School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc (Eng) “Creative Computing”

Project Title:
Lold.js – Multimodal Lightweight Online Laughter Detection

Supervisor:
Julian Hough

Student Name:
Alberto Morabito

Final Year
Undergraduate Project 2019/20

Queen Mary
University of London

Date: 07 April 2020

Abstract

Automatic laughter detection has been a topic of interest in the past decade, both in its unimodal and multimodal form. Its application ranges from Human Computer Interaction (HCI) and social signal processing to medicine and psychology. There are not many works in the literature that try to detect laughter from audio visual signals using mobile hardware. This project is an attempt at achieving multimodal, lightweight, real-time laughter detection for mobile devices. It aims at being robust and accurate in real-life situations. For this matter, multimodality as a mean of improving robustness is tested and evaluated. Moreover, two other factors that could contribute to improve the robustness are tested: audio features selection and dataset manipulation by artificially adding noise. There are three main outcomes for this project: a lightweight machine learning model that detects laughter from audio signals, developed from the ground up; a multimodal, real-life dataset, as opposed to ones recorded under lab conditions, which is used as a benchmark to effectively test the robustness of the model; a developer friendly JavaScript API which gives access to unimodal or multimodal laughter detection. This is built on top of said model in conjunction with an existing emotion detection model, which uses video signals.

Contents

Chapter 1: **Introduction**

1.1 Background

In the context of social interactions, laughter plays an important role. As a non-verbal cue, it carries a wide spectrum of emotions – most often associated with positive feelings, it can also assume different connotations (Szameitat et al., 2019). Thus, it can be used to infer important information about an individual's emotional status. Automatic laughter detection has been a topic of research for over a decade and has been used in multiple ways: to estimate the emotive state of an individual (Zhalehpour et al., 2016; Petridis and Pantic, 2010; Kantharaju et al., 2018; Hagerer et al., 2018; Di Lascio et al., 2019), to label and organize multimedia (Pantic and Vinciarelli, 2009; Cai et al., 2003), to recognize the speaker's identity (Knox and Mirghafori, 2007), to improve speech recognition by ignoring laughter episodes (Kennedy and Ellis, 2004), to diagnose and evaluate depression (Navarro et al., 2014; Navarro et al., 2019), to improve virtual agents (Tatsumi et al. 2014). Aside from academic studies, there are examples of companies that use, amongst other things, video data to determine users' emotions. Affectiva's Emotion Analytics Dashboard (AffDex) is a solution offered by Affectiva (McDuff et al., 2016). AffDex aims to help improve market research by capturing and analysing customers' reactions to ads. Another example of a similar technology is iMotion, which outperforms AffDex as shown by Stöckli et al. (2018). On a more experimental level, there are cases of real-time automatic detection of laughter in the wild during social interactions. These focussed on presenting the collected data on a device screen (Flutura et al., 2016; Hagerer et al., 2017). The same idea has been applied in meeting scenarios (Kennedy and Ellis, 2004; Knox and Mirghafori, 2007). In this case, the data could either be displayed in real time or it could be stored and analysed later, uncovering patterns that could improve the social interactions. Another valid use-case of automatic laughter detection is enhancing apps that monitor health. As mentioned, laughter can assume different connotations and convey different emotions. Thus, its automatic detection could add valuable information to the overall evaluation of the users' emotive state. This data could then be presented in an informal way, as part of already existing metrics available in common health tracking apps, or even in medical diagnosis as shown by Navarro et al. (2019). One last use-case not already mentioned, is making use of laughter detection for purely recreative apps. Microsoft used laughter detection at the national comedy centre, in an AI-assisted "You laugh, you lose" contest (Roach, 2018).

1.2 Problem Statement

To the best of my knowledge, there are no free-to-use JavaScript high-level API that provide real-time multimodal laughter recognition. This precludes developers who do not have experience with machine learning to either integrate laughter detection in their existing projects to enhance them or create a new application that exploit laughter detection.

As mentioned in the Background section, one of the use cases can be improving health monitoring apps. These have seen a rise in interest the past years (Research2Guidance, 2017; Ali et al., 2016). The possibility of easily integrating this model with existing apps can bring tangible value, as laughter is an important source of information about a person's emotional state.

Although there have been attempts to create a robust real-time laughter detector to use in mobile/wearables (Di Lascio et al., 2019; Hagerer et al., 2017), the area is arguably

underexplored. Hence, contributions to the subject can have a concrete impact in the scientific community.

1.3 Aims

The aim of the project is to investigate real-time, multimodal laughter detection. The exploration will focus on four main points: detection improvements, if any, achieved with multimodality; detection improvements, if any, by artificially adding noise to training data; robustness of laughter detection in real life scenarios; optimised performance for mobile devices.

Multimodality has been proven to be a better approach than unimodality (Poria et al., 2017). As part of the project, empirical testing will be conducted to evaluate if the above statement still holds true in real life conditions, using a multimodal dataset which contains realistic examples as opposed to lab-conditions examples.

Artificial manipulation of the data used to train the model, specifically by adding noise, will be evaluated to assess whether a possible performance improvement in noisy environments is achieved.

As the objective is to implement an API that can be used in real-life application, a focus on the robustness of the model in in the wild scenarios is needed. As demonstrated by Hagerer et al. (2018) it is possible to achieve high accuracy in non-lab conditions, using wearable hardware. However, the paper did not present empirical results on this matter, perhaps for the lack of a readily available multimodal, real-life laughter dataset. Publicly available datasets are for the majority recorded under lab conditions – i.e. in absence of noise, with a head-mounted microphone with good audio quality. For this matter, a small real-life settings multimodal laughter dataset will be created and used as a benchmark for the models' performance. The new multimodal dataset will contain a mix of low-quality recordings, various background noise and multiple speakers, trying to mimic natural conditions as close as possible.

The model will run in the browser thus it will be naturally cross-platform. This project aims to also target mobile devices, as any mobile device that has access to a browser will also be equipped with hardware to capture audio and video. To support mobile devices, some considerations have to be made: on the software performance side, TensorFlow.js can exploit the GPU power via the browser. This will give more lenient constraints to the model's computational expensiveness. On the hardware side, different microphones' sampling rates will be considered. Publicly available audio-visual databases are often recorded with a higher resolution than mobile devices can afford, thus some pre-processing of the training data will be needed.

1.4 Objectives

The objective is to develop a robust, lightweight, real-time, multimodal laughter detection machine learning model and wrap it in a high-level JavaScript API, which builds on top of TensorFlow.js. This will provide developers with an easy to use tool that can either improve existing apps or be used in novel ways. The model should run smoothly on mobile devices and achieve good accuracy in real life scenarios.

Along the API development, two different evaluations will be conducted: one related to how multimodality can improve robustness in real life conditions, and another one on

how artificially adding noise to a dataset can have the model generalise better for previously unseen data.

As the timeframe for this project is relatively small, only one model will be implemented from the ground up. Specifically, a unimodal laughter detection model that uses audio features. For the video features, I will integrate an existing open-source API named "face-api.js"[1], which provides a rich interface and various models, including an emotion detection from facial expressions.

As mentioned in the Problem Statement section, the outcome of this project will be something novel, as there are no JavaScript API that aim to do multimodal laughter recognition. Despite the fact that I will build it on top of literature findings, there is a tangible effort to be made in building a reliable laughter detection model from scratch, without, for instance, transfer learning. What is more, the API and the model will be developed using different technologies: a good amount of work will be required to have all pieces of the technology stack fit in together. Finally, the multimodal real-life dataset will provide a previously unseen benchmark to test models' robustness. there are a set of challenges which will be discussed in the Research Questions section.

1.5 Research Questions

One of the main questions that need to be answered is which set of audio features achieves good accuracy with modest computational power.

This project aims to run in real-time on mobile devices. The computational power of said devices is limited. The model needs to give predictions at a high enough rate to run smoothly in real time. The computational cost of a model, both during training and deployment, depends on the number of features that needs to be extracted and fed to the model and the complexity of the model itself. For these reasons, finding the minimal subset of features that achieves good accuracy is key, along with a simple yet accurate prediction yielding neural network architecture.

Another important question is how a model can be trained to generalise well in real life scenarios, considering how different hardware will capture different audio data.

As mentioned in the Aims section, the quality of audio data the model will be fed during deployment will be different from the one it will be trained on. Specifically, the training data will have a higher sampling rate. This can lead to poor generalisation of the model. To counter this problem, many techniques have been used. In this project I will explore the usage of down-sampling (Akhtar et al., 2018; Bedoya and Falk, 2016).

Next issue to consider is that the training data is being recorded under lab conditions. but the model should perform well in real-life scenarios. This means that the model should account for noise, the possibility of more than one person laughing simultaneously, speech and laughter overlap and silence. For this matter, I will refer to the work of Hagerer et al. (2018) and integrate Voice Activation Detection, which was shown to be a robust solution to the problem of noisy settings and frame loss due to flaky hardware. On the same matter, the performance of an artificially noisy dataset will be explored. The noisy dataset will be created by overlapping different noises (e.g. wind, crowd) onto the samples of the original dataset. Two models will then be trained on both noisy and regular dataset. These will then be evaluated to assess whether an improvement was made.

