UNIVERSITY OF TURKU

MANUAL FOR THE PIPELINE FOR ANALYZING, VISUALIZING, AND PREDICTING THE DYNAMICS OF GAZE TO NATURALISTIC SOCIAL VIDEOS

Einari Vaaras
einari.vaaras@utu.fi, einarinpostia@gmail.com
Department of Psychology and Speech-Language Pathology
August 2021

CONTENTS

Clarification for the notations of this document:

- files and file paths/directories are written in *italics*
- commands for the command line/terminal are marked with a grey background
- websites are marked with blue underlined letters

LIST OF FIGURES

1. BACKGROUND

This manual gives details on how to install and use the present processing pipeline for analyzing, visualizing, and predicting the dynamics of gaze to naturalistic social videos. The pipeline is built around OpenFace[1], an open-source tool for facial behavior analysis. All of the scripts are publicly available in GitHub[2] and are written in the Python programming language to ensure that everyone can use the pipeline without the need for licensed software. What the pipeline contains is the following:

1. Scripts for pre-processing video data.
2. Scripts for analyzing video and fixation data using OpenFace.
3. Scripts for converting processed features into a suitable format for machine-learning algorithms, scripts for training a convolutional neural network model, and scripts for analyzing the performance of the model.
4. Scripts for determining the importance of the features of the convolutional neural network model.
5. Scripts for visualizing practically every process in the pipeline.

Please note that not all scripts or steps of the processing pipeline are necessary in order to utilize the pipeline in your studies. For example, if you have a set of videos and their respective fixation data in CSV files and you would want to visualize how the fixations look in the videos, you could just use the script described in Section 3.2.6 and ignore the rest. Or, if you would simply wish to change the resolution of some videos, you can just use the script described in Section 3.1.1. Many scripts in the present pipeline can be applied to various projects with very minor modifications, mostly just changing the input and output data directories.

For the machine-learning part of the processing pipeline, please note that the present scripts are not a ready-to-use solution for all other projects with similar data. For further discussion, please see part 2 of Appendix A.

[1] https://github.com/TadasBaltrusaitis/OpenFace
[2] https://github.com/infant-cognition-turku/gaze_dynamics

2. INSTALLATION

This section describes how to install the prerequisites for the pipeline to work properly. Please note that the installation process has been tested only on 64-bit operating systems “Windows 10 Home edition” for Windows and “Big Sur 11.0.1” for Mac, which might result in minor modifications required for other versions of Windows and Mac operating systems. The installation process has not been tested on any Linux operating system distribution, and hence a Linux installation guide is not featured in the present document. However, there should be no specific reasons on why the pipeline should not work for Linux distributions with very minor modifications. Also note that not everyone needs to install all the prerequisites listed here, only install those modules that are required for the scripts that you are going to use.

2.1 Recommendations for installation

The scripts of the present pipeline are implemented using the Python programming language, so you need to install Python first if you do not already have it on your computer (see online how to do so). If you do not yet have Python installed, I recommend using an Anaconda distribution to deal with compatibility problems that might occur by using `pip install <package_name>` or `pip3 install <package_name>`. By using the command `conda install <package_name>`, the Anaconda distribution automatically takes care of version compatibilities with different modules. For further instructions on the Anaconda distribution, please refer to the website https://www.anaconda.com.

2.2 Shutter Encoder and ffmpeg

The Shutter Encoder software can be installed on both Windows and Mac from the following website: https://www.shutterencoder.com/en/. Follow the instructions on the website for further details. Note that installing Shutter Encoder is only necessary for some minor video pre-processing actions (see Section 3.1.5 for additional information).

At its core, Shutter Encoder uses ffmpeg (https://www.ffmpeg.org/), which is perhaps the most widely-used video editing and coding software available. As a convenient addition when installing Shutter Encoder, Windows users will get a ready-to-use binary file of ffmpeg as a side product of the installation (located in Shutter Encoder’s “Library” folder inside the install directory). Mac users will need to install ffmpeg with the command `brew install ffmpeg`. The ffmpeg software is only used in the pre-processing scripts described in Sections 3.1.1, 3.1.2, and 3.1.3.

