation

#### Face recognition

Currently, digiKam has 4 algorithms used for facial recognition, which are: Eigen Face, Fisher Face, LBPH, DNN-based algorithm.

Eigen Face, Fisher Face and LBPH are implemented based on OpenCV framework.

Default algorithm for face recognition is LBPH, since it is the oldest and most complete algorithm implemented in digiKam. However, it needs at least 6 faces of a same person pre-tagged by user before being able to suggest face tags.

DNN-based algorithm was implemented with dlib framework in Liu's work during GSoC 2017. Although he did not describe exactly the architecture of neural network used for training and testing, it was implied that ResNet [1] architecture was implemented. Looking closely into the codes, I found evidence that he might implement a custom version of ResNet-34 [1].

**How do facial recognition algorithms in digiKam work?**

Eigen Face, Fisher Face and LBPH algorithms depends much on statistical analysis of image. Generally, basing on some characteristics well-studied that all faces share in common, we try to find a mathematical representation for each face. Finally, we compute the distance between faces with those representations and then decide whether a face "looks" similar to some other faces or not. Hence, we will be able to identify a face and predict face tag.

Meanwhile, DNN-based algorithm works in a quite different way. Analogously, DNN-based algorithm can be seen like a more "human" approach, where the model has to "learn" the subtle characteristics that human faces share in common. During the training process on many faces, the model will make prediction (i.e. *forward*), compute the "score" compared to real tags for faces, and use that value to "fit" itself steadily (i.e. *backward*). After thousands of iteration, the model will reach a level of "understanding" about human faces and then be able to use that to tell whether a face is the same as another face or not.

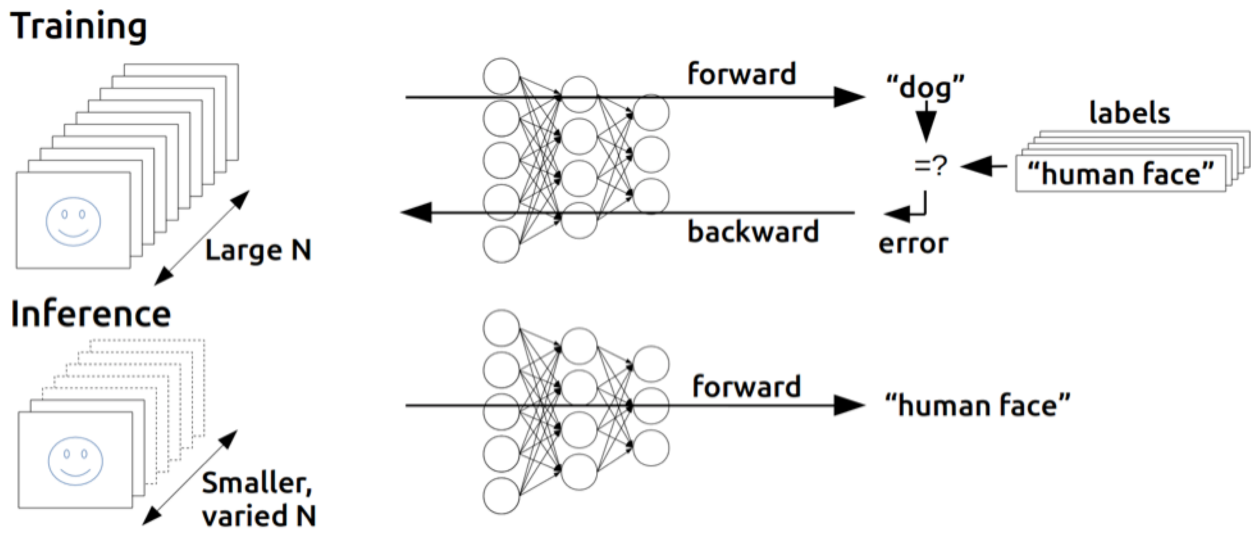
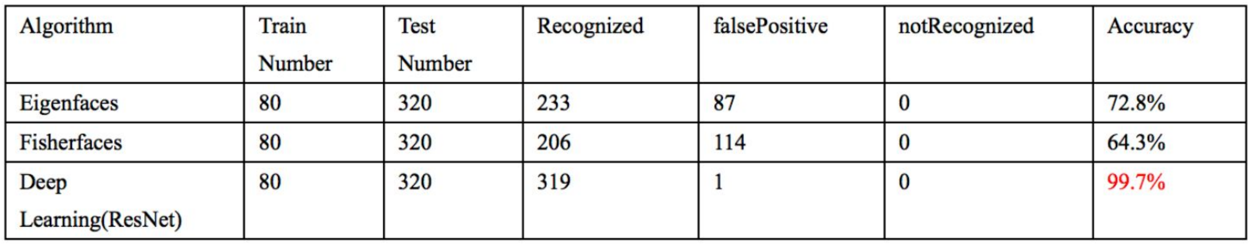


