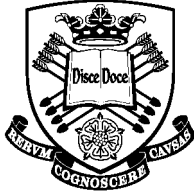


Disability and Dyslexia Support Service

Yellow Sticker

Department of Mechanical Engineering

I wish to indicate that I am a student with
dyslexia or other written communication
difficulties.



The
University
Of
Sheffield.

Department
Of
Mechanical
Engineering

MEng Mechanical Engineering

Lip Recognition Dataset Tool

Demetrios Loizides

May 2020

Dr Tim Dolmansley

Thesis submitted to the University of Sheffield in partial
fulfilment of the requirements for the degree of

Master of Engineering

SUMMARY

In the recent years we have seen large improvements in the field of Machine Learning (ML) and Artificial Intelligence (AI). Especially tasks such as face recognition and Automatic Speech Recognition (ASR)¹ have even been developed up to a commercial scale. Lip Recognition Technology (LRT) hasn't been developed as much as other ML technologies. However, ML learning architectures which implement lip-recognition have been developed, trained and tested on proprietary or limited usage datasets. The purpose of this report is to tackle this issue by creating an open source tool (program or *pipeline*²) which allows users to create datasets suitable for training ML lip-recognition algorithms. The tool was developed in python 3 and makes use of youtube-dl and FFmpeg command line tools. The tool consists of a series of functions which work sequentially to extract and process Audio-Visual (AV) content primarily from the largest available source online (YouTube). Various options have been developed allowing the customisation of the dataset. Simultaneously the range of available options and the tools functional design make it compatible with both online and local media sources. The tool was tested to work on Ubuntu 19.4 and python 3.7.7 in the anaconda environment and contains all the necessary features to develop a suitable dataset. Further work should focus on optimising the performance and including additional features such as alignment of speech to text.

¹ ASR word error rate (WER) of 5.1% on Microsoft's research report, 6.6% on IBMs and 6.7% on googles (110).

² A program used to extract and process digital media such as video and audio files for the purpose of training and testing ML algorithms. Programmes developed for such purposes have been named *pipelines* in various researches (9) (2).

NOMENCLATURE

AI	Artificial Intelligence
ALR	Automatic Lip-Recognition
ASR	Automatic Speech Recognition
AV	Audio-Visual
BOB	Box of Broadcasts
CNN	Convolutional Neural Networks
DNN	Deep Neural Network
FR	Face Recognition
FPS	Frames Per Second
HMM	Hidden Markov Models
ID	Identity
IoT	Internet of Things
LR	Lip Reading
LRS2	Lip Reading Sentences in the Wild (dataset)
LRS3	Lip Reading Sentences 3 (dataset)
LRT	Lip Recognition Technology
LRW	Lip Read in the Wild (dataset)
ML	Machine Learning
MT	Multiple Towers
MVP	Minimum Viable Product
OS	Operating Systems
P2FA	Penn Phonetics Lab Forced Aligner
SR	Speech Recognition
UI	Uniquely Identify

CONTENTS

Summary.....	iii
Nomenclature.....	iv
Contents.....	v
Acknowledgements.....	viii
1 Background.....	1
1.1 Lip Reading.....	1
1.2 Lip-Reading Datasets.....	2
2 Project Outline.....	2
2.1 Introduction to the Tool.....	2
2.2 Objective.....	3
2.3 Aims.....	3
3 Literature Review.....	4
3.1 Lip Reading.....	4
3.2 Past Studies of Lip-Recognition Technology (LRT).....	4
3.3 Lip-Reading Databases.....	6
3.3.1 GRID Corpus & Lombard Grid.....	6
3.3.2 Common Voice.....	7
3.3.3 MIRACL-VC1.....	7
3.3.4 VivaVoice.....	7
3.3.5 Lip Reading in the Wild (LRW).....	8
3.3.6 Lip Reading Sentence 2 (LRS2).....	8
3.3.7 Lip Reading Sentence 3 (LRS3).....	8
3.4 Pipelines.....	8
3.4.1 LRW (BBC) Pipeline.....	9
3.4.2 LRS2 (BBC) Pipeline.....	9
3.4.3 LRS3 (TED) Pipeline.....	10
3.5 Literature Review Summary.....	10
4 Copyright Exceptions.....	10
4.1 Downloading Content from YouTube.....	12
4.2 Fair Use Conclusions.....	12
5 Ethics.....	12
6 Project selection.....	13
6.1 Design Approach.....	13
6.2 Design Matrix.....	14
6.2.1 Results Breakdown.....	14
7 The Tool.....	15

7.1	Coding Principles	15
7.2	Introduction.....	16
7.2.1	Brief Design Overview.....	16
7.3	Dependencies	17
7.4	Main body Functions Breakdown.....	18
7.4.1	Essential Configurations.....	18
7.4.2	File Generation.....	18
7.4.3	List All Available Download formats.....	19
7.4.4	Downloading YouTube Content.....	21
7.4.5	Convert Audio File	21
7.4.6	Convert Video File	21
7.4.7	Download Subtitle Formats	22
7.4.8	Download Subtitles.....	22
7.4.9	Process Subtitles	25
7.4.10	Max Video Length.....	27
7.4.11	Replace Special Characters	27
7.4.12	Save Subtitles	28
7.4.13	Chopped Sample Folder.....	28
7.4.14	Slicing.....	29
7.4.15	Final Result Folders.....	31
7.4.16	Final Audio Files	31
7.4.17	Remove Audio	32
7.4.18	Face Recognition and Cropping	32
7.4.19	End Results	39
8	Testing.....	44
9	Future Improvements.....	47
10	License.....	48
11	Summary	49
12	APPENDIX 1.	50
12.1	MIT Licence.....	50
12.2	Future Software Architecture.....	51
12.3	Projects Gant chart	52
12.4	Link to Source Code	52
12.5	List of Tested URLs	52
12.6	Ethical Approval Form	53
12.7	Link to FOURCC Codecs	53
12.8	Demo File Notice	53
12.9	Variable 4 Layer 1 Key-Names Descriptions	53

13	References.....	54
----	-----------------	----

ACKNOWLEDGEMENTS

I would like to thank my parents for funding my university studies, my uncle from whom I have inherited my passion for science and my personal tutor for agreeing to supervise my self-initiated project.

1 BACKGROUND

1.1 Lip Reading

Lip Reading (LR), the task of recognising one's speech only from the movement of the lips and the tongue, it is a difficult task for both humans and machines. Lip Recognition Technology (LRT) has a broad range of potential applications and uses ranging from the identification of speech from people with neurological voice disorders to the improvement of Automatic Speech Recognition (ASR) in noisy environments.