2.3 Python dependencies

The following Python dependencies are required for the OpenFace analysis pipeline:

1. opencv-python
2. numpy
3. tqdm
4. scipy

These dependencies can be installed e.g. by using `pip3 install <package_name>` or `conda install <package_name>`. In addition, since OpenFace is able to analyze only one face at a time, if you have multiple faces present in a video and you want to keep one of these faces and blur the rest, you need to install Py-Feat (see https://py-feat.org/content/intro.html for further instructions) to be able to use the script described in Section 3.1.4. Furthermore, if you want to use the machine-learning part of the pipeline (Section 3.3), the following Python dependencies are required:

1. pytorch (see https://pytorch.org/ for specific installation command)
2. tensorflow
3. sklearn
4. pyplot

Install these dependencies in a similar manner as those described in the beginning of the present section.

2.4 OpenFace

This section describes how to install OpenFace for Windows and Mac operating systems. OpenFace's GitHub wiki page (https://github.com/TadasBaltrusaitis/OpenFace/wiki#installation) also includes instructions for Ubuntu Linux users.

2.4.1 OpenFace for Windows

For Windows users, ready-made binaries for OpenFace can be downloaded and used to skip the installation process. For further information, please refer to the website https://github.com/TadasBaltrusaitis/OpenFace/wiki/Windows-Installation.

2.4.2 OpenFace for Mac

Mac users need to build the OpenFace binaries from scratch. The first thing to do is to try out the easy route (Option A from https://github.com/TadasBaltrusaitis/OpenFace/wiki/Mac-Installation),

which is the script on the page https://gitlab.com/Thom/fea_tool/-/blob/master/installer_scripts/macOS/openFace.sh (Figure 1).

```
#!/bin/bash
# script to install openFace for mac

# run this script while being in the fea_tool directory
#

brew update
brew install gcc
brew install boost
brew install tbb
brew install openblas
brew install --build-from-source dlib
brew install wget
brew install opencv

cd external_libs
mkdir openFace
cd openFace
git clone https://github.com/TadasBaltrusaitis/OpenFace.git
cd OpenFace

mkdir build
cd build
cmake -D WITH_OPENMP=ON CMAKE_BUILD_TYPE=RELEASE ..
make

cd ..
bash download_models.sh
cp lib/local/LandmarkDetector/model/patch_experts/*.dat build/bin/model/patch_experts/
```

Figure 1. *Simple script for installing OpenFace on Mac.*

To test out with your webcam whether OpenFace installed successfully, execute the following command on the command line (when you are in the OpenFace install directory): `build/bin/FaceLandmarkVid -device 0`.

If the previous script was not successful, then you should try the more difficult route (Option B from https://github.com/TadasBaltrusaitis/OpenFace/wiki/Mac-Installation). If this route does not turn out to be successful, then you can try out the following installation process, which combines information from OpenFace's Wiki page and a few different internet sources together with hands-on experience (successfully worked with Mac's "Big Sur 11.0.1" operating system):

1. Install Homebrew (https://brew.sh/) for an easy way to install various libraries.
2. Perform the following commands in the command line interface to install necessary libraries:

- brew update
- brew install gcc –HEAD
    - If the install fails for some reason, try the same command again for a few times. If that does not help, try again with the command brew install gcc.
- brew install boost
- brew install tbb
- brew install openblas
- brew install --build-from-source dlib
    - If the install fails for some reason, try the same command again for a few times. If that does not help, try again with the command brew install dlib.
- brew install wget
- brew install opencv
    - For some reason, OpenCV did not want to install nicely when performing this command. For the first time performing the command, two errors occurred: one including “Failed to read Mach-O binary” and another including “Directory not empty”. Once first restarting the computer and then performing exactly the same command again, the installation was able to proceed further than on the first try with only one error message. Once repeating the same command over and over again, the installation was able to proceed further and further in the process each and every time without modifying anything in between. Finally, when repeating the same command approximately 10 times in a row, the installation process fully succeeded.