Figure 1: Simplified representation of neural network train and prediction phase

**What are the performance metrics of current facial recognition algorithms in digiKam?**

In Liu's work during GSoC 2017, he implemented tests and conducted a comparison [4] between 3 algorithms, which are Eigen Face, Fisher Face and DNN-based algorithm using orl dataset [5] as training and test set.

The results shown an obvious outperformance of DNN-based algorithm, in comparing with the other 2. Below is one of his results, described as "*2 images of each face as training images, 8 images of each face as testing images. We got 80 in training set and 320 in testing set*":



| Algorithm | Train Number | Test Number | Recognized | falsePositive | notRecognized | Accuracy |
|---|---|---|---|---|---|---|
| Eigenfaces | 80 | 320 | 233 | 87 | 0 | 72.8% |
| Fisherfaces | 80 | 320 | 206 | 114 | 0 | 64.3% |
| Deep Learning(ResNet) | 80 | 320 | 319 | 1 | 0 | 99.7% |

Figure 2: Accuracy comparison between digiKam facial recognition algorithms

We can see that DNN-based algorithm has an outstanding accuracy comparing to other algorithms.

**Why does DNN-based algorithm outperform other algorithms?**

The fact that Eigen Face, Fisher Face and LBPH are heavily statistic-based is the main reason while they are not accurate enough. Indeed, the mathematical model behind each of those algorithms is not perfect, because it is highly biased by approximative theories and prior assumptions on few facial characteristics well studied so far (e.g. face landmark, eye position, etc.). By learning from thousands of human faces to establish its own model, DNN-based algorithm is obviously more robust and accurate on a wide range of faces, especially for faces of people from different races.

**How was DNN approach implemented in digiKam?**

DNN-based algorithm in digiKam is implemented with dlib, an open source C++ cross-platform toolkit for machine learning algorithms.

Wrappers for DNN face model can be found in dlib-dnn directory, where there are different levels of wrapper, to maximize the possibility of extension:

1. Top level wrapper at opencvdnnfacerecognizer.cpp [6]
2. Lower level at dnn dnnfacemodel.cpp [7]
3. Lowest level at facerec_dnnborrowed.cpp [8]

Lowest level of wrapper calls member function *getFaceVector* [9] FaceDb. This member function then calls *getFaceVector* [10] of DNNFaceKernel defined in dnn_face.h. In this header file, we can see that *getFaceVector* [10] generally reads a pretrained model of neural network, gets the face region from shape predictor and “trains” the faces. The “training” process is executed when operator() is called with anet_type in the code at line 189: *net(faces)* [11].

Looking deeper into the codes, we can find the definition of loss_metric, which is the template type of anet_type, in loss.h [12]. Further downwards, we can see the definition for of *operator()* of loss_metric in core.h [13] where the data is really trained.