ASR is known as the method of using computer hardware and software to identify human voice (1). Since 1994 ASR has improved greatly making lip-reading redundant in many cases. However, there are still many applications where lip-reading technology could be applied, such as:

- Automatic real time dictation (auto-generated subtitles) of silent films or off-mic conversations between politicians and celebrities (2).
- Intelligence agencies may make use of the technology for information theft (spying) or even as a method of biometric identification.
- Law enforcement, in the gathering of evidence through CCTV systems, smartphones, laptops and various other camera integrated systems.
- Complementary feature of speech recognition (e.g. letters 'm' and 'n' are easily confused while they are visually distinct (2)) (3) and assist in automatic dictation of movies. Companies which provide video calling as a service such as Facebook, Skype, WhatsApp etc. will also be interested in the technology. Ultimately lip-recognition may be implemented in the new age driverless cars to assist speech recognition to recognise the driver's commands while the radio plays in the background.
- Medical industry could use the technology to help patients with speaking disorders who have difficulties hearing and to improve their hearing aids (4). For instance, it could be used to assist the communication of mute people by integrating this technology to an Internet of Things (IoT) device.

Despite its many potential applications, LRTs have not improved as greatly as ASR or Face Recognition (FR). This could be due to two main reasons. Firstly, it is harder to achieve low error rates due to the many influential factors such as spoken speed, image resolution, low frame rate, accents, variable light conditions (2) etc. Secondly, LRT is less researched as there is less demand in comparison to SR. In addition there is a lack of comprehensive and large enough freely available datasets.

1.2 Lip-Reading Datasets

Most research studies in the past have only been able to recognise single characters mainly with the use of Hidden Markov Models (HMM) (2). This could have been due to the lack of both computing power and large vocabulary datasets since the creation of small datasets of digits 0-9 or with alphabetical characters a-z is a much easier task. It is only in the recent years that we have seen a significant improvement on LRT where Deep Neural Network (DNN) models have been implemented to recognise whole words and sentences (2) (5). DNN architectures such as Convolutional Neural Networks (CNN) require large training datasets in order to avoid *overfitting*³ and to become generalised (6). Unfortunately the only large enough and word rich datasets suitable for lip-reading are Lip Read in the Wild (LRW) from BBC, Lip Reading Sentences in the Wild (LRS2) again from BBC and Lip Reading Sentences 3 (LRS3) from TED and TEDx (7). These datasets are available to be used for non-commercial, academic research purposes only. In order for one to obtain any of these datasets, a data sharing agreement with BBC research and development department must be signed. The agreement must be then sent via e-mail to the BBC (rob.cooper@bbc.co.uk) for confirmation (8). The datasets are not easily accessible; the procedure of obtaining the datasets requires manual evaluation of the submitted application which could be a fairly time consuming process. Since the datasets are made with proprietary content from BBC programs and TED conferences, requests originating from non-UK based and non-academic individuals are likely to be rejected as there is no control over the datasets usage once shared.

2 PROJECT OUTLINE

2.1 Introduction to the Tool

Any individual who would like to research or commercialise LRT, apart from building their own ML models or use a premade architecture will need to use a dataset to train and test their model. The purpose of the work conducted was to create a *pipeline* program named *Tool* which would allow individuals to easily create

³ *Overfitting* is when a model becomes over trained onto a dataset that starts capturing (using) the noise within the data in its prediction. Since the noise is produced randomly this results to a reduction in performance (accuracy).

these datasets. The *Tool* should be capable of producing datasets suitable for DNN to be trained and tested on.

The *Tool* will be mostly useful to researchers who wish to study lip-recognition or any individual who is interested in the field and would like to create their own custom dataset. The *Tool* will be licenced under an open source licence which will grant the use, modification, copy, sale and redistribution of the *Tool* to anyone for free. The *Tool* will not be developed for the purpose of making a profit. Hence anyone, regardless of their economical background may make use of the *Tool* for personal or commercial purposes.

Three potential approaches for creating the dataset have been considered. The first approach makes use of freely available sources on the internet such as YouTube to extract the content (audio, video, subtitles etc.). The second approach gathers content by recording AV content locally from volunteers using the computers pre-build camera or any other device. The third and final approach gathers content through an online application where volunteers can record and submit content. For the second and third approach the transcript must be prepared in advance and be presented to the volunteer to read out loud in front of a camera.

All three methods result in a video file of someone's face speaking and a transcript file. Since all methods share the same end result, the procedures for transforming the content into the suitable format where ML algorithms can be trained are identical.

2.2 Objective

The objective of the project was to create an open source program (*Tool*) which will allow anyone to create their own large and comprehensive datasets suitable for training lip-reading DNN models.

2.3 Aims

The aims of the project were the following:

- Ensure the *Tool* is developed sufficiently to the point that a lip-reading dataset can be built.
- Make sure the *Tool* is capable of collecting AV content either from an internet source (YouTube), locally or through online volunteers.
- Ensure modularity and adaptability. Hence, the *Tool* should be easily adapted to support any of the three methods of collecting AV content.

- Ensure millisecond video splitting accuracy and there for the ability to produce video segments each containing a single word or whole sentence instances⁴.
- Ability to separate the audio from the video and hence create pure audio and pure video datasets which could be used for ASR and Automatic Lip-Recognition (ALR).
- Include some intelligence in the *Tool*, such as face recognition, in order to assist in the identification of the useful content (a talking head) in the video.
- Extract useful features such as the change in position of facial land mark positions.

3 LITERATURE REVIEW

3.1 Lip Reading

Lip reading is a very difficult task for both humans and machines (9). According to a research study conducted in 1982 hearing impaired people have shown a 17% accuracy for a short range of only 30 monosyllabic words (9). Unfortunately, due to homophemes⁵ letters such as 'p', 'b' and 'm' which are visually identical cause LRT to become incapable of distinguishing some words. At the same time SR models easily confuse letters such as 'm' and 'n' which are visually distinct (2). There are ways of overcoming these difficulties such as combining lip-reading with speech recognition or use some form of context based prediction technique such as the trigram analysis⁶ (9).

3.2 Past Studies of Lip-Recognition Technology (LRT)

Most work done in the past does not employ DNN methods in the attempt to solve lip-reading (2) (9). In 1997 Goldschen was one of the first scientists who attempted to perform lip-recognition without using a combination of both audio and

⁴ Word/sentence instance is a file sliced out of a whole video at specific times to contain only a single word or a sentence.

⁵ Word which produce different sounds but involve identical lip movements.

⁶ Trigram analysis is a method of establishing context through the use of three-word clusters [66]. With the use of probabilities, the third word in sequence is predicted based on the first two words [66].

visual features. Recognition was accomplished with the use of hidden Markov models (HMMs) on hand-segmented phones instead of lip-movements (9).