3. Download the ZIP package of OpenFace (https://github.com/TadasBaltrusaitis/OpenFace). Extract the ZIP package into some directory where you want to keep OpenFace.
4. In the command line interface, go to the directory of OpenFace and run the script *download_models.sh* to download the landmark detection model (e.g. using the command sh ./download_models.sh or a similar command).
5. Install “Command Line Tools for Xcode” (https://developer.apple.com/downloads/) to make the installation process faster. However, this is not mandatory, since GCC is automatically built using Homebrew if you don’t want to use Xcode for some reason.
6. Install XQuartz (https://www.xquartz.org/) to make the installation easier.
7. In the OpenFace directory, there is a file *CMakeLists.txt*. In the very end of the file, add the following text (6 rows):

```
find_package( X11 REQUIRED )
MESSAGE("X11 information:")
MESSAGE("  X11_INCLUDE_DIR: ${X11_INCLUDE_DIR}")
MESSAGE("  X11_LIBRARIES: ${X11_LIBRARIES}")
MESSAGE("  X11_LIBRARY_DIRS: ${X11_LIBRARY_DIRS}")
include_directories( ${X11_INCLUDE_DIR} )
```

8. In the same file (*CMakeLists.txt*), in line 32 there should be the following text:

   find_package(OpenCV 4.0 REQUIRED COMPONENTS core imgproc calib3d highgui objdetect)

   Replace this text with the text

   find_package(OpenCV HINTS usr/local/Cellar/opencv/XXX/)

   where XXX is the version of OpenCV that is installed to your computer. By default, Homebrew installs programs to the directory *usr/local/Cellar/*, so if you go to the directory *usr/local/Cellar/opencv*, you should find your computer's version number of OpenCV. For example, if you have version 4.1.0_2 of OpenCV, you should replace the line 32 in the file *CMakeLists.txt* with:

   find_package(OpenCV HINTS usr/local/Cellar/opencv/4.1.0_2/)

9. After this, execute the following commands in the OpenFace directory using the command line interface:
   - mkdir build
   - cd build
   - cmake -D CMAKE_BUILD_TYPE=RELEASE ..

   If for some reason the command third command does not work, try out the following command instead:
   - cmake -D CMAKE_BUILD_TYPE=RELEASE CMAKE_CXX_FLAGS="-std=c++11" -D CMAKE_EXE_LINKER_FLAGS="-std=c++11" ..

10. In the directory where you are now, execute the following command using the command line interface:
    - make

11. Now, if everything went well, in the /bin directory (found in the OpenFace directory /build) there should be executable binary files. When you are in the OpenFace directory, you can try out if OpenFace works with your webcam by executing the following command on the command line:
    - build/bin/FaceLandmarkVid -device 0

3. USAGE

This section describes what each script does in and how to use each script in the present processing pipeline. Python scripts can be run with the command python <script_name> or python3 <script_name> from the command line/terminal, or by using some software for running Python scripts (e.g., Spyder, PyCharm, and Visual Studio). Note that in each script that requires an output directory, the output directory is automatically created if it does not already exist.

3.1 Pre-processing

This section gives descriptions about different scripts that can be applied to pre-processing video data. The final subsection (Section 3.5.5) describes how to use Shutter Encoder for video pre-processing.

3.1.1 change_video_resolution.py

The script *data_pre_processing/change_video_resolution.py* changes the resolution of videos in a given directory to a user-set resolution. This is convenient especially if the video data and the fixation data are not in the same resolution. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.1.2 change_video_sharpness_singlefile.py

Sometimes OpenFace might not be able to make proper face detections, e.g., due to bad lighting. In these cases, making the videos sharper might help in getting better face detections with OpenFace. The script *data_pre_processing/change_video_sharpness_singlefile.py* changes the sharpness of a single video file. This script is meant for cases if there are only a few videos that require sharpening. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.1.3 change_video_sharpness_multifile.py

The script *data_pre_processing/change_video_sharpness_multifile.py* changes the sharpness of all video files in a given directory. This script and its usage is similar to what is described in Section 3.1.2, except that this script is meant for cases if there are multiple videos that require sharpening.