Basically, the *operator()* basically calls forward propagation [14] on neural network. Concretely, forward propagation computes output of a layer and passing it as input for the subnetwork (i.e. next layers down the road). Therefore, *getFaceVector* [10] only computes a kind of representation for faces with “knowledge” from neural network.

After that, when DNN tries to suggest tag for a face, it calls member function *predict* [15]. From the definition of this member function, we can see that it compares the representation of that face (got from *getFaceVector* [10]) with the representations of all other faces in database. It computes a set of euclidean distances between that face and other faces belonging to a tag, and for all the tags that user has confirmed. Then by finding the minimum distance, it decides whether that face belongs to a tag specific or it is a totally unknown face.

**What are current problems of DNN approach?**

Although DNN-based algorithm has a nearly perfect accuracy, it is extremely computational expensive, for 2 main reasons:

- Forward propagation may be computationally expensive if matrix calculations are not optimized. For example, ResNet-34 neural network has 34 layers with residual connections. Each layer does convolution operations with from 32 to 256 filters of size 7x7 on image of size 128x128. Since it is a huge amount of operations to conduct for each image, processing on hundreds of images will take hours to finish.
- Prediction meaning to compare a face with all faces in database may become too costly when user has hundreds of faces.

Besides, dlib is not well documented and its code base is very complicated for maintenance. In Liu’s work, he only implemented and tested ResNet with dlib, because importing dlib codes to digiKam already took a lot of time. Liu did a great work importing dlib codes in digiKam and making it work, but it really introduces a significant dependency to digiKam. In addition, dlib’s functionality overlaps OpenCV DNN module at some point. Hence, it is redundant in digiKam functional scope.

### Face detection

Face detection is currently performed on photos to determine the region where faces appear. It is also essential for the performance of facial recognition, since it detects the region of faces which are later cropped and fed to neural network model. Without face detection, face recognition will never be performed.

**What is current state of face detection in digiKam?**

Current implementation in digiKam uses OpenCV Haar Cascade classifier, which is good and fast. However, I have encountered cases where faces are not detected. Hence, faces in those case will never be recognized.

**Why should we re-implement face detection in digiKam?**

Since there exists cases where face detection with Haar Cascade algorithm does not work, we can also try DNN approach. Indeed, there is a well-known model called YOLO [2] used for object detection, which can detect object very accurately in real-time. Thus, DNN approach is also promising for digiKam face detection as well in terms of performance. In addition, using DNN approach also means that we only use one OpenCV module for both face recognition and face detection, which will reduce significantly code base complexity and maintenance effort.

Indeed, there are many examples and comparisons about using DNN for face detection (or more generally for object detection). I found a blog [16], where OpenCV Haar Cascade, OpenCV DNN module and dlib performance were compared for face detection. Besides, I also found a link [17] to OpenCV forum, where DNN-approach performance was discussed, including implementation examples and how to improve performance.

## B. Studied software components

Thanks to great work of Gilles Caulier, digiKam now can be compiled with OpenCV 4.0. Indeed, OpenCV 4.0 has many improvements and new features [18]. Hence, I am aiming for coding with OpenCV 4.0 API in this project.

Besides, according to suggestion from Marcel Wiesweg, I will use face-rec database [19] as test set. It is indeed a bigger and more current test set than orl dataset [3] used by Liu in his work.

Particularly for this project, OpenCV DNN [20] module will be implemented to replace dlib codes for DNN approach in digiKam face recognition. Originating as a GSoC project by Vitaliy Lyudvichenko, DNN module finally goes into OpenCV mainstream after plenty of optimizations. Aiming to fit in with OpenCV philosophy of an image processing framework, DNN module only allows forward propagation on pretrained model. Hence, it is more limited than dlib. However, using DNN module for prediction should be much faster.

In addition, OpenCV FLANN [21] module will also be studied during this project. As FLANN is optimized for fast nearest neighbor search in large datasets and for high dimensional features, it may be good solution to reduce the latency when computing distances used for face tag prediction.