Integration of automatic lip-reading with acoustic speech to improve overall SR has also been attempted in the past. In 1994 Paul Duchnowski performed integration of visual information with acoustic speech using a multi-state time delay neural network (10). The work done was only capable of recognising spelled characters using the German alphabet (10). In other words, to recognise a word such as 'hello', each letter had to be spoken separately in sequence such as 'h' 'e' 'l' 'l' 'o' (10).

In 2000 Neti et al. used IBMs ViaVoice database to perform sentence speech recognition using HMM on extracted audio-visual (AV) features (9).

In 2016 Researchers at the University of Oxford created LipNet (9), a lip-reading software which implements deep learning end-to-end (E2E)⁷ trainable models (11). The software, instead of classifying (identifying) discrete words, attempts to recognise whole sentences (continuous speech). Due to continuous speech, coarticulation⁸ occurs causing distortion at the words' boundaries. This particular approach was inspired from the fact that human lip readers are more successful in identifying longer words rather than shorter words (9). LipNet was trained and tested on the GRID corpus dataset (9). With use of the iBug landmark predictor 68 facial landmarks were predicted and the region of the lips was cropped (9). The cropped region was used for the training and prediction of words (9). The program achieved 95.2% accuracy in contrast to 52.3% for experienced human lip readers and the previous 86.4% state-of-the-art accuracy (12). It is important to mention that high accuracy results such as these were due to the fact that the GRID corpus dataset consists of a very limited range of words making the lip-reading task much easier (13). The algorithm is incapable of recognising a sufficient range of words to be used commercially. However, it was at the time a solid proof concept that DNN could be used to tackle lip-reading which led to further research. The source code of LipNets is freely available on GitHub.

Another research paper on LRT was published by the University of Oxford in 2016 (2). Similar to LipNet, various studies use CNN architectures to recognise whole

⁷ E2E training refers to the method of training an algorithm directly from the sampled data.

⁸ Coarticulation is the phenomenon which causes two words to become partly merged due to the process of continuous speech. For example the end position of the lips when a word is spoken becomes the starting position of the next word and vice versa.

words from continuous speech (2). In order to account for noise due to the relative motion between the head and the camera, rather than implementing visual registration⁹ the model was trained to recognise some degree of tolerance (2). A program known as *pipeline* was developed which automatically collected and produced hundreds of different single word instances from BBC with over a million instances in total (2). Data augmentation¹⁰ was implemented in order to expand the dataset and improve performance by reducing *overfitting* (2). The *pipeline* was used to create the LRW dataset (2). Despite the large size of the dataset, the models were trained and tested only on 500 and 333 different words for the most commonly word instances in the dataset (2). Each word in the training and testing datasets is contained more than 800 times (2). Out of four architectures tested, Multiple Towers (MT) achieved the highest accuracy rates of 65.4% for a set of 333 words and 61.1% for a 500 set.

The research of LRT at the University of Oxford continued in the subsequent year of 2017 (14). The study published two contributions, a new dataset Lip Reading Sentence (LRS2) and an algorithm WLAS which transcribes videos of mouth movements into characters (14). LRS2 was created from BBC video samples and contains 100,000 of sentences (14). The WLAS model makes use of DNN and is capable of operating over pure audio, pure video or both inputs to perform dictation (14). WLAS is a sequence-to-sequence model, reads whole sentences as inputs and attempts to predict whole sentences (14). The study exceeded in performance all previous studies when the WLAS model was tested on LRS2, LRW and GRID datasets (14). The study also showed that lip-reading could be used to improve SR (14).

3.3 Lip-Reading Databases

3.3.1 GRID Corpus & Lombard Grid

The GRID Corpus consists of 34 speakers, 16 female and 18 male speakers (15). The dataset contains 1000 sentences per speaker and a total of 27.5 hours (9). Samples are 3 seconds long and contain 25 fps. The dataset contains audio at 44.1 kHz. FFmpeg was used to convert the video files to MPEG-1 format (15). Each sentence is structured with a fixed pre-determined sequence of classified words:

⁹ Visual or image registration is the process of transforming the coordinate systems of different frames within a sample into a single common coordinate system (113)

¹⁰ Data augmentation is a common technique used for enlarging ML datasets. The technique enlarges the dataset by creating duplicates of the samples with small modifications such as adding noise to the samples.

command, colour, preposition, letter, digit, adverb (13). The dataset contains a vocabulary of only 51 words and it is deemed unsuitable for large scale lip-recognition (16) (15). LipNet was trained and tested on this dataset by using samples containing the region of the lips cropped at 100×50 pixels per frame (9).

The Lombard Grid is considered an extension of the GRID dataset as the samples follow the same sentence format. The dataset consists of 54 speakers, 30 female and 24 male (17). The dataset contains 100 utterances per speaker, 50 front facing and 50 side facing and in total 16,200 samples of audio, front view and side view video samples (17). The audio files of the dataset are of 'wav' format and the video files of 'mov' format (17).

Both datasets have been produced by the University of Sheffield and are freely available for personal and commercial use under the CC BY 4.0 licence.

3.3.2 Common Voice

Common Voice is Mozilla's open source multi-language database of voices that anyone can use to train SR algorithms (18). The database was created through an online application where volunteers contribute samples of their voice. The idea of introducing a platform open to the public for everyone to contribute is brilliant and apparently very successful. The database has reached a size of 38 GB with 3401 recorded hours in total across 40 different languages (19). The database is specifically for the training of speech recognition algorithms only, as it consists purely of audio content. It is one of the largest and most diverse datasets available to the general public licensed under the creative commons-0 (CC-0, no copyright) which allows any use of the dataset (18).

3.3.3 MIRACL-VC1

MIRACL-VC1 is a lip-reading dataset which consists of five men and ten women repeating a pre-defined list of ten words and ten phrases (20). The dataset contains video samples of size 640×480 pixels and a total number of 3000 instances. It is available for research purposes only (20).

3.3.4 VivaVoice

VivaVoice is a proprietary lip-reading dataset developed by IBM. The dataset consists of 290 subjects performing continuous speech (21). The database consists of 10,500 words and a total of 24,325 utterances. The dataset contains video samples of size 704×480 pixels and frame rate of 30fps. Unfortunately, the dataset is not available to the public.

3.3.5 Lip Reading in the Wild (LRW)

The LRW dataset was introduced by the University of Oxford in 2016 to overcome the shortage of large vocabulary datasets (2). The dataset consists of proprietary data from the BBC (2). The dataset contains 1000s of hours of spoken text with 1000 different words with over 1million word instances for more than 1000 different speakers (2). The overall length of the dataset is 173 hours (22). Each sample in the dataset contains a single word instances sampled at 25fps (2). The dataset is available for pure research purposes under the CC-BY-NC-ND 4.0 licence (2).