[1] Available at: https://github.com/justadudewhohacks/face-api.js/ [Accessed 14 March 2020]

Another key question is related to modality fusion. This was proven to be a better choice over unimodality (Poria et al., 2017). This project will put this statement under further testing. For this matter, a real-life scenario dataset will be created, such that it contains annotated laughter episodes which are not recorded under lab conditions. This will serve as a benchmark to test whether the multimodality enhances robustness in in the wild scenarios compared to a unimodal approach.

1.6 Report Structure

Section 2 presents a review of the literature findings, with a focus on laughter recognition from audio, as that is the biggest self-contained part of this project. Section 3 focuses on the design of the API, the approach to train and build the model and the creation of the multimodal dataset. In section 4 details of the implementations are discussed, both for the machine learning model, the JavaScript API, the noisy dataset, and the demo API client. Section 5 presents evaluations methods and results for the project. Lastly, in section 6, conclusions are presented.

Chapter 2: **Literature Review**

2.1 Laughter detection

2.1.1 From Audio

2.1.1.1 Evolution of models over the years

The oldest attempts in automatically detecting laughter typically relied only on audio features. Older works like the Kennedy and Ellis (2004) used Support Vector Machine (SVM). Better results were achieved by using Hidden Markov Models (Campbell et al., 2005). Using Gaussian Mixed Model (Truong and Van Leeuwen, 2007) was another step towards improving robustness of laughter detection. The most recent, better performing model is based on Recurrent Neural Networks (RNN), using Long Short-Term Memory (LSTM) layers (Hagerer et al., 2017).

The RNN LSTM architecture presented by Hagerer et al. (2017) consists in fully connected recursive layers, where the outcome depends on current and past inputs (LSTM) – and future inputs for Bidirectional LSTM (BLSTM). Its first layer has 140 inputs, followed by two hidden LSTM layers, with respectively 90 and 60 cells and the final layer with 4 regression outputs: “Degree of voice activity”, “Overlap”, “Male” and “Female”. This model was adapted and used by Hagerer et al. in wearables (2017) and later combined with a Voice Activity Detection to further improve its accuracy in real-life scenarios (Hagerer et al., 2018).

2.1.1.2 Audio features

Pre-processing raw audio data is important. By extracting meaningful features, a model can learn patterns and interpret the data in a more accurate way, making feature extraction a key part of developing a successful model.

2.1.1.2.1 Mel-frequency cepstral coefficients

Mel-frequency cepstral coefficients (MFCCs) are recognized as the state-of-the-art for Automatic Speech Recognition (ASR) and have been widely used for over a decade (Akhtar et al., 2018; Hagerer et al. 2017; Gosztolya et al., 2016; Gupta et al., 2016, Kennedy and Ellis, 2004).

Spectral features like MFCCs are not the only ones considered. In fact, MFCCs can be used in conjunction with prosodic features to improve the accuracy of the model (Gupta et al., 2016; Akhtar et al., 2018).

2.1.1.2.2 Short-Term Energy and Zero-Crossing Rate (ZCR)

Short term energy and ZCR are the older approaches at separating voiced from unvoiced segments of an audio signal (Bachu et al., 2008). The energy of the signal is the sum of the intensity of each sample for a given timeframe. The ZCR is the count of how many times the signal crosses the 0. These features are relatively easy to compute, but do not perform well on signals with non-static background noise or signals where the noise is predominant (high signal-to-noise SNR ratio) (Ma and Nishihara, 2013).

2.1.1.2.3 Spectral flatness

Spectral flatness is an improvement over the ZCR measure, in that it is robust to noise (Ma and Nishihara, 2013). As the name suggests, the spectral flatness is a measure of the homogeneity of the spectrum. The spectrogram of white noise is very flat, with all

frequency ranges having a similar amount of energy (high spectral flatness). On the other hand, low spectral flatness indicates a non-uniform signal, which is a fairly accurate match to what speech is.

2.1.2 Multimodality

Results in the literature show how considering more than one modality when detecting laughter can improve the accuracy of the model (Poria et al., 2017). The two modalities that have been used in conjunction the most are audio and video (Reuderink et al., 2008; Petridis and Pantic, 2009; Escalera et al., 2009; Petridis et al., 2013; Lingenfelser et al., 2014; Ito et al., 2015; Bedoya and Falk, 2016; Turker et al., 2017; Kantharaju et al.,2018; Akhtar et al., 2018). Other than face features analysis, body movement (Flutura et al., 2016; Niewiadomski et al., 2016) or a combination of physiological signals (Tatsumi et al., 2014; Di Lascio et al., 2019) have also been used to detect laughter with good accuracy.

2.1.2.1 Modality fusion

Different modality fusion approaches have been documented in the literature. The most common one is to generate a single feature vector by concatenating the feature vectors of all modalities. The fusion layer is commonly a Support Vector Machine (SVM) (Di Lascio et al., 2019, Akhtar et al., 2017). An important consideration to make is that two poor unimodal models will not perform better when fused, as Poria et al. (2017) suggested.

2.1.3 Mobile devices

Attempts to implement laughter recognition models in either mobile or wearable devices were made only recently. In the work presented by Hagerer et al. (2018), only one modality (audio) is used. Di Lascio et al. (2019) fused two modalities, focussing on sensor readings that are easier to extract in in-the-wild scenarios, rather than audio-visual features.

A proven to be robust model for offline speech overlap detection is based on Bidirectional Long Short-Term Memory (BLSTM) and Recurrent Neural Networks (RNN) (Hagerer et al., 2017). This has been effectively used for real-time laughter recognition in mobile devices (Hagerer et al., 2017), but using LSTMs rather than BLSTM, as the latter can only work offline on complete data streams (as it takes into account previous and future inputs, thus needs to have access to the entirety of the data). The LSTMs model has been improved in another work of Hagerer et al. (2018), by integrating a voice activation detector (VAD). This allows the model to discern not only between laughter or no laughter, but also speech, general activity, and silence.

2.1.4 Datasets

The complete list of datasets, used individually and sometimes in conjunction, is as follows: SSPnet Vocalisation Corpus (SVC) (Salamin el al., 2013), Switchboard (SWB) (Godfrey and Holliman, 1993), MUSAN (Snyder et al., 2015), TEDLIUM (Rousseau et al., 2012), LibriSpeech (Panayotov et al., 2015) and AudioSet (Gemmeke et al., 2017). These will be further discussed in the Analysis and Design chapter.

Chapter 3: **Analysis and Design**

3.1 Laughter Detection Model

3.1.1 Training data

The selection of the dataset on which to train the model is important. As a general rule, more data results in a better performance of the model. The quality and correctness of the annotations also plays an important role. For this project, datasets containing naturally occurring laughter episodes are preferred over ones in which laughter is acted. This is because this project focuses on real life scenarios. On the same note, datasets recorded using a telephone are preferred, as they closely mirror the data that will be fed to the model during deployment – In essence, training the model on low resolution audio should have the model generalise better.

Different public databases are analysed to find the one that would suit in the wild scenarios the most – e.g. with noise annotations, speech overlap, per word annotations. The latter kind of annotation is useful to develop an accurate VAD model.

3.1.1.1 SSPNet Vocalization Corpus (SVC)

The SSPNet Vocalization Corpus (Salamin el al., 2013) could be a great candidate dataset to train the proposed model with, as it is a collection of phone calls recording. For this reason, there is no need to down-sample the data, which is already at 16Khz. The full dataset includes 2763 audio clips, where each clip is 11 seconds. Each clip has at least one episode annotated but can have more. The two classes of utterances are “laughter” and “filler”. Laughter annotations make no distinctions between voiced/unvoiced laughter, laughter with speech overlap, etc. In essence, any laughter episode is labelled as “laughter”. Filler annotations include utterances such as “uh”, “ehm” and so forth, without making any further distinction. The dataset comprises 120 unique subjects, both male and female. It is an unbalanced dataset, with 2988 filler and 1115 laughter annotations. Most of the initial models were trained on this dataset, as this was the first readily available dataset. However, the SVC is relatively small (only 8.5 hours of recordings) and the speaker voice is very loud and central. Furthermore, the annotations available are not enough to build a VAD system. For these reasons, the dataset proved not to be enough to train a robust model that performs well in real-life conditions. Evaluation of models trained on this dataset are presented in the Evaluation chapter.

3.1.1.2 TEDLIUM and LibriSpeech

To implement a Voice Activity Detection system, new datasets were needed. The search criteria for these was that they had to include annotations for speech and noise or silence. This is so that the model could learn to detect when speech versus silence or noise was taking place in the signal. For this matter, two datasets were tried: TEDLIUM and LibriSpeech.

TEDLIUM is a collection of TED talks in NIST sphere format (SPH). It contains 2351 samples, for a total of 452 hours of audio. There are 2351 aligned automatic transcripts in STM format. These also include noise and silence. LibriSpeech is a set of 1000 hours of English speech taken from audiobooks. Note that the annotations only included the starting time of the annotated transcription as opposed to have per word timing annotations.

Neither of these two datasets contained any laughter annotation, thus a laughter detection model could not be trained. To overcome this, samples from SVC were added to the pool of samples used to fit the model. However, the difference in how samples were recorded for SVC and TEDLIUM/LibriSpeech resulted in biased predictions (over-fitting) and consequently poor performance of the model. Thus, the idea of using two different datasets in conjunction was no further pursued.