3.1.4 pyfeat_blurface.py

Since OpenFace is only able to analyze one face at a time, a video should contain only one face visible before it is analyzed with OpenFace. The script *data_pre_processing/pyfeat_blurface.py* keeps only one face in a video and blurs/covers the rest of the faces that appear in a video, and is meant to be used for cases in which there are multiple faces seen in a video. To change the settings of the script, modify the file *conf_pyfeat_blurface.py*. To run the script, you can either use the command `python pyfeat_blurface.py` (if your configuration file has the name *conf_pyfeat_blurface.py*) or the command `python pyfeat_blurface.py <configuration_file>` (if you want to use a configuration file with a different name). This script uses Py-Feat (https://py-feat.org/content/intro.html) for face detection.

3.1.5 Cutting videos/changing video framerate using Shutter Encoder

Shutter Encoder is a handy open-source tool for video editing. For cutting videos using Shutter Encoder (e.g., if the fixation data and the video data do not start from the same video frame):

1. Drag/select your video(s) to-be-edited to the Shutter Encoder software
2. “Choose function” → “Output codecs” → Select “H.264”
3. “Input and output point” → Select “Change input and output point”
4. Select the start and end points of the video (can be selected frame-by-frame), and then select “Apply”
5. Select “Start function”

For changing the framerate of the video with Shutter Encoder (e.g., if video framerate and fixation data framerate are different from each other):

1. Drag/select your video(s) to-be-edited to the Shutter Encoder software
2. “Choose function” → “Output codecs” → Select “H.264”
3. “Advanced features” → “Conform by blending/interpolation” to (**desired fps**) fps
4. Select “Start function”

Please note that if you have an Nvidia GPU on your computer, you can speed up video processing with Shutter Encoder by approximately 20 times if you turn on from “Advanced features” the setting “Hardware acceleration: Nvidia NVENC” (or similar).

3.2 Analyzing video and fixation data

This section gives descriptions about different scripts that can be used for analysis for fixation data and pre-processed video data.

3.2.1 compute_openface_feats.py

The script *openface_analysis/compute_openface_feats.py* performs OpenFace video analysis for video files in a given directory. In addition, the script computes eye aperture from OpenFace features (if selected) and adds it to OpenFace's output CSV file. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.2.2 visualize_head_pose_and_landmarks.py

After using the script *compute_openface_feats.py*, you can visualize the 68 facial landmarks and head pose features provided by OpenFace using the script *visualization/visualize_head_pose_and_landmarks.py* in the video files that were analyzed using OpenFace. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.2.3 feature_interpolation.py

After visualizing the output of OpenFace using the script from Section 3.2.2, it is common that OpenFace might get bad detections in a few video frames here and there. To fix these brief misdetections, use the script *openface_analysis/feature_interpolation.py*. The script removes video frames with bad features based on a user-determined confidence threshold, and then creates new features using interpolation. With minor modifications, this script can be used for practically any time series data in a CSV format. Note that the script works best if there are no long intervals with bad detections. To change the settings of the script, modify the file *conf_feature_interpolation.py*. To run the script, you can either use the command `python feature_interpolation.py` (if your configuration file has the name *conf_feature_interpolation.py*) or the command `python feature_interpolation.py <configuration_file>` (if you want to use a configuration file with a different name). Once you have interpolated OpenFace's output CSV files using this script, it is a good idea to create a visualization of the interpolated features using the script from Section 3.2.2, and then to compare this visualization to a visualization of the original non-interpolated features. The script adds one column, "interpolated", to the end of the interpolated CSV file which can be utilized to check that which and how many frames were interpolated by the script. Please note that if there are long intervals with bad detections and interpolating features is not enough for fixing occurring issues, consider sharpening the videos (Sections 3.1.2 and 3.1.3) and then running the script from Section 3.2.1 again.