3.3.6 Lip Reading Sentence 2 (LRS2)

LRS2 was published the year after LRW (2017) by the University of Oxford. LRS2 was produced from BBC programs that have been on live television between 2010-2016 (14). The dataset, in contrast to LRW, contains whole sentences rather than single word instances (14). The dataset, contains mostly news programs, as they contain stable and continuous talking heads (14). The dataset consists of more than 100,000 natural sentences, a vocabulary of 120,693 words and a total of 224 hours of length (14) (22). The video instances are samples at 25 fps (14). WLAS was trained using LRS2 with cropped input images at the region of the mouth of size 120 × 120 pixels (14). The dataset is available for pure research purposes under the CC-BY-NC-ND 4.0 licence (14).

3.3.7 Lip Reading Sentence 3 (LRS3)

LRS3 was introduced in 2018 by the University of Oxford (22). The dataset consists of TED and TEDx (YouTube) video samples of dimensions 224×224 pixels only and a frame rate of 25fps (22). The audio samples are at 16 kHz (22). The content was collected from TED and the TEDx YouTube channel (22). The samples contain whole sentences from 14,352 speakers and 128,136 vocabulary words (22). The dataset consists of a total length of 438 hours (22). Unlike LRW and LRS2, the LRS3 is very unlikely to contain the same person speaking in both training and test sets because each TED talk involves a different speaker as opposed to news programs (22). The dataset is available for pure research purposes under the CC-BY-NC-ND 4.0 licence (22).

3.4 Pipelines

Pipelines, are programs which have been developed to extract and manipulate data from datasets, in the necessary format, needed for training ML models. The program developed in this project is essentially a *pipeline* program.

3.4.1 LRW (BBC) Pipeline

The *pipeline* performs the following operations to converted content from BBC into the LRW dataset:

1. Aligned the text to the audio with the use of the Penn Phonetics Lab Forced Aligner (P2FA) (2).
2. Apply the HOG-based face detection method to identify the frames which contain a face, hence the frame to apply tracking (2).
3. Determine the speaking head by:
 - a. Apply facial landmarks on all faces detected,
 - b. Extract the *temporal signal*¹¹ from the relative motion between the upper and lower lip landmarks (2).
 - c. Implement a linear SVM classifier to make the distinction (2).
4. Extracted the subtitles by performing Optical Character Recognition on printed subtitled in the video (2).
5. Crop the region of the lips (2).
6. Apply data augmentation such as random cropping, flipping etc. (2).
7. Create test and validation datasets of 500 and 333 vocabulary sizes containing the most common spoken words in the dataset (2).

3.4.2 LRS2 (BBC) Pipeline

The *pipeline* used for LRS2 contains many similar processes as the *pipeline* of LRW (23). The processes performed are:

1. Alignment of text to audio, identical to step 1 of LRW (BBC) *pipeline* (14).
2. Apply face detection, identical to step 2 of LRW (BBC) *pipeline* (14).
3. With the use of regression trees facial landmarks were extracted (14).
4. Audio and video streams were aligned based on the CNN SyncNet developed by Joon Son Chung and Andrew Zisserman (24) (14). The same network was used to determine playback music and reject voice over samples (14).
5. Separate the text in sentences using punctuations in the transcript (14).
6. Crop the region of the lips (14).

¹¹ *Temporal signal* is the time signal produced from a tracked point across multiple video frames.

3.4.3 LRS3 (TED) Pipeline

The *pipeline* used to create LRS3 contains many similarities as the *pipelines* of LRW and LRS2. The *pipeline* performs the following processes:

1. Applies a CNN face detector based on the Single Shot Multibox Detector (SSD) for every frame (25) (23). The SSD face detector is considered faster and it is also capable of detecting faces from all angles as opposed to the HOG-based detector used in LRW and LRS2 (23) (2).
2. Alignment of text to audio, identical to step 1 of LRW (BBC) *pipeline* (14).
3. Alignment was evaluated by running Kaldi-based ASR model on the samples (23).
4. Alignment of audio to video, identical to step 4 on LRS2 (BBC) *pipeline* (23).
5. Crop the region of the lips (23).

3.5 Literature Review Summary

Recent research suggests DNN is the most effective approach to tackle lip-reading. Longer samples make lip-recognition easier for both humans and machines, hence the processing of sentence samples is preferred over individual words. The LRW, LRS2 and LRS3 datasets are the most comprehensive and vocabulary sufficient datasets which could be used to train DNN. These datasets contain samples at 25 fps and a vocabulary of more than 100,000 words. The models which deploy DNN are trained and tested through word or sentence samples of cropped regions of about 100×100 pixels at the region of the mouth. All *pipelines* share very similar functionalities such as alignment between text, audio and video, face detection, speaker detection, landmark extraction and cropping at the region of the lips etc.

4 COPYRIGHT EXCEPTIONS

Since the English language was established as the international language of the world, most tools and trained models available, are in American English. The *Tool* will primarily make use of samples from fluent English speakers. As a result, we are mostly concerned about the copyright laws of the US and UK where most videos used will originate.

In general, in order to make use of a copyrighted material such as a YouTube video it is required to obtain permission from the copyright owner. Fortunately, there is an exception to copyright law known as 'fair use' in the United States (US) and 'fair dealing' in the UK. Fair use allows the reuse of copyright protected material

(including YouTube videos) under certain circumstances without the permission of the copyright owner (26).

In the US only the court is allowed to determine what can be considered as fair use and the regulations vary between countries (26). This is fairly problematic as there is no clear distinction of what can be considered as fair use and what cannot; hence, some risk is introduced regardless of the precaution measurements taken. The judge's decision of whether or not a copy write complies for fair use is based on the following attributes:

1. The purpose and character of the use (27). If the work was modified such as a new expression or meaning is added then the content can be considered transformative and therefore will fall under the terms of fair use (27) (26). This usually applies for videos which comment or refer to other videos and it is not applicable in our case.
2. The nature of the copyrighted work (27). The use of factual works such as a documentary which may be used for educational purposes is more likely to be accounted as fair use rather than fictional works that can be used for entertainment purposes (27). Usually the reporting of current events that may benefit the general public fall under this category (27).
3. The amount and substantiality of the portion taken (27). The less the copyrighted material used the more likely it will be considered as fair use.
4. The effect of the use upon the potential market (27). Whether the use of the copyrighted material may impact the income of the copyright owner to an existing or potential market (27).

According to the US law, a dataset consisting of copyrighted content may be considered as fair use for the following reasons: (a) the copyrighted content which will be contained in the dataset will be modified (cropped etc.) to produce new meaning, and (b) the dataset will not be commercialised (sold for money) and will be used for research purposes (the training of machine learning algorithms).