3.1.1.3 Switchboard

The final attempt in creating a robust model was done using the Switchboard dataset. This dataset consists of 260 hours of telephone conversations with laughter, silence and per word annotation. However, the quality of the recording is lower than average and there is a constant static background noise. The sample rate is only 8kHz – as opposed to at least 16kHz for the SVC and other datasets. Nonetheless, this dataset has much more data and precise per word annotations which also include laughter and silence. Having much more data and detailed annotations resulted in models trained with SWB performing better both on validation data and on the multimodal dataset, as shown in the Evaluation chapter. The model trained on this dataset is the one implemented in the Lold.js API.

3.1.2 Technology stack

3.1.2.1 Python

The original idea was to have a one-sided stack. In essence, build and train the model in JavaScript and use the same language to develop the API. This has several advantages: less moving pieces, so less room for errors; identical settings for training and testing of the model, which is harder to achieve if the development of the model is done with a different framework, language, and different resources than the deployment's – identical settings are key for the model to predict as it is meant to; less technologies to learn, which is a reasonable thing to keep into account, given the limited timeframe for the project.

TensorFlow.js allows to build a model and train it in the browser, closely mirroring the workflow using Python. However, in the JavaScript world there is much less flexibility when it comes to side tasks such as feature extraction. For instance, to the best of my knowledge, there exists only one framework that extracts MFCCs in real-time, Meyda.js[2]. On the other hand, Python has a wider collection of libraries that would enhance and speed up the development of the model. Thus, the stack settled as follows: Python is used to build, fit, and evaluate the model. TypeScript is used to build the API, use the audio model after its conversion to TensorFlow.js, implement modality fusion and a demo app.

On the Python side of the stack, Pandas and NumPy are often used to sanitize, parse and deal with annotations – other times vanilla JavaScript is used. Audio features are extracted with the aid of the framework Librosa. The model is created using Keras. Once built, the CLI utility "tensorflowjs_converter" is used to convert the model to TensorFlow.js. Running the conversion creates a `model.json` and a `shard.bin` containing the weights of the model, which can then be used in the TypeScript API.

The Python side stack is available at this GitHub repository https://github.com/AlbertQM/laugh-model

[2] Available at: https://meyda.js.org/ [Accessed 18 April 2020]

3.1.2.2 TypeScript

The other side of the stack, covering the development of the API is done in TypeScript – using the latest ECMA syntax. TypeScript is a superset of JavaScript which solves many of the common JavaScript problems by adding strict type checking and more. Although TypeScript requires extra configuration to be put in place and it adds new syntax which slows down the development, it helps to catch most JavaScript errors early. What is more, once familiar with the new syntax, TypeScript in fact smooths the development, thus the decision to adopt it.

The project is converted to a npm package early on, as it will make use of many Node.js packages (e.g. Meyda.js). TypeScript files need to be compiled to JavaScript before they can run in the browser. To then serve the JavaScript files, Webpack is used. Webpack is a static bundler for JavaScript application: it takes and entry point (a .js file, which we compiled from a .ts file), builds a dependency graph and outputs a single bundle.js, which can then be easily served locally or deployed. More details on the setup can be found in the Implementation chapter.

This side of the stack focuses on: correctly implementing face-api.js; correctly implementing the audio model; get access to audio-visual signals from users' hardware; extract features in real-time; fuse the predictions of the two models; implement the API to give access to both models (individually or in conjunction). The TypeScript application also takes into account optimisation, in an attempt to have mobile hardware run at a reasonable frame rate.

During development, a simple API client is implemented. This helps to test the performance of the model - both its robustness and memory/computational performance on mobile. It will also be used to showcase what has been developed in this project. The demo API client is available at https://albertqm.github.io/laugh-api.js/

The TypeScript side of the stack can be found here https://github.com/AlbertQM/laugh-api.js

3.1.3 Neural Network Architecture

The model's recurrent neural network architecture is as follows: an input layer that receives the features extracted from the raw audio data, then two hidden layers (with 90 and 60 LSTMs cells respectively, the same number of LSTM cells used by Hagerer et al. (2017)) and an output layer with "speech", "laughter" and "silence". There is 0.5 dropout after each hidden layer. ReLU is used as an activation function for the hidden layers, and SoftMax for the output layer, as this is a multi-class classification problem. The loss function used is the categorical crossentropy. The optimiser used is ADAM.

The features in the input layer of the final version of the audio model are chosen so that both sides of the stack, Python's Librosa and JavaScript's Meyda.js, are able to extract the same values given the same signal.

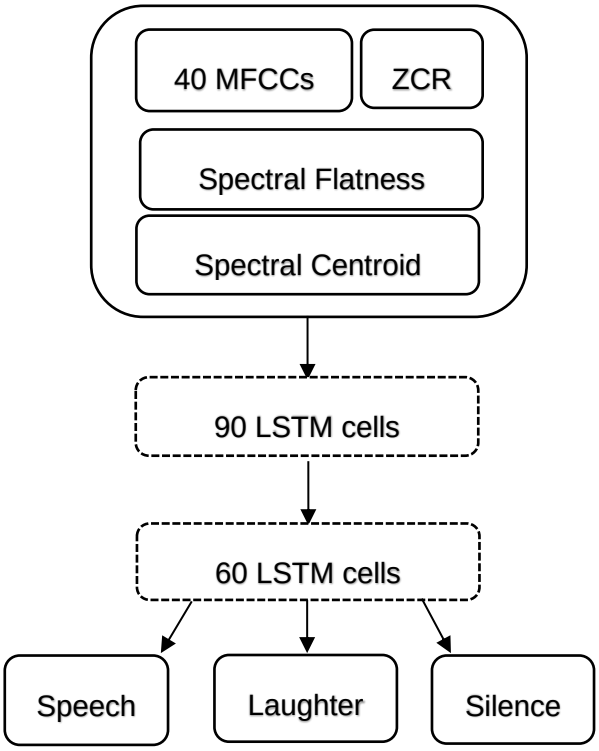

Figure 1. Illustration of the RNN Architecture. Input layer has 43 audio features. Hidden layers are dashed. All layers are fully connected. The model outputs the confidence for three classes.

This RNN configuration is relatively shallow but has 9603 configurable parameters. In general, the deeper the model gets, the more flexible it gets, too – flexibility also means better performance, if overfitting is avoided (this could be done with a regularisation technique, e.g. with dropout, which was used for this project). However, adding more layers also results in more configurable weights that need to be computed during test time. In order not to make the computation of a prediction too costly, given the model aims at running smoothly in mobile devices, a decision is made to stick with the configuration presented in figure 1.

3.1.4 Audio features

The input to the model consists of four different types of features: the first 40 Mel-frequency cepstral coefficients, which are the intensities of the first 40 Mel-bands filters; the zero-crossing rate (ZCR), which captures the number of times the signal crosses the zero value in the time window considered. This measure is useful to detect voice activity (Bachu et al., 2008) which can in turn avoid detecting false positives laughter episodes; the spectral centroid, which gives an indication of the brightness of a sound; the spectral flatness, which can be used to determine how noisy a sound is. A pure sine wave has a flatness close to 0, whereas white nose has a flatness close to 1. The latter feature is added in an attempt to improve robustness of detecting voice activity in noisy conditions, following Ma and Nishihara (2013) work.

One of the additional challenges faced during the project is the hard to achieve consistency of the features extracted in both sides of the stack. As mentioned, features extracted in Python and JavaScript must be exactly the same for the model to work. Many libraries are available for Python that allow extracting any kind of features from any kind of signal. In JavaScript, the choice is very limited. In fact, only one framework is able

to extract MFCCs in real time: Meyda.js. After a personal contribution to said library[3], it was possible to correctly extract the 40 MFCCs to match the ones extracted with Python's Librosa.

To test that the features used to train the model (extracted with Python) are the same as the ones fed to the model during test time (extracted with Meyda.js), a manual experiment was made. A sample file was recorded with a microphone. Features were then extracted from this recording with both parts of the stack and the resulting arrays were observed manually.

Firstly, the MFCCs were compared: these matched in both sides of the stack, thus were used in the deployed version of the audio model.

An attempt at adding an extra 50 features (for a total of 93) based on the Mel spectra coefficients was also made. These extra features improved the model's performance, as shown in the Evaluation chapter. However, the two sides of the stack did not compute the same values for these features, which were hence discarded in the deployed model.

3.1.5 "Noisy" SVC dataset

Sounds recorded with mobile hardware, in noisy real-life conditions, are different from the clean lab conditions datasets. Some experimentation was done by artificially adding noise to the SVC dataset, in an attempt to see whether this technique would have the model generalise better in in the wild situation. Five different types of noises were manually chosen from the MUSAN dataset. These were: wind noise, white noise, crowd noise, static noise, and quiet empty room background noise. These noises were chosen as they are likely to be present in in the wild situations. Find implementation details in the Implementation chapter. Results of the difference between SVC and noisy SVC are presented in the Evaluation chapter.

3.1.6 Real-life multimodal dataset

To effectively evaluate how multimodality affects the prediction's robustness in real life conditions, the afore mentioned noisy SVC datasets was not enough, as it only covers testing the unimodal (sound) model. Thus, a multimodal laughter dataset was created by extracting short clips from YouTube and manually labelling them as "laughter" or "other". The clips were chosen to be diverse and noisy: multiple speakers, different camera angles, shaky or low-quality video or audio. A total of 100 clips were created, half of which included laughter episodes and half did not. This evaluation dataset is small, as the process to build it was not scalable. Clips from YouTube8M[4] could also be used. YouTube8M is a collection of human labelled YouTube clips, containing almost 240.000 labelled samples. Only samples labelled as "laughter" should be taken into consideration to expand the real-life multimodal dataset, plus any other category to have a balanced dataset.