## C. Planned implementation

### Face recognition

Liu's evaluation for different facial recognition algorithms was conducted with his own preprocessing but not the one in face workflow of digiKam. Therefore, so as to be able to evaluate the real performance of facesengine, we need tests implementing the whole face workflow. Current tests are implemented in facesengine testsuite directory [22]. Hence, I intend to review and complete those tests to achieve:

- Accuracy evaluation
- Performance evaluation, in terms of: latency, memory utilisation.

As a part of my work for this project, I intend to port current codes using OpenCV API in digiKam to maintain compatibility with OpenCV 4.0 API.

The main part of work for face recognition is reimplementing DNN-based approach with OpenCV DNN [20] module. For this part, I plan to:

- Implement OpenCV DNN alongside dlib
- Study different pretrained models
- Improve predict method for new face
- Evaluate overall performance in comparison with dlib

One of the most important step of this part will be selecting the appropriate DNN model which provides a satisfactory harmonisation between high level of accuracy and low computational cost. Currently, I found some good candidates:

- Squeeze-and-Excitation Networks [3]
- LightenedCNN_A.caffemodel, which was already mentioned as tested by Liu with OpenCV DNN module in his work before during GSoC 2017 [23]. He reported that it is light and fast, but not adequately accurate. I will inspect it more in details.
- Model Zoo [24] is also a source that I intend to reference later. Indeed, it has many pretrained model with corresponding paper.

Finally to solve the performance-bottleneck problem relating to prediction latency, I intend to base my implementation on OpenCV FLANN [21]. Then, by finishing performance test, I will be able to conclude more certainly about the improvement of this part.

## Face detection

Idea is also using pretrained model for object detection. I found a blog [25] describing an implementation of face detection with opencv. The blog also mentions ResNet-10 used for DNN module. Thus, ideal neural network models are light networks. After discussing with Gilles Caulier, he suggests that I should only do this task after finishing face recognition, since this task is purely for improvement of face detection.

Concretely, I will study YOLO [2] algorithm and implement a pretrained model with DNN module, so that we can use it to predict the bounding box for face region on photos as what is currently implemented using Haar Cascade algorithm. Indeed, changes will only be done in opencvfacedetector.cpp [26], so as not to break backward compatibility with UI of face management workflow.

## IV. Tentative timeline

**May 6 - May 27 (Community Bonding Period)**

- Review and complete test bench for face recognition
- Write standalone benchmark test for latency and memory

May 28 to June 11 (Week 1 - 2)

- Port dlib to OpenCV DNN with current neural network architecture
- Review and modify unit tests if needed

June 11 to June 23 (Week 3 - 4)

- Compare performance of OpenCV DNN and dlib
- Study different models
- Review and document for codes ported with OpenCV

**June 24 to June 28: Phase 1 Evaluation**

June 24 to July 8 (Week 5 - 6)

- Compare performance of different OpenCV DNN model implementation
- Continue on studying different models
- Review and document for codes ported with OpenCV

July 8 to July 21 (Week 7 - 8)

- Study on predict method optimization
- Review and modify unit tests if needed
- Review and document for codes ported with OpenCV

**July 22 to July 26: Phase 2 Evaluation**

July 22 - July 29 (Week 9)

- Study on predict method optimization (continue)
- Review and document for codes ported with OpenCV (continue)
- Review and modify unit tests if needed (continue)

July 29 - August 18 (Week 10 - 12)

- Ported Face detection with OpenCV DNN module
- Review and modify test bench and performance standalone test if needed
- Review and document for codes ported with OpenCV

**August 19 to August 26: Final phase Evaluation**
August 19 to August 26 (Week 13)

- Code coverage is done. Changes done (if required) for bug removal and improving functionality.
- Submit

**September 3: Final results of Google Summer of Code 2018 announced**

## V. Other commitment

I am currently working on my final year internship. My class for this semester lasts until July 19 and then I am going back to Vietnam for vacation. Therefore, I will take week 9 (July 22 to July 29) to catch up my work. After that, I will be able to work full time for GSoC.