Similarly, the UK includes an exemption to its copyright law known as fair use. As opposed to the US, in the UK defines clear boundaries of which actions are considered as fair use. As stated on the government website 'allows researchers to make copies of any copyright material for the purpose of computational analysis if they already have the right to read the work' (28). Hence, copies of copyrighted material are permitted for the purpose of pure research for text and data mining purposes if the individual is allowed to read the content. Therefore any video available to read on the internet (including from YouTube) can be downloaded and processed to test the functionality of the *Tool*. In addition, a lip-reading dataset

could be created using this content where ML models are trained assuming neither the dataset nor the model is sold or used commercially. However, the code (*Tool*) written to modify the YouTube videos and thus create the dataset may be licensed under any license.

4.1 Downloading Content from YouTube

YouTube is unquestionably the largest and most popular AV dataset available on the web. Most YouTube videos are copyright protected and YouTube cannot grant the rights to download and make use of the copyrighted content posted on YouTube (29). Unfortunately, YouTube cannot help identify the video owner. This is can be problematic as requesting explicit permission for each video would slow down greatly the process of building a sufficiently large dataset to train a lip recognition algorithm.

4.2 Fair Use Conclusions

Both the US and UK laws share many similarities. The UK states clearly that copyrighted material may be used for pure research purposes only. The US suggests the same but with the distinction that this can only be determined by the court. YouTube does not provide help in identifying the owners of a video apart from the video content available on their YouTube channel. Hence, identifying the origin of the owner requires research. Since international copyright law does not exists, it is unknown of whether or not the country of origin, the video was created in, determines which law is applicable or from the country the video was downloaded and used. For example if a video was produced in the US and it is downloaded and processed in the UK, which of the two laws applies?

5 ETHICS

If the design approach involves the creation of a dataset from local or online recordings, volunteers will be asked to submit samples of themselves speaking in front of a camera. Participants are required to read and consent to the terms mentioned in the ethical approval form (Appendix 12.7). The contributors apart from submitting samples of themselves talking will also be asked to provide useful information such as gender, age, ethnic origin and mother tongue. The protection of the privacy of the participants and the prevention of storing sufficient information to Uniquely Identify (UI) a person are taken into serious consideration. The approach of storing personal information and protecting it with the use of

encryption was discarded as it is prone to human errors. The following steps ensure the protection of the participant's privacy by avoiding the collection of sufficient information to track or to UI them:

- Make use of randomly generated identities (IDs) in the dataset instead of names.
- Crop and keep only the region of the mouth stored in the dataset. Full face images could be used to UI someone using image search engines (30) or even unlock smart devices with enabled facial biometric identification (31).
- Extract landmark positions only at the region of the lips. Facial landmark points across the whole face could be used to identify someone.

If the design approach makes use from content which is already available to read from the internet, then there is no need to limit (partly remove) the content stored locally. However, since the content will already be freely available to read, the videos' URLs will be stored. If this design approach is followed, the content could be used without the owner's consent as the purpose of the work conducted will fall under the UK fair usage exception.

6 PROJECT SELECTION

Three potential approaches have been considered in the creation of the dataset. The first approach makes use of content freely available from the internet, the second of local recordings and the third from online recording through a web application.

The software architecture of how a commercialised version of the *Tool* which incorporates all three design approaches operates is show on Figure 43 in the Appendix 12.2.

6.1 Design Approach

The selection of the design approach was guided by Paul Nanouk's ideology. In his own words this is '*doing the most good for the most people with the least amount of effort*' (33).

During the development of the *Tool* the principles of agile software development (Figure 1) will be applied along with the UNIX OS coding

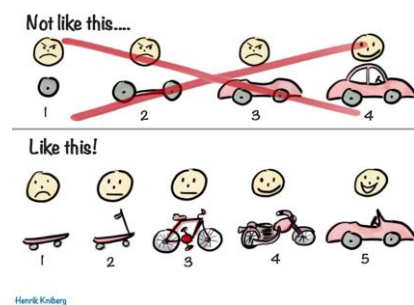


Figure 1 Minimum viable product, agile development by Henrik (32).

principles. The approach of developing the Minimum Viable Product (MVP) first (Figure 1) was applied on the development of this project. Rather than attempting to produce a perfect application, the most essential processes, such as video extraction, slicing, face recognition and lip-cropping were developed first.

6.2 Design Matrix

In order to decide the best design approach, a simple decision matrix was filled in with values ranging from 1 – 5. Number 5 was used for excellent and 1 for poor performance. The following properties were considered:

Quality: The quality parameter refers to the quality of the end result dataset. The value is based on the ability of the method to produce a dataset with identical or better specifications in comparison to other lip-reading datasets such as LRW.

Difficulty: The difficulty is based on the complexity and expertise needed to create the *Tool*. It is characterised as the ‘*amount of effort*’ from Paul Nanouk’s quote.

Time: Time which will be required to build the dataset. The less the time needed to create the dataset the more realistic the end goal of the project.

Table 1 Decision matrix table.

	Approaches	Quality	Time	Difficulty	Total
1	Internet	3*	4	3	36
2	Local	2*	1	4	8
3	Online	2*	3*	1	6

Approaches on Table 1 have been ranked with comparison to one another and from the available possessions such as the build in camera of the laptop Dell Inspiron 7559. The asterisk ‘*’ indicates there is an uncertainty or variation due to the lack of information.

6.2.1 Results Breakdown

The values assigned to quality (Table 1) originate from the ability of each approach to produce a dataset containing high resolution and high frame rate (25>) samples. The built-in camera of the available computer (Dell Inspiron 7559) specifies 30 fps (maximum) and a resolution of 720p (34). When tested in practice using the cv2 library in python 3, the maximum fps achieved was only 15 fps. When face recognition was applied performance dropped even further to near 10 fps. The frame rate of the samples in the dataset is very important for LRT (35). The average

time taken to speak a common word such as 'hello' is about 300ms¹². For 10 fps this results to only 3 data points which is insufficient. According to the literature, even 25 fps contained in other datasets such as LRW is too low when speaking at high speed rate (35). Due to these reasons both local and online approaches have been ranked with the same value as the specifications between personal computers are similar. The asterisk assigned accounts for the variation in specifications and the potential of using an external high fps camera. The internet approach was ranked higher as videos on YouTube range from 144p to 4k resolutions and framerates of 25, 30 and up to 60fps. With the use of the command line tool youtube-dl all available formats for a specific URLs can be obtained.

The Time (Table 1) corresponds to the amount of time needed to build the dataset. The local approach requires recording of the samples from scratch. This is a tedious and time consuming process and hence it is ranked the lowest out of the three. The online approach relies from online volunteers to contribute recorded samples which will then be processed in a suitable format. The rate at which raw samples are submitted depends on the popularity of the platform; nevertheless the samples must also be verified. The internet approach was ranked higher than the rest as it makes use of pre-recorded samples available on the internet and does not involve the additional step of producing the samples.