3.2 Lold.js - TypeScript API

3.2.1 API Architecture

The TypeScript API provides a simple interface to implement lightweight laughter detection either in a multimodal or unimodal way. The latter is made available to cover the scenario in which users did not grant access to video or audio stream, or the signal is not available. For instance, in a work meeting scenario it is very likely to have

[3] Available at: https://github.com/meyda/meyda/pull/309 [Accessed 14 March 2020]
[4] Available at: https://research.google.com/youtube8m/ [Accessed 14 March 2020]

microphone in the room that could be used to detect laughter, but not a webcam. Thus, unimodality is a valuable option.

Lold.js main component is a JavaScript class. This was designed to hide all the complexity related to the machine learning models, so that any client only needs to care about getting audio and/or video signal from the browser to access the models' predictions. The latter is done with getter methods.

Lold class's constructor has three arguments: the video signal, the audio signal, and an optional options object. Each client implementing the API needs to provide either video, audio or both during initialisation. The options object offers a way to personalise the behaviour of the models. The first option is "Prediction mode": this defaults to "multimodal", but can take two other values, "audio" or "video". Using this option is useful to turn off the predictions of one of the models, in order to save computational power. The other configurable option is the video source type: this can either be "stream" or "video". This is because the browser AudioContext needs to be fed a different type of object depending on whether the audio channel is embedded in the video element (this is the case for offline video), or if they are two separate channels (this is the case for e.g. webcam live feed)

The API exposes four different methods. The first one, "loadModels", will load both models so that they can be used to make predictions. Then, there is a "startPrediction" and a "stopPrediction": these are used to begin feeding the signal to the models. Clients can stop the prediction when not needed, saving CPU and battery (if running on a mobile device). Lastly, the getter method "getPrediction". This will return the prediction of both models at the frame in which it is requested.

3.2.2 Modality fusion

By design, the fusion of the predictions of the audio model and the video model is not part of the API. The modality fusion is done by the client at the JavaScript layer. The client gets both models' predictions values (audio and video) and can decide how to fuse them. For the client developed as part of the project, the modality fusion is done with a threshold: for a given frame, if both predictions are above 65% confidence then predict "laugh", otherwise predict "other".

The reason for this is mainly the relatively small timeframe for this project. Several alternatives could have been explored, such as multi-task learning, or training a meta-learner to fuse the two models' outputs, or adding another layer in the model.

Chapter 4: **Implementation**

This project's implementation consists of three main parts. The largest chunk is a machine learning model that detects laughter from audio signals, build and trained from scratch. This was the first part implemented. Then, the Lold.js API was developed along with a client which implemented it. The latter was developed to test the audio model on new data (e.g. live webcam feed), and later to test multimodality. In fact, the client implementation was done incrementally. Initially the API focussed only on the laugh detection model. Once implemented, this was tested on the browser. These tests then fed back into the other side of the stack, where the re-tuning and re-training of the model happened. Once the audio model performance reached a high standard, the face-api.js integration was developed; upon completion, the modality fusion was addressed. Once all pieces were working together, a general refactor and tidy up was performed, shaving off rough edges left during implementation. Note that despite the fact that the multimodal dataset created as a part of this project is not included in the diagram below, it plays an important role for the project.

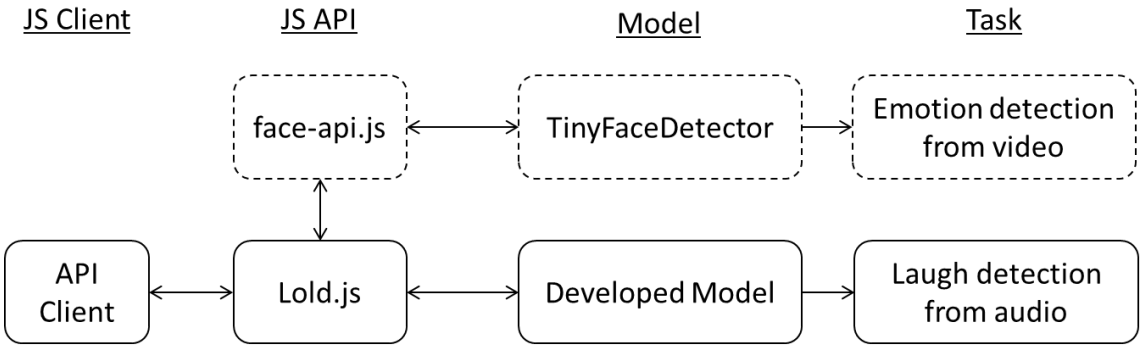

Figure 2. Diagram illustrating which parts have been implemented in this project (solid) and which parts have been re-used (dashed) and the overall structure

4.1 Laughter Detection Model

4.1.1 Pre-processing of the data

The first step in implementing the audio model is dealing with the data. Features are extracted from the datasets and used to train the model. A full list of datasets used for this project is available in the Datasets section of chapter 2.

For each dataset, some pre-processing of either annotations or data was needed. In fact, in all cases, the annotations, which come in different formats, were not ready to be parsed by the Python script created as a part of this project. To sanitise them, vanilla JavaScript was used, sometimes along with Python.

To extract features with said Python script, the annotations had to follow this format:

1. "Sample name", used to retrieve the sample file from the HDD, e.g. "Sample_001.wav"
2. "Start" and "End" of the utterance/speech event, in milliseconds. Used to extract the features from the exact timeframe in the sample.

3. “Class”, a human defined label used to train the model – This is supervised learning.

4.1.1.1 Sanitising annotations

4.1.1.1.1 TEDLIUM

The following is an example of annotations from the TEDLIUM dataset:

```
BillGross_2003 1 BillGross_2003 1002.47 1011.51 <o,f0,female> really
good combination of those two factors {BREATH} it turns out there's a
lot of powerful sun {BREATH} all around the world obviously but in
special places(2) {NOISE} where it happens to be relatively inexpensive
to place these <sil> (BillGross_2003-1002.47-1011.51-F0_F-S135)
```

Each annotation has: name of the speaker; speaker ID; name of the speaker; utterance start time in ms; utterance end in ms; speaker metadata; transcript; annotations metadata. This needs to be converted to a format which can be consumed by the script.

As we can observe from the example annotation, the interval for each speech annotation is long and not very precise. In other words, the beginning and end of the annotated episode matches with a transcript which include silence, noises, utterances, and words – i.e. multiple classes. The model trained with this dataset did not perform well.

4.1.1.1.2 SVC

The following is an example of the headers and one row of the annotations from the SVC dataset:

```
Sample, original_spk, gender, original_time, type_voc, start_voc,
end_voc, [type_voc, start_voc, end_voc,...]

S0007,F19-
R,Female,360.887,filler,5.457,5.921,filler,6.411,6.566,filler,7.975,8.
462
```

As we can see, it can have an arbitrary number of utterances in a single row – The example above has 3. The sanitisation process for these annotations includes splitting each of the possible extra annotations into separate rows. The annotations include the beginning and end of the utterance (with the precision to the millisecond), and its class, making it a great candidate to train a good model.

A vanilla JavaScript method is used to sanitise these labels. The output for the above presented example would be as follows:

```
S0007,F19-R,Female,360.887,filler,5.457,5.921
S0007,F19-R,Female,360.887,filler,6.411,6.566
S0007,F19-R,Female,360.887,filler,7.975,8.462
```

These can be used by the Python script to extract features.

4.1.1.1.3 SWB

The following is an example of three rows of the annotations from the SWB dataset:

```
sw2005A-ms98-a-0001     A.1   0.000000    0.800000          [silence]
      [silence]
sw2005A-ms98-a-0001     A.1   0.800000    1.280000          Okay  okay
sw2005A-ms98-a-0057     A.141 421.040250  421.507375        [laughter]
      [laughter]
```

Each row contains: sample metadata; speaker metadata; start of the annotation in seconds; (optional) operation on the label, e.g. deleted; outdated label; new label.

The annotations are very precise, with per word timing. They also include laughter and silence. For this reason, the model trained on them has three output classes: “speech”, “laughter” and “silence”. The precise annotations, the amount of data and having three different classes made the model trained on this dataset the most robust and precise, especially in real life conditions.

For the SWB, the annotations were divided into multiple files and folders. Thus, another pre-processing step was performed. Annotations were merged into a single pickle file, to avoid having to re-merge them together when needed – As it can take up to five hours to process them on the machine used. Lastly, the file metadata included in each row of the annotations needed to be sanitised. A Python method was defined for this purpose.

4.1.1.2 Extracting features

Once annotations are ready to be digested by the Python script, the feature extraction can begin. Librosa is the main framework used to extract the features. “Librosa.features” methods allow to easily extract all the features needed for the development of the model. MFCCs, ZCR, spectral flatness and spectral centroid are extracted and concatenated into a single list. Once extracted, the full list of features is saved in a Python pickle to avoid having to re-compute them.

4.1.2 Model training

The neural network architecture is described in 3.1.3. This was implemented with a separate Python script that uses Keras.

The input layer of the various models trained was different, as different set of features have been tried. The two hidden layers remained the same for every model. Finally, the output layer was different depending on the dataset – as the classes detected by each model are different. For SVC, the two classes available in the annotations are “laugh” and “filler”. For TEDLIUM and LibriSpeech “speech” or “noise”. For SWB “speech”, “laughter” and “silence”.