## VI. About me

My name is Thanh Trung Dinh. I am a final year student in Embedded System Engineering at Université de Technologie de Compiègne, in France.

My first experience with C++ and Qt was in my Object Oriented Programming class at university 2 years ago, where I developed a simple text editor for the in-class final project with Qt. Since then, I have done many projects with C++ and Qt. In the last internship, I worked with Gilles Caulier (digiKam coordinator and lead developer) for the project of upgrading an acquisition video system, whose code base is written in Qt. Hence, I also had a great chance to familiarize with Gilles' coding style, which inspired me a lot.

During GSoC 2018, I worked for project: "digiKam Web Services tools authentication with OAuth2". I successfully finished the main parts of project when porting O2 for Google Drive, Google Photo, Facebook and Smugmug plugins in digiKam, as well as firstly created a template for a unified plugin manager for all web services.

I also have experiences with OpenCV. Currently, I am working on my final year internship in a medical company whose products are genetic therapies for gene-related eye diseases. I work directly with a device which transcodes normal light with different spectrum. Thus, I use OpenCV a lot at work for image processing and eye movement analysis.

Before GSoC 2018, I contributed a patch to digiKam core on porting geolocation tool from QWebKit to QWebEngine. In this task, I worked on shrinking the patch while maintaining backwards geolocation tool compatibility with QWebKit. Recently, I have picked up my work during GSoC 2018 to prepare codes for student who will continue my work on that project this year. I finished the Twitter Plugin for export tool, the plugin that student working for digiKam on that project last year did not have time to finish.

Below is the list of commits that I submitted, my status report for GSoC 2018 and list of bugs fixed

**Project contribution history**
Commits during GSoC 2018
All commits to digiKam

**Status Report**
https://community.kde.org/GSoC/2018/StatusReports/ThanhTrung

**Related fixed bugs**
https://bugs.kde.org/show_bug.cgi?id=363859
https://bugs.kde.org/show_bug.cgi?id=309508
https://bugs.kde.org/show_bug.cgi?id=383174
https://bugs.kde.org/show_bug.cgi?id=348274
https://bugs.kde.org/show_bug.cgi?id=395199
https://bugs.kde.org/show_bug.cgi?id=395199

## VII. Reference

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2016.91

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2016.91

[3] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. doi:10.1109/cvpr.2018.00745

[4] https://yjwudi.github.io/2017/06/18/Accuracy/

[5] https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[6] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/opencvdnnfacerecognizer.cpp

[7] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/dnnfacemodel.cpp

[8] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/facerec_dnnborrowed.cpp

[9] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/facedb/facedb.cpp#n425

[10] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/dnnface/dnn_face.h#n106

[11] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/dnnface/dnn_face.h#n189

[12] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/dnnface/nn/loss.h#n1140

[13] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/dnnface/nn/core.h#n2274

[14] https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/dnnface/nn/core.h#n2280

[15]
https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/recognition/dlib-dnn/facerec_dnnborrowed.cpp#n189
[16]
https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/
[17]
http://answers.opencv.org/question/185474/dlib-hack-for-improving-face-detection-rate/
[18] https://github.com/opencv/opencv/wiki/ChangeLog#version400
[19] http://www.face-rec.org/databases/
[20] https://docs.opencv.org/4.0.0/d6/d0f/group__dnn.html
[21] https://docs.opencv.org/4.0.0/dc/de5/group__flann.html
[22] https://cgit.kde.org/digikam.git/tree/core/tests/facesengine
[23] https://yjwudi.github.io/2017/06/25/investigation/
[24] https://github.com/BVLC/caffe/wiki/Model-Zoo
[25]
https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/
[26]
https://cgit.kde.org/digikam.git/tree/core/libs/facesengine/detection/opencvfacedetector.cpp