In the difficulty parameter (Table 1) the local approach was ranked the highest as it is the simplest to implement and the online approach the lowest as it is the most complex to implement (requires setting an online application to a server etc.).

According to the design evaluation matrix, the internet approach surpassed significantly the other approaches and will thus be the approach to be developed.

7 THE TOOL

7.1 Coding Principles

When large programs are written, specific and well defined guidelines must be followed to preserve clarity and manage complexity. This is essential to ease fault identification and debugging. The UNIX programming style/strategy was adopted during the development of the *Tool* program. UNIX based Operating Systems (OS)

¹² By measuring the time taken for one to say the word 'hello' 10 times and dividing by 10 we find that it takes an average of 300ms to speak out loud the word.

such as Linux, apart from been free are unquestionably one of the most versatile, efficient and long-lasting OS. They are primarily used in critical restless systems such as servers and supercomputers [57].

According to the book 'The Art of Unix Programming', in order to write a long lasting program the following coding practices should be followed:

- 'Write programs to do one thing and do it well' (36) (37) (38).
- 'Write programs to work together' (36).
- 'Expect the output of every program to become the input to another' (37).
- 'Build it out of simple parts connected by well-defined interfaces, upgrading a part without breaking the whole.' (39).
- 'Keep it simple to make it faster' (38).

The following statements emphasize the concepts of modularity, reusability and simplicity which should be considered in the design.

7.2 Introduction

The *Tool* is essentially a *pipeline* program which instead of using a proprietary dataset such as BBC's news programs, it extracts content from YouTube. Content is downloaded with the use of youtube-dl an open source command-line program developed specifically for the purpose of downloading content from YouTube. Despite the fact that youtube-dl is optimised for YouTube it is capable of extracting AV content for more than a thousand websites (40). This design feature allows it to automate many operations similar to other *pipelines* to create datasets similar to LRW, LRS2 and LRS3. The program handles many different file formats as it makes use of FFmpeg which supports a very wide range of available formats including mp4, mp3, wav, avi, mkv, y4m and more. FFmpeg was used in the creation of LRS3 and is a very powerful open-source command-line program. It is capable of decoding, encoding, transcoding, filtering and playing most if not all media formats. The *Tool* does not support interactive mode; all desired changes to the default parameters must be specified prior the execution of the program.

7.2.1 Brief Design Overview

The program was broken down into multiple files each containing multiple functions. The main body of the program calls functions defined in these files in the correct sequence to produce the desired result. The output of each function becomes the input of the next function. Once the desired settings are specified, the program continues execution without the need of any further user interpretation. The initial functions (Function 4) are responsible of downloading content from

YouTube. This function could be replaced with other functions suitable for downloading content from other sources such as Box of Broadcasts (BOB) without requiring changes to the remaining functions. Similarly, the functions responsible for manipulating AV content could be applied on local data.

7.3 Dependencies

The software was developed in python 3 and it was designed and tested to work on Kubuntu 19.04¹³. The software has the following dependencies:

- Python 3.7.7 with the anaconda environment. In particular the following modules are used: pandas, numpy, cv2, dlib, time, string, itertools, subprocess, pickle, csv, json, os, argparse, pydub, audiosegment, random and date.
- Ubuntu based distribution of 19.04 version or greater.
- The bash¹⁴ command line program (default terminal).
- FFmpeg; an open source multimedia handling command line tool.
- Youtube-dl; an open source media extraction command line tool.

The overall program consists of ten files, nine of which are necessary for the execution of the program and one for demonstration and testing purposes.

Table 2 Software file content breakdown.

File Name	Brief Description
main.py	Main body of the program which is executed by the user.
module_convert_audio_to_wav.py	Contain the functions called by the main.py. In order to reduce complexity and make debugging easier, functions responsible for complicated tasks make use of other smaller and simpler functions. Each function and its sub-functions are stored in the same file, hence all functions are self-contained.
module_face_detection.py	
module_process_subtitles.py	
module_sample_name.py	
module_save_variables.py	
module_video_processing.py	
module_youtube_extract.py	

¹³ Name of OS based on Ubuntu Linux distribution.

¹⁴ Bash also known as terminal is a similar program to the windows command-prompt.

DEMO_face_detect.py	Demo file showing the user the effect of various different face recognition options available.
shape_predictor_68_face_landmarks.dat	Necessary file for loading the facial land marks.

7.4 Main body Functions Breakdown

The program is executed from main.py. A link to the GitHub repository, where the whole source code of program is published, can be found in the Appendix 12.4. The functions are listed in their execution order and any identical function arguments (parameters) across multiple functions share the same purposes. For example the argument *SAVE* is used to specify if the output result of the function should be stored locally for all functions.

7.4.1 Essential Configurations

In order for the program to execute apart from satisfying all dependencies and loading all necessary module files, which contain the functions, the user needs the following: 1) stable internet connection, 2) URL to specific YouTube video which should be downloaded, 3) specify the name of the video file which will be downloaded.

7.4.2 File Generation

Function 1 Random string generator.

```
random_string_INPUT = str(module_sample_name.passw_gen(MODE=0, LENGTH=3))
```

Function 1 generates a random string variable (`random_string_INPUT`) which will become the input of Function 2. The *MODE* argument accepts integer values 0-6 and specifies the range of possible characters contained in the string. The available options are alphabetical characters a-Z, numbers 0-9, special characters and all possible combinations of them. The default value of 0 corresponds to pure alphabetical characters. The *LENGTH* argument accepts positive integer values and specifies the number of characters the string consists of. Figure 2 shows an example result produced by Function 1.



random_string_INPUT	str	1	JzG
sentence_chunk_samples_info_acpse	NoneType	1	None

Figure 2 Function 1 output, variable `random_string_INPUT`.

Function 2 Folder generator.

```
FOLDER_PATH = module_sample_name.folder_gen(RANDOM_STRING = random_string_  
INPUT, FILE_PERMISSION = '777')
```

Function 2 generates the folder in which all the downloaded content (subtitles, video and audio) will be stored and creates a string which contains the absolute (whole) path to that folder (Figure 3). The argument *RANDOM_STRING* specifies the name of the generated folder which is the output from *Function 1*. *FILE_PERMISSION* specifies the permissions granted to that folder, the default value '777' corresponds to the permissions read, write and execute for all users. The function makes use of the command line tool *mkdir* to generate the folder. All folder permission options can be found in the documentation page (41).

```
In [3]: FOLDER_PATH  
Out[3]: '/media/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/LEARNING/UniSheff/Mech/4/FYP/fyp-code/  
useful_bit/important_files/JzG'  
In [4]:
```

Figure 3 Function 2 output, variable FOLDER_PATH.