Categorical cross-entropy was always used as a loss function, as the problem was modelled as a multi-class classification where the sum of the probability of each class adds up to 1. For instance, if the model is detecting “laughter” with 80% confidence, there is only 20% confidence left to be split between “speech” and “silence” (in the case of SWB).

To make the most out of the relatively little data, cross-validation is used. Training on 80% of the data and leaving 20% for testing. Cross validation was always performed for all datasets (SVC, SWB, TEDLIUM). Each model was then trained for 100 epochs, using a batch size of 128.

After the training is complete, three plots were created: one for accuracy, one for the loss function, and a Receiving Operating Characteristic (ROC) curve with Area Under Curve (AUC) value. Finally, each model was converted to TensorFlow.js using Python’s library “tensorflowjs_converter” to be able to run it in the browser.

4.1.3 Noisy SVC

The creation of the noisy dataset was implemented as follows: for each sample of the original dataset, one of the five noises was selected at random. Each noise was parsed as a PyDub AudioSegment. Then, PyDub’s overlay method was used to overlap the noise onto each SVC clip. The length of these clip was 11 seconds, whereas the length

of the noises clips was longer. Thus, the entirety of each SVC clip could be covered by a single noise source. The resulting wav file was exported in with the same filename as the original SVC, so that already existing annotations could be used on the new dataset.

4.1.4 Real-life multimodal dataset

A small dataset was created by manually selecting videos from YouTube. These were then downloaded and annotated. The annotations include the YouTube video ID, the starting time in millisecond of the episode recorded, the duration of the clip in seconds and the label, which is either “laugh” or “other”. Once the annotations were ready, a script was used to split up the whole videos into labelled segments. The script was written in vanilla JS and uses FFMpeg to manipulate the video files. The resulting dataset contains a total of 100 samples, 50 laughter and 50 non-laughter episodes.

4.2 Lold.js - TypeScript API

4.2.1 Development environment

TypeScript needs to be compiled into regular JavaScript. A fully functional development environment is created with Webpack. It has hot-reload, file-watcher for changes and auto incremental compilation, to speed up the implementation of the project. TypeScript and Webpack settings can be seen in “tsconfig.json” and “webpack.config.js” respectively. Another reason for using Webpack is to have the code ready to be shipped once ready. In fact, a single script is set up to build and deploy the app.

To start the dev env, the following commands need to be run:

```
tsc --watch
```

This tells the TypeScript compiler to watch for file changes in the project folder.

```
npm start
```

This tells webpack to pack and serve the JavaScript files TSC had compiled.

4.2.2 API implementation

As mentioned, Lold.js provides an easy to use interface to both unimodal and multimodal laughter detection models. Clients can get started with a few lines of code. To instantiate a Lold.js class, clients need to provide a video element and an audio stream:

```
const lold = new Lold(videoEl, audioStream);
// Load video and audio models and weights
await lold.loadModels();
// Start prediction using audio-visual fetures
lold.startMultimodalPrediction();
// Get a prediction. This can then be e.g. shown to users
const prediction = lold.getMultimodalPrediction();
```

The getter method then return the prediction from one or both models. This can then be used by the client to e.g. display something on the screen.

4.2.3 Demo API client

To test the audio model, the modality fusion and the mobile performance, a demo client was built and deployed[5]. This includes a page that uses multimodal detection on a live

[5] Available at: https://albertqm.github.io/laugh-api.js/ [Accessed 19 March 2020]

webcam audio-visual feed, and another page which demonstrated the laughter detection on a videoclip.

As mentioned, webpack is used to bundle the app. Thus, the deployment is straightforward. The client app is built using Webpack's production bundling script. This produces a minified bundle which is ready for production. The npm module "gh-pages" is used to deploy the app to GitHub pages.

Chapter 5: **Evaluation**

5.1 Audio Model Evaluation

Evaluating a model can be done programmatically. During training, two evaluation metrics are extracted and plotted: accuracy, loss. Furthermore, the ROC curve for each model is calculated. These metrics together can give an indication of how well a model performs.

5.1.1 SVC and Noisy SVC

As mentioned, a noisy SVC dataset is created to determine whether artificially mixing noise could improve the robustness of the model. Following are three plots relative to a model trained on features extracted from the SVC:

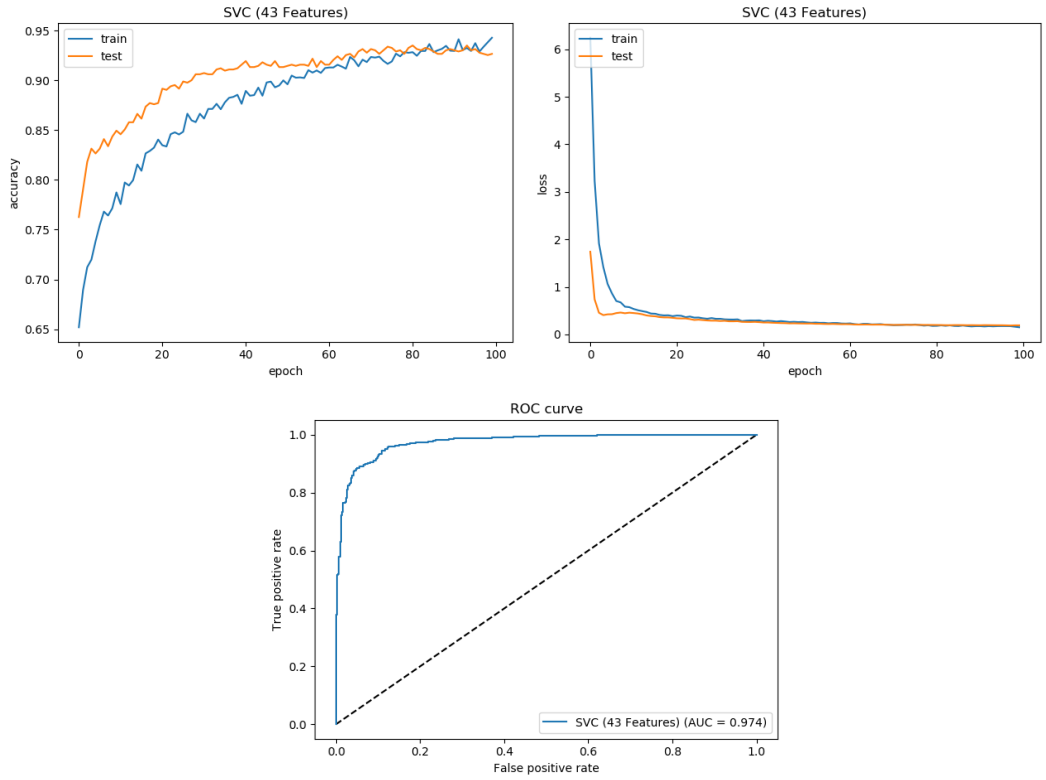

Figure 3. Accuracy, loss, and ROC curve relative to the model trained on the SVC dataset, using 43 features.

Below are plots relative to the noisy SVC:

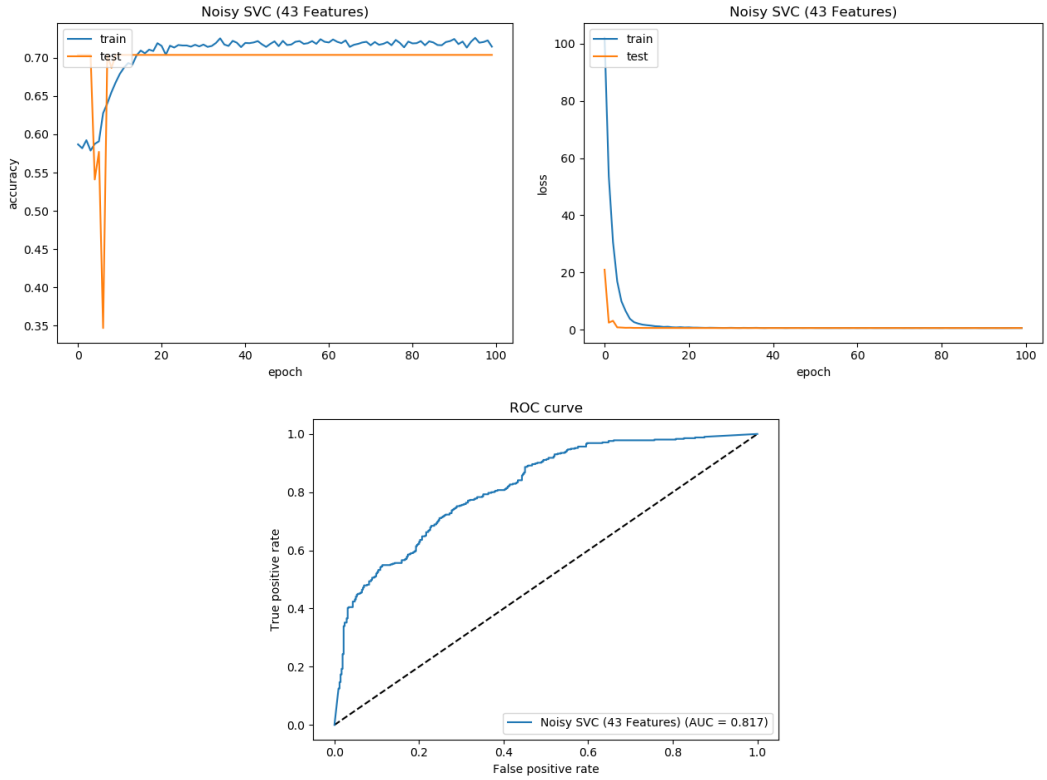

Figure 4. Accuracy, loss, and ROC curve relative to the model trained on the noisy SVC dataset, using 43 features

We can observe how the noisy SVC is performing much worse than the regular SVC on the test set. However, the former performs better when fed unseen, real-life data – e.g. a live webcam feed or using the multimodal dataset created as part of this project. This confirms what observed by Hagerer (2018).