3.2.4 register_gaze_roi.py

Once you have OpenFace features ready, you can combine the information of OpenFace features and fixation data with the script *openface_analysis/register_gaze_roi.py*. This script combines the information of OpenFace features and fixation data, and computes the information of where a person is looking at (left eye/right eye/nose/mouth/face/other) into a CSV file frame-by-frame. This computation might take a while, so the script is made so that if it is suspended, it will continue the computing process from where it left off the next time the script starts running again. To change the settings of the script, modify the file *conf_register_gaze_roi.py*. To run the script, you can either use the command `python register_gaze_roi.py` (if your configuration file has the name *conf_register_gaze_roi.py*) or the command
`python register_gaze_roi.py <configuration_file>` (if you want to use a configuration file with a different name).

3.2.5 visualize_registered_gaze_roi.py

To visualize the output of the script from Section 3.2.4 (i.e., where the gaze is pointed at) in video files, use the script *visualizations/visualize_registered_gaze_roi.py*. The script visualizes the point of the fixation together with the measurement errors in x and y direction (a diamond shape), and also the label of the fixation (left eye/right eye/nose/mouth/face/other) is added to the videos. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.2.6 visualize_fixations_specific_person.py

If you simply want to visualize fixations in a video file (i.e., you only have fixation data and their respective videos) for a specific test subject without the need for OpenFace features, you can use the script *visualizations/visualize_fixations_specific_person.py*. The script visualizes a red cross into the point of the fixations. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.3 Machine learning

This section gives descriptions about different scripts that can be utilized for applying machine learning to train models to predict where the gaze is pointed at based on extracted features. The weights of the pretrained convolutional neural network model are in the file *best_model_temporal_adv_90_frames.pt* which is located in the GitHub page of the pipeline (https://github.com/infant-cognition-turku/gaze_dynamics). The present implementation uses a

simplistic approach for machine learning in which the model is trained to learn a binary prediction whether a person is looking at a face or not based on OpenFace features.

3.3.1 compute_train_test_feats.py

The script *machine_learning/compute_train_test_feats.py* combines the information of OpenFace features (Section 3.2.1) and gaze directions (Section 3.2.4), and turns this information into a training and test set for machine-learning algorithms. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script. With its default values, each training/test sample consists of 90 frames (0.75 seconds with 120 FPS) with 25 features (head location and orientation, action units, eye aperture) in each frame, together with the binary label (looking at face/non-face) as the ground truth label.

3.3.2 train_ml_model.py

The script *machine_learning/train_ml_model.py* trains a convolutional neural network model using the features that were computed using the script in Section 3.3.1. The model is built using the PyTorch deep learning library. To use the script, make changes (if necessary) to the hyperparameter settings in the file, and then run the script.

3.3.3 compute_ml_model_score.py

The script *machine_learning/compute_ml_model_score.py* computes the model prediction score (more confident the model is about correct predictions → higher score) and the prediction accuracy (selecting the class with a higher output probability as the predicted class) of the trained convolutional neural network model on the test set. To use the script, make changes (if necessary) to the hyperparameter settings in the file, and then run the script.