7.4.3 List All Available Download formats

Variable 1 Name downloaded content.

```
NNAME = '/Name of the downloaded video goes here without special characters'  
NNAME = NNAME.replace(' ', '_')
```

Variable *NNAME* in variable 1 contains the name assigned to the files downloaded. The slash '/' at the beginning must be included in order to store the files in the generated folder produced by *Function 2*. The second line substitutes all spaces as they are special characters and could become the source of future syntax errors.

Variable 2 Video URL.

```
INPUT_URL = 'youtube URL goes in here'
```

Variable *INPUT_URL* in Variable 2 contains the URL of the YouTube video. The URLs should be carefully selected to ensure there is no background noise in the video and that it contains primarily a single person talking in front of the camera. The most suitable YouTube videos are of news programs, podcasts, interviews and videos of educational purposes. Figure 4 shows various YouTube filtering options which could be used to identify suitable videos.

Multiple URLs correspond to exactly the same YouTube video. Each time a video is added to a playlist a different URL is assigned pointing to that video. Any URL

specified will work, however if a playlist URL is assigned youtube-dl will download the specified video and along with all other videos in the playlist. The *Tool* is only capable of processing one video at a time, hence any additional videos which have been downloaded will not be processed.

FILTER				
UPLOAD DATE	TYPE	DURATION	FEATURES	SORT BY
Last hour	Video X	Short (< 4 minutes) X	Live	Relevance
Today	Channel		4K	Upload date
This week	Playlist	Long (> 20 minutes)	HD	View count
This month	Film		Subtitles/CC X	Rating
This year	Programme		Creative Commons	
			360°	
			VR180	
			3D	
			HDR	
			Location	
			Purchased	

Figure 4 Youtube filtering options.

Function 3 Lists all available download formats.

```
INPUT_FILE_NAME=str(FOLDER_PATH) + NNAME
available_formats = module_youtube_extract.list_available_AV_formats(URL =
INPUT_URL, CLEAN=True, LIST=True, SAVE=True, FILE_NAME = INPUT_FILE_NAME +
'_down_formats')
```

Function 3 makes use of the command line tool youtube-dl to requests all available video and audio formats. *CLEAN* and *LIST* arguments specify different output formats which are stored in the variable *available_formats* (Figure 5). Each available format corresponds to its own unique identification code (Figure 5). The codes assigned to each format are fixed. If a video does not support the specifications of a particular format, the code of the format is excluded from the list (Figure 5). The *SAVE* argument specifies if we wish to save the result locally and the *FILE_NAME* argument specifies the desired file name.

136	mp4	1280x720	720p	944k , avc1.4d401f, 30fps, vi ...
248	webm	1920x1080	1080p	1742k , vp9, 30fps, video onl ...
137	mp4	1920x1080	1080p	2456k , avc1.640028, 30fps, v ...
22	mp4	1280x720	720p	326k , avc1.64001f, mp4a.40.2 ...
18	mp4	640x360	360p	350k , avc1.42001e, mp4a.40.2 ...
137	mp4	1920x1080	1080p	2456k , avc1.640028, 30fps, video only, 50.49MiB

Figure 5 *available_formats* variable.

7.4.4 Downloading YouTube Content

Function 4 Download youtube video.

```
module_youtube_extract.down_audio_video(URL = INPUT_URL, VIDEO_QUALITY_CODE=22, AUDIO_QUALITY_CODE=140, MERGE=False, FILE_NAME = INPUT_FILE_NAME)
```

Function 4 downloads any video from YouTube or from the list of websites supported by youtube-dl (40). The *URL* argument specifies the video to download, the *VIDEO_QUALITY_CODE* and *AUDIO_QUALITY_CODE* correspond to the desired audio and video code formats specified from the output of *Function 3*. The *MERGE* argument specifies whether or not we wish to merge the audio and video files or keep them as two separate files and the *FILE_NAME* specifies the name of the files. The audio and video files share the same names but different extensions. Youtube-dl mainly supports mp4 for video formats and m4a for audio.

7.4.5 Convert Audio File

Function 5 Convert audio to wav format.

```
module_convert_audio_to_wav.file_conversion_to_wav(FORMAT_FROM='.m4a', FILE_NAME=INPUT_FILE_NAME, BIT_RATE='192k')
whole_pure_audio_file_name_dir = INPUT_FILE_NAME + '.wav'
```

Function 5 makes use of FFmpeg to convert from any desired specified extension of the argument *FORMAT_FROM* to a wav file. The *BIT_RATE* argument specifies the bit rate (sampling rate or data points per second) of the new file. The variable *whole_pure_audio_file_name_dir* contains the absolute path of the wav file.

Conversion from m4a to a wav format is necessary in order to align the speech to text using the P2FA (42). The P2FA requires a wav audio file of 8kHz, 11,025kHz or 16kHz to execute (42).

7.4.6 Convert Video File

Function 6 Convert any-to-any video file.

```
video_converted_to_mkv = module_video_processing.convert_from_mp4_to_mkv(FILE_NAME=INPUT_FILE_NAME, INPUT_EXTENSION='.mp4', OUTPUT_EXTENSION = '.mkv')
```

Function 6 was developed in order to copy (does not encode or decode) the downloaded video of mp4 format from Function 4 into an mkv format. The conversion to mkv format results into less slicing errors than the mp4 format. The arguments are the *FILE_NAME* defined previously, the extension of the file *INPUT_EXTENSION* and the extension of the resulting file *OUTPUT_EXTENSION*.

The absolute path of the newly created file name is assigned to a variable. The function makes use of FFmpeg.

7.4.7 Download Subtitle Formats

Function 7 Extract available subtitle formats.

```
string_subtitle_formats, manual_subtitles_exist, automatic_subtitles_exist
= module_youtube_extract.list_available_subtitles(URL = INPUT_URL, FILE_N
AME = INPUT_FILE_NAME, TXT_CLEAN=False, TXT=False, JSON=False)
```

Function 7 with the underline use of youtube-dl makes a request from the web server for all available subtitles of a specific URL. The arguments *TXT* and *JSON* specify in which file format to save the subtitles, as a text file or a json file. *TXT_CLEAN* specifies a clean save format if the *TXT* argument is specified as well. The outputs of the function are the string variable *string_subtitle_formats* containing all available formats as shown on Figure 6. The output *manual_subtitles_exist* is assigned the Boolean values True or False based on whether or not the owner has added their own subtitles manually to the YouTube video. The output *automatic_subtitles_exist* specifies if the owner has enabled automatically generated subtitles to the video. Similarly, a Boolean value is assigned.