5.1.2 93 features SVC

An additional model was trained using 93 features instead of 43 (adding MFCCs intensities and deltas). With these extra features, the performance of the model trained on the noisy SVC yielded better results on the test dataset, although not as good as the regular SVC. As mentioned, these extra features could not be extracted in the JavaScript side of the stack, thus this model is discarded in favour of the one trained with 43 features on the SWB dataset.

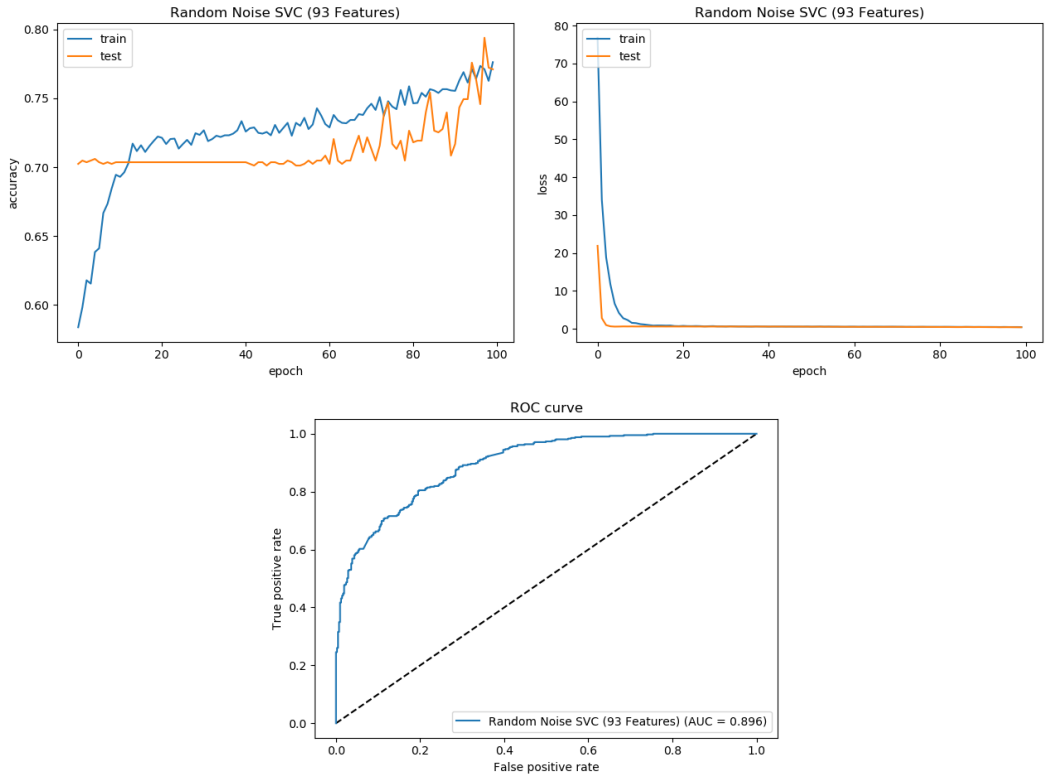

Figure 5. Accuracy, loss, and ROC curve relative to the model trained on the Noisy SVC dataset, using 93 features.

5.1.3 Switchboard (SWB)

The model trained with SWB data is the one that performed better, thus is the one implemented in the API. The final model test accuracy is 87%, with an AUC of 965.

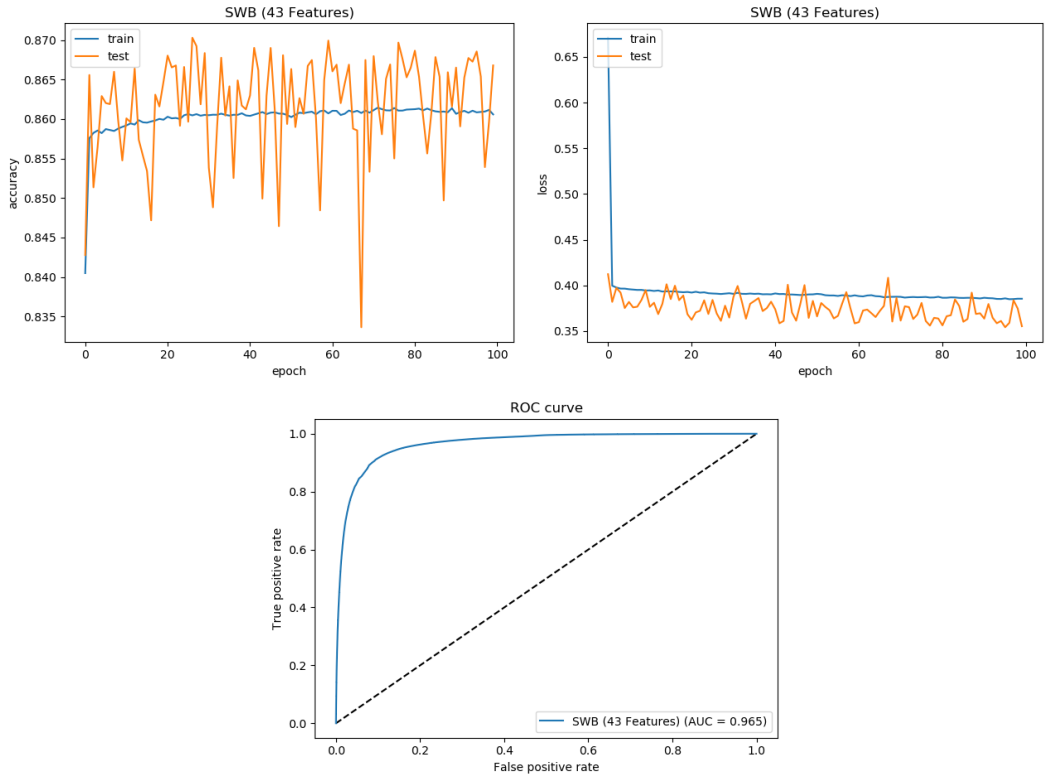

Figure 6. Accuracy, loss, and ROC curve relative to the model trained on the SWB dataset, using 43 features.

A more in-depth evaluation of this model is presented below.

5.2 Lold.js API Evaluation

Evaluation of the API's performance on a browser is done manually as follows: the demo API client is set up to play one clip at a time; the clip is played in its entirety while both models run and fire a prediction every 100ms; if these predictions are both above their respective threshold, a "multimodal laugh" episode is detected.

The test is performed on the real-life multimodal dataset created as a part of this project. Both modalities are tested in conjunction and in isolation. A general overview is provided below. More details for each modality can be found in the respective sub-chapter.

The results below were obtained with a threshold of 65% for both modalities.

Modality	Accuracy	Specificity (TNR)	Recall (TPR)	Precision	F1 score
Audio	61%	68%	54%	62.79%	58.06%
Video	64%	44%	84%	6%	70%
Multimodal	70%	92%	48%	85.71%	61.54%

5.2.1 Audio only

Following are the results of the audio modality in isolation.

Correctly classified 61/100 (61% accuracy), misclassified 39/100 (39% error rate).

	Predicted Laugh	**Predicted Other**
Actual Laugh	27 (TP)	23 (FN)
Actual Other	16 (FP)	34 (TN)

This model taken in isolation is not very accurate. Moreover, we can see how it is a little biased towards detecting “other” (57 predictions were “other”). The number of false negatives is also high, meaning the model could not catch 23 laughter episodes and wrongly labelled them as “other”.

There is much room for improvements. For instance, using transfer learning would have perhaps resulted in a better model. The time and hardware constraints of this project were too strict to be able to train a model from scratch and have it achieve robustness in real life settings.