3.3.4 compute_ml_model_prediction_scores.py

The script *machine_learning/compute_ml_model_prediction_scores.py* computes the model prediction score for each test subject using the trained convolutional neural network model (a separate score all the videos for each test subject). A higher score means that the child deviates from the model less, while a lower score means that the child deviates from the model more. In addition, the script creates visualization plots for the scores of each video. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.3.5 visualize_temporal_ml_model_predictions.py

For each input video, the script *visualization/visualize_temporal_ml_model_predictions.py* creates a visualization video which shows the output of the trained convolutional neural network model when the model is applied to the OpenFace features of a video file. This visualization includes both the predicted class (gaze at face/non-face) and the confidence of the model. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.3.6 create_model_predictions_for_plot.py

The script *visualization/create_model_predictions_for_plot.py* creates a file containing the model output predictions for the probability of looking at the face for a set of videos. This file is used for visualizing the output of the trained convolutional neural network model and the real looking-at-the-face probabilities with the script *visualization/create_face_prediction_plot.py*. For convenience (if the model prediction output is used for some other purposes), this script has been made separate of the script *visualization/create_face_prediction_plot.py*. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.3.7 create_framewise_gaze_statistics_for_plot.py

The script *visualization/create_framewise_gaze_statistics_for_plot.py* creates a file containing the real looking-at-the-face probabilities of the data population. This file is used for visualizing the output of the trained convolutional neural network model and the real looking-at-the-face probabilities with the script *visualization/create_face_prediction_plot.py*. For convenience (if the real looking-at-the-face probabilities are used for some other purposes), this script has been made separate of the script *visualization/create_face_prediction_plot.py*. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.3.8 create_face_prediction_plot.py

The script *visualization/create_face_prediction_plot.py* creates visualization plots containing the output of the trained convolutional neural network model and the real looking-at-the-face probabilities utilizing the output of the scripts of Sections 3.3.6 and 3.3.7. To use the script, make changes (if necessary) to the configuration settings in the file, and then run the script.

3.4 Feature selection

This section gives descriptions about different scripts that can be used to determine the importance of the features of the trained convolutional neural network model. In each feature selection process, a feature that is “turned off” is randomized.

3.4.1 feature_selection_backward.py

The script *feature_selection/feature_selection_backward.py* ranks each feature's importance using the backward feature selection method. The output of the feature selection method is saved into a text file. To use the script, make changes (if necessary) to the different settings found in the file, and then run the script.

3.4.2 feature_selection_forward.py

The script *feature_selection/feature_selection_forward.py* ranks each feature's importance using the forward feature selection method. The output of the feature selection method is saved into a text file. To use the script, make changes (if necessary) to the different settings found in the file, and then run the script.

3.4.3 feature_selection_random_subset.py

The script *feature_selection/feature_selection_random_subset.py* ranks each feature's importance using the random subset feature selection method. The output of the feature selection method is saved into a text file. To use the script, make changes (if necessary) to the different settings found in the file, and then run the script. Please note that the algorithm runs indefinitely, so the output file should be checked every now and then whether the feature importance order has converged.

APPENDIX A: ADDITIONAL NOTES

1. It is mentioned a couple of times in this document that OpenFace is only able to analyze one face at a time. By examining the GitHub page of OpenFace more closely, you can notice that there is also a possibility to change the OpenFace run command so that multiple faces in a video can be analyzed at the same time. However, as stated in the GitHub manual, OpenFace does not provide any guarantee that face IDs remain consistent throughout the entire analysis. This means that it might happen even multiple times during the same video that face IDs switch places, which makes multi-face detection unreliable. Through thorough testing by the author of the present manual, face IDs switching places is unfortunately very common with naturalistic videos, especially in cases when there are a few face misdetections here and there. Therefore, the present processing pipeline has been made so that the multi-face detection of OpenFace is not considered to be reliable. Please note that this might change in the future if OpenFace is updated.
2. The present pipeline can be directly used for creating training and testing data for a neural network model, training the model, and then testing the model's performance. However, the network structure and the hyperparameters of the training procedure have been selected to be optimal for the data the pipeline was developed with. This means that you will get some results when running the scripts of the present pipeline with some other data without any expertise in machine learning, but in order to improve these results there is a need for an expert in machine learning. This is due to the fact that in order to improve the obtained results with some other data, the user needs to know, e.g., how to experiment with different input features, which models can be applied to the data, how to experiment and fine machine-learning algorithms, and how to select optimal loss functions and optimization algorithms.