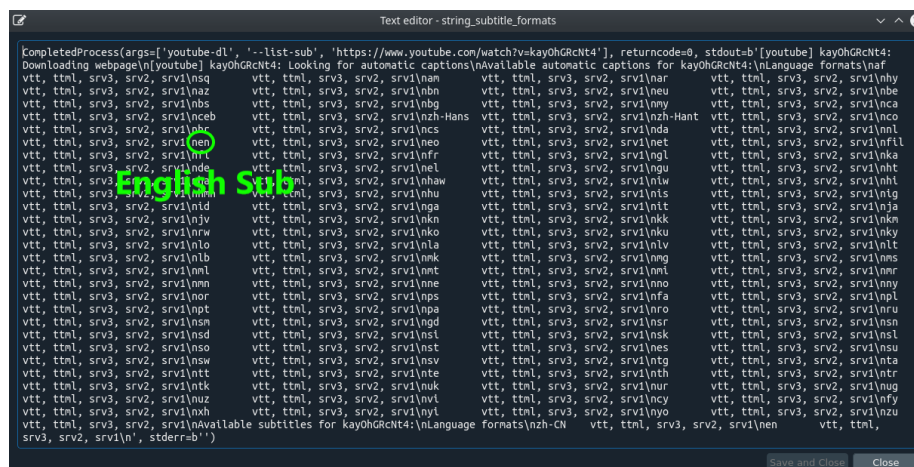


Figure 6 string_subtitle_formats

7.4.8 Download Subtitles

Function 8 Download subtitles.

```
man_sub_var, auto_sub_var = module_youtube_extract.down_sub(URL = INPUT_URL,
FILE_NAME=INPUT_FILE_NAME+'SUBTITLES', TYPE='vtt', LANGUAGE='en', MAN_SUB =
manual_subtitles_exist, AUTO_SUB = automatic_subtitles_exist, SAVE=True)
```

Function 8 downloads the manually added subtitles of the owner and the automatically generated subtitles if they have been enabled and stores the results in

variables `man_sub_var` and `auto_sub_var`. The function makes underline use of `youtube-dl` and accepts the outputs from Function 7 for arguments *MAN_SUB* and *AUTO_SUB*. The *TYPE* and *LANGUAGE* arguments specify the file type in which subtitles will download into and the desired language within the list in the variable `string_subtitle_formats`.

7.4.8.1 Subtitle Combinations

Four potential cases have been encountered which require different handling procedures.

The first case is encountered when the video does not contain neither automatically generated subtitles nor manually added subtitles. This could be caused due to the fact that the video does not contain anyone speaking or the owner did not add their own subtitles or the option of automatically generating subtitles was disabled by the owner. In this particular case, since subtitles are absent, subtitles could either be manually added by listening to the video and typing them into a text file or automatically generating them using a speech-to-text machine learning model such as DeepSpeech. It should be noted that speech-to-text models are not perfect and errors (incorrect dictations) could occur.

The second case scenario would be when either the user has posted their own subtitles but has disabled automatically generated subtitles. The time range, of when each sentence in the subtitles is presented in the video, is also contained in the subtitle file. However, time frame of each spoken word is unknown hence text to audio alignment is needed.

The third case is when only automatically generated subtitles are available. One of the excellent properties of YouTube's automatically generated subtitles is the alignment of the subtitles per word. As a result, in this case where subtitles are aligned per word, an aligner such as P2FA is not needed.

The last case is when both automatically generated subtitles and manually added subtitles are available.

7.4.8.2 Subtitle Formats

From all four cases, it is made clear that the videos which fall under the case in which subtitles are automatically generated and aligned produces the easiest and most convenient case as it avoids the need of an aligner. Since the MVP (Design Approach 6.1) could be built without the use of an aligner, the implementation of

P2FA was skipped at this stage and might be implemented after the realisation of the MVP.

By testing a wide range of videos (Appendix 12.5) the following cases have been encountered for both manually added subtitles and automatically generated subtitles:

1. Sentence subtitles with alignment per sentence as shown in Figure 7.
2. Sentence subtitles with both alignment per word and per sentence as shown in Figure 8.

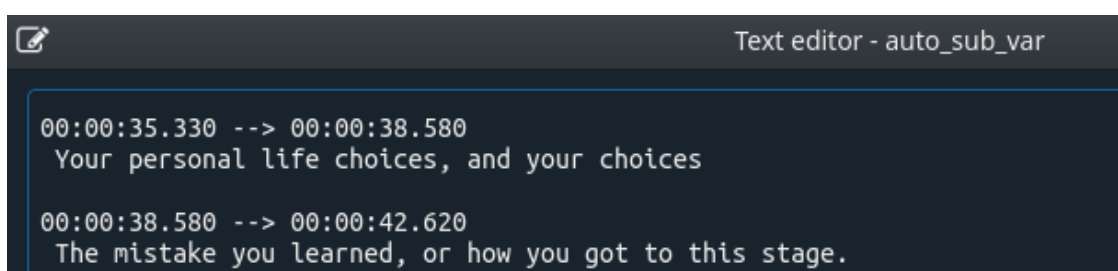


Figure 7 Raw subtitle example with alignment both per word and per sentence.

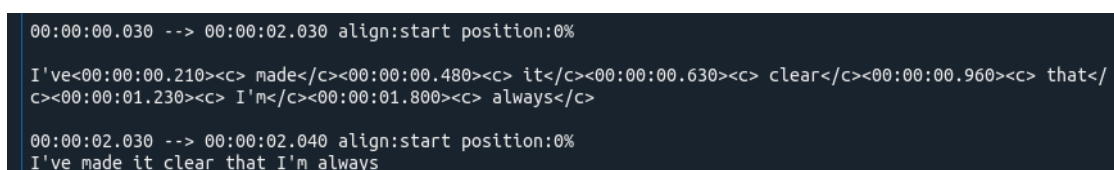


Figure 8 Raw subtitle example with alignment of both per word and per sentence.

Subtitle variables have been created to account for each case scenario which resulted into six different variables in total. These variables are separated further into lists per text (Figure 10) and their time values (Figure 11), hence a total of 12 subtitle variables are needed. These subtitles are listed in Table 3 with the following notation 'auto/man' '_content/time' '_per_word/sentence' '_easy/hard'.

- auto: automatically generated, man: manually added by the owner.
- content: contains the subtitles, time: times presented in the video.
- word: subtitles per word, sentence: subtitles per sentence.
- easy: contains timings per word, hard: contains timings per sentence.

Table 3 Classified subtitle variable names.

acpwe: auto_content_per_word_easy, atpwe: auto_time_per_word_easy,
--


```

mcpwe: man_content_per_word_easy, mtpwe: man