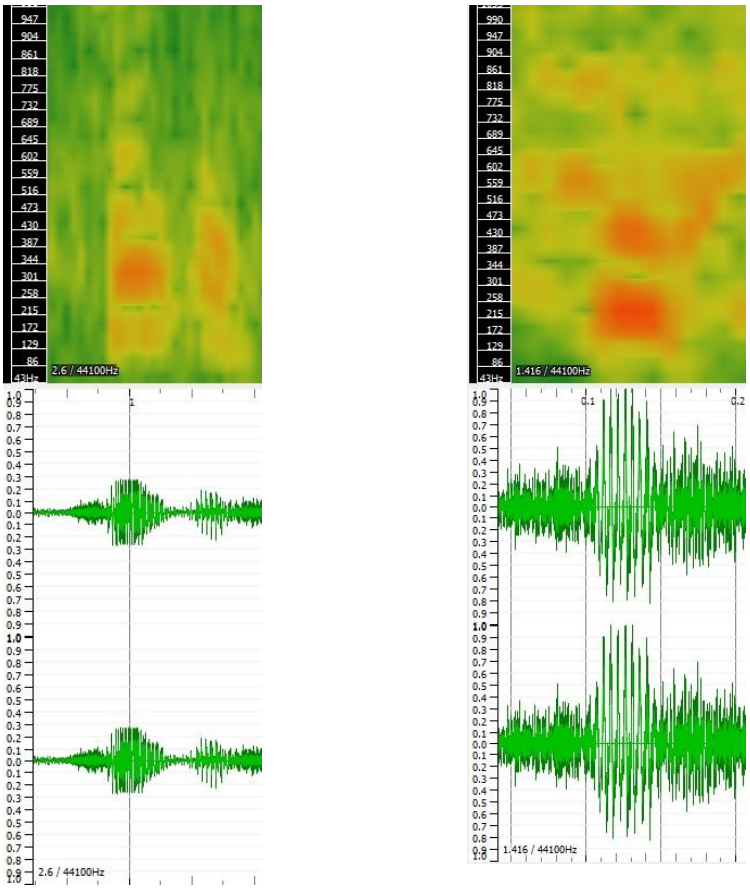

Figure 7. An example of a false positive detection for the audio model (left), and a true positive (right). The top part is the spectrogram of both channels mixed, the bottom the waveform of both channels separated

In the above example, false positive is spectrogram appears to have some similarities to an actual laughter episode, particularly the intensities in the lower frequencies.

For the above extract of a clip, the video model was not detecting laughter. Thus, the multimodal decision was correctly labelled as “other” – an improvement over a unimodal approach.

5.2.2 Video only

Following are the results of the video modality in isolation.

Correctly classified 64/100 (64% accuracy), misclassified 46/100 (46 % error rate)

	Predicted Laugh	**Predicted Other**
Actual Laugh	42 (TP)	8 (FN)
Actual Other	28 (FP)	22 (TN)

This model considered in isolation performed slightly better than the audio model alone (3% better accuracy). However, this model is heavily biased into detecting “laugh” over

"other", as 70% of its detection were "laugh". This might be because face-api.js's video model actually tries to detect "happiness", which can be triggered by a simple smile.

Figure 8. An example of a false positive detection for the video model

Figure 8 is another example of when the multimodality helped improved the accuracy of the model. In the frame shown, the video model detects laughter with 100% confidence. However, the audio model detects laughter with 0%. The minimum threshold for the multimodal prediction (65% confidence for both models) is not met, thus "other" is predicted.

5.2.3 Multimodal

Following are the results of both audio and video combined at the JavaScript level. As mentioned, the threshold for detecting "laugh" are set to 65% or more confidence for each modality – a "laugh" prediction is fired only when both models concurrently detect a laughter episode.

Correctly classified 70/100 (70% accuracy), misclassified 30/100 (30% error rate).

	Predicted Laugh	**Predicted Other**
Actual Laugh	24 (TP)	26 (FN)
Actual Other	4 (FP)	46 (TN)

The modality fusion improved the overall accuracy (from a lowest of 61% to 70%). Observing the confusion matrix, we can see how there are only 4 false positives. Depending on the use of the model, this might be desirable. By adjusting the thresholds mentioned at the beginning of this sub-chapter, the performance of the model could shift.

Figure 9. An example of a true positive detection for the modality fusion

In the example above, we can see that both models have a confidence of 65% or above, thus the multimodality prediction would be “laugh”.

The threshold of confidence was set to be equal for both models. However, these values can vary independently. After the evaluation, it was observed how the video model is biased towards detecting “laugh”, thus its threshold should be set higher. Likewise, the audio model was observed to be biased towards detecting “other”, hence its threshold should be lowered. Ideally, a meta-learner model should be trained to find the best combination of these parameters. However, due to time constraints this was not implemented.

5.3 Mobile performance

The performance of mobile hardware was tested in two ways: using the mobile hardware developer tools to monitor CPU and GPU; using Chrome developer tools on the remote target device.

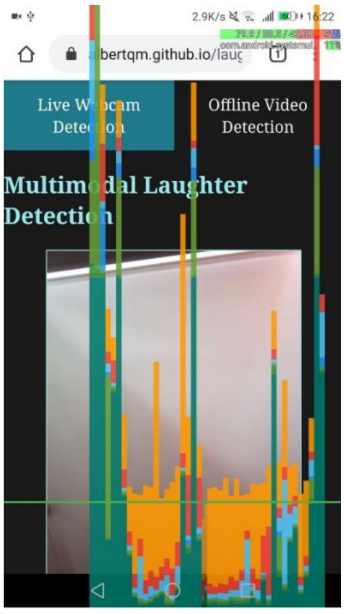

Figure 10. A screenshot of the CPU and GPU monitoring on the mobile hardware. Top right corner shows CPU usage (91%). The vertical bars on the screen are the GPU usage. The horizontal green line is a fixed line at 30% GPU usage.

The mobile performance was recorded on a P9 Lite (released in April 2016). It mounts two quad-core processors (4x2.0 GHz Cortex-A53, 4x1.7 GHz Cortex-A53) and a Mali-T830MP2 GPU.

The performance on mobile was observed to be not up to standards. The combination of Meyda.js extracting features and the TensorFlow.js backend was too demanding for the hardware. The GPU was constantly under stress and the CPU was always observed to be almost fully in use. This resulted in very low FPS, screen freezes for >5s seconds.

Another performance evaluation was completed using Chrome developer tools, which can access a mobile device connected via USB to a PC. Chrome's performance tool was used record and profile a period of 90 seconds.

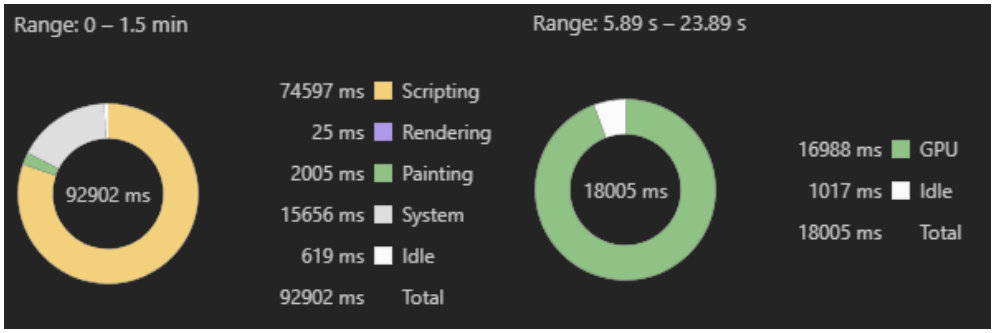

Figure 11. On the left, a general overview of the performance over a 90 second window. On the right, a focus on the GPU usage during a detection.

The performance profiling started on website load. As anticipated from the hardware tool test, only a tiny fraction of the time the phone was idling (fig 11). At the point of the first detection, when both Meyda.js and TensorFlow.js are called, the GPU was observed to be put under stress. The right side of figure 11 shows how a prediction used the GPU for 17 seconds straight before going back to normal - The phone froze for that entire period. In essence, the demo API client deployed cannot run smoothly on mobile devices.

Chapter 6: **Conclusion**

This project focused on multimodal laughter detection in real-life settings. There were multiple correlated aims. Firstly, build a machine learning model from scratch, which would detect laughter in life like scenarios. Secondly, assess how different approaches could improve the robustness of said model. Thirdly, provide a benchmark dataset to effectively measure improvements in a way that mirrors more closely real-life scenarios as opposed to lab conditions. Lastly, provide a developer friendly API that provides access to a lightweight multimodal laughter detection.

6.1 Real-life scenarios robustness

6.1.1 Modality fusion

Multimodality proved to be more accurate over a single modality laughter detection, confirming Poria's thesis (2017). The multimodality improved the accuracy despite the fusion happening at a high level, in the JavaScript layer of the client of the API.

However, the modality fusion has large room for improvement. It could be done at a lower level, by adding and extra layer in the RNN architecture, aiming at merging the two model's raw predictions in the best possible way.

6.1.2 Noisy dataset

A model trained on a dataset which was artificially distorted with noise was observed to perform worse on the evaluation set of the dataset itself, but better on the real-life dataset. However, note that the testing on the dataset itself was done using Keras, whereas the testing on the multimodal dataset was done manually with the process described at the beginning of the Evaluation chapter.

6.2 Multimodal real-life dataset

The dataset proved to be a good benchmark for multimodal laughter detection in in the wild situations, as opposed to the widely available and much larger datasets which are recorder under lab conditions.

The multimodal dataset created as part of this project is made publicly available. However, said dataset is currently too small to provide relevant data. A bigger dataset which comprises a diverse range of clips, like YouTube8M, should be used to further refute or confirm the thesis that multimodality improves the model's accuracy over one modality only.

6.3 Lold.js API

6.3.1 Mobile performance

Overall, the performance was not satisfactory. Lold.js went through various iterations which tweaked its performance, for instance using only one TensorFlow.js backend for both Meyda.js and face-api.js. None of the optimisations tried worked to an acceptable

extent: the mobile hardware tested was not able to run the API client created to demo the project. This would need more investigation, as it might be an API, client, or hardware problem.

A future iteration on Lold.js could include a collaboration with face-api.js, implementing laughter detection at the models' level rather than at the clients' level.

References

Akhtar, Z., Bedoya, S. and Falk, T.H., 2018, April. Improved Audio-Visual Laughter Detection Via Multi-Scale Multi-Resolution Image Texture Features and Classifier Fusion. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 3106-3110). IEEE.

Kennedy, L.S. and Ellis, D.P., 2004. Laughter detection in meetings.

Bedoya, S. and Falk, T.H., 2016, September. Laughter detection based on the fusion of local binary patterns, spectral and prosodic features. In 2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP) (pp. 1-5). IEEE.

Kantharaju, R.B., Ringeval, F. and Besacier, L., 2018, October. Automatic Recognition of Affective Laughter in Spontaneous Dyadic Interactions from Audiovisual Signals. In Proceedings of the 2018 on International Conference on Multimodal Interaction (pp. 220-228). ACM.

Hagerer, G., Pandit, V., Eyben, F. and Schuller, B., 2017, June. Enhancing lstm rnn-based speech overlap detection by artificially mixed data. In Audio Engineering Society Conference: 2017 AES International Conference on Semantic Audio. Audio Engineering Society.

Hagerer, G., Cummins, N., Eyben, F. and Schuller, B.W., 2017. " Did you laugh enough today?"-Deep Neural Networks for Mobile and Wearable Laughter Trackers. In INTERSPEECH (pp. 2044-2045).

Di Lascio, E., Gashi, S. and Santini, S., 2019, May. Laughter Recognition Using Non-invasive Wearable Devices. In Proceedings of the 13th EAI International Conference on Pervasive Computing Technologies for Healthcare (pp. 262-271). ACM.

Gosztolya, G., Beke, A., Neuberger, T. and Tóth, L., 2016. Laughter classification using Deep Rectifier Neural Networks with a minimal feature subset. Archives of Acoustics, 41(4), pp.669-682.

Gupta, R., Audhkhasi, K., Lee, S. and Narayanan, S., 2016. Detecting paralinguistic events in audio stream using context in features and probabilistic decisions. Computer speech & language, 36, pp.72-92.

Lingenfelser, F., Wagner, J., André, E., McKeown, G. and Curran, W., 2014, November. An event driven fusion approach for enjoyment recognition in real-time. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 377-386). ACM.

Szameitat, D.P., Alter, K., Szameitat, A.J., Darwin, C.J., Wildgruber, D., Dietrich, S. and Sterr, A., 2009. Differentiation of emotions in laughter at the behavioral level. Emotion, 9(3), p.397.

Zhalehpour, S., Onder, O., Akhtar, Z. and Erdem, C.E., 2016. BAUM-1: A spontaneous audio-visual face database of affective and mental states. IEEE Transactions on Affective Computing, 8(3), pp.300-313.

Roach, J. (2018). Microsoft AI and laughter converge at the National Comedy Center. [online] The AI Blog. Available at: https://blogs.microsoft.com/ai/ai-laugh-battle-national-comedy-center/ [Accessed 21 Nov. 2019].

Campbell, N., Kashioka, H. and Ohara, R., 2005. No laughing matter. In Ninth European Conference on Speech Communication and Technology.

Truong, K.P. and Van Leeuwen, D.A., 2007. Automatic discrimination between laughter and speech. Speech Communication, 49(2), pp.144-158.

Reuderink, B., Poel, M., Truong, K., Poppe, R. and Pantic, M., 2008, September. Decision-level fusion for audio-visual laughter detection. In International Workshop on Machine Learning for Multimodal Interaction (pp. 137-148). Springer, Berlin, Heidelberg.

Niewiadomski, R., Mancini, M., Varni, G., Volpe, G. and Camurri, A., 2015. Automated laughter detection from full-body movements. IEEE Transactions on Human-Machine Systems, 46(1), pp.113-123.

Petridis, S. and Pantic, M., 2010. Audiovisual discrimination between speech and laughter: Why and when visual information might help. IEEE Transactions on Multimedia, 13(2), pp.216-234.

Stöckli, S., Schulte-Mecklenbeck, M., Borer, S. and Samson, A.C., 2018. Facial expression analysis with AFFDEX and FACET: A validation study. Behavior research methods, 50(4), pp.1446-1460.

Cai, R., Lu, L., Zhang, H.J. and Cai, L.H., 2003, July. Highlight sound effects detection in audio stream. In 2003 International Conference on Multimedia and Expo. ICME'03. Proceedings (Cat. No. 03TH8698) (Vol. 3, pp. III-37). IEEE.

Knox, M.T. and Mirghafori, N., 2007. Automatic laughter detection using neural networks. In Eighth Annual Conference of the International Speech Communication Association.

Pantic, M. and Vinciarelli, A., 2009. Implicit human-centered tagging [Social Sciences]. IEEE Signal Processing Magazine, 26(6), pp.173-180.

Navarro, J., Del Moral, R., Alonso, M.F., Loste, P., Garcia-Campayo, J., Lahoz-Beltra, R. and Marijuán, P.C., 2014. Validation of laughter for diagnosis and evaluation of depression. Journal of affective disorders, 160, pp.43-49.

Navarro, J., Fernández Rosell, M., Castellanos, A., delMoral, R., Lahoz-Beltra, R. and Marijuan, P.C., 2019. Plausibility of a Neural Network Classifier-Based Neuroprosthesis for Depression Detection via Laughter Records. Frontiers in Neuroscience, 13, p.267.

McDuff, D., Mahmoud, A., Mavadati, M., Amr, M., Turcot, J. and Kaliouby, R.E., 2016, May. AFFDEX SDK: a cross-platform real-time multi-face expression recognition toolkit. In Proceedings of the 2016 CHI conference extended abstracts on human factors in computing systems (pp. 3723-3726). ACM.

Tatsumi, S., Mohammad, Y., Ohmoto, Y. and Nishida, T., 2014. Detection of hidden laughter for human-agent interaction. Procedia Computer Science, 35, pp.1053-1062.

Flutura, S., Wagner, J., Lingenfelser, F., Seiderer, A. and André, E., 2016, October. Laughter detection in the wild: demonstrating a tool for mobile social signal processing and visualization. In Proceedings of the 18th ACM International Conference on Multimodal Interaction (pp. 406-407). ACM.

Research2Guidance, 2017. Mhealth App Economics 2017: Current Status and Future Trends in Mobile Health.

Ali, E.E., Chew, L. and Yap, K.Y.L., 2016. Evolution and current status of mhealth research: a systematic review. BMJ Innovations, 2(1), pp.33-40.

Szameitat, D.P., Alter, K., Szameitat, A.J., Darwin, C.J., Wildgruber, D., Dietrich, S. and Sterr, A., 2009. Differentiation of emotions in laughter at the behavioral level. Emotion, 9(3), p.397.

Escalera, S., Puertas, E., Radeva, P. and Pujol, O., 2009, June. Multi-modal laughter recognition in video conversations. In 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (pp. 110-115). IEEE.

Lingenfelser, F., Wagner, J., André, E., McKeown, G. and Curran, W., 2014, November. An event driven fusion approach for enjoyment recognition in real-time. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 377-386). ACM.

Ito, A., Wang, X., Suzuki, M. and Makino, S., 2005, November. Smile and laughter recognition using speech processing and face recognition from conversation video. In 2005 International Conference on Cyberworlds (CW'05) (pp. 8-pp). IEEE.

Turker, B.B., Yemez, Y., Sezgin, T.M. and Erzin, E., 2017. Audio-facial laughter detection in naturalistic dyadic conversations. IEEE Transactions on Affective Computing, 8(4), pp.534-545.

Poria, S., Cambria, E., Bajpai, R. and Hussain, A., 2017. A review of affective computing: From unimodal analysis to multimodal fusion. Information Fusion, 37, pp.98-125.

Salamin, H., Polychroniou, A. and Vinciarelli, A., 2013, October. Automatic detection of laughter and fillers in spontaneous mobile phone conversations. In 2013 IEEE International Conference on Systems, Man, and Cybernetics (pp. 4282-4287). IEEE.

McKeown, G., Cowie, R., Curran, W., Ruch, W. and Douglas-Cowie, E., 2012. Ilhaire laughter database. In Proceedings of 4th International Workshop on Corpora for Research on Emotion, Sentiment & Social Signals, LREC (pp. 32-35).

Godfrey, J. and Holliman, E., 1993. Switchboard-1 Release 2 LDC97S62. DVD. Philadelphia: Linguistic Data Consortium.

Snyder, D., Chen, G. and Povey, D., 2015. Musan: A music, speech, and noise corpus. arXiv preprint arXiv:1510.08484.

Rousseau, A., Deléglise, P. and Esteve, Y., 2012, May. TED-LIUM: an Automatic Speech Recognition dedicated corpus. In LREC (pp. 125-129).

Gemmeke, J.F., Ellis, D.P., Freedman, D., Jansen, A., Lawrence, W., Moore, R.C., Plakal, M. and Ritter, M., 2017, March. Audio set: An ontology and human-labeled dataset for audio events. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 776-780). IEEE.

Bachu, R.G., Kopparthi, S., Adapa, B. and Barkana, B.D., 2008, June. Separation of voiced and unvoiced using zero crossing rate and energy of the speech signal. In American Society for Engineering Education (ASEE) Zone Conference Proceedings (pp. 1-7).

Ma, Y. and Nishihara, A., 2013. Efficient voice activity detection algorithm using long-term spectral flatness measure. EURASIP Journal on Audio, Speech, and Music Processing, 2013(1), p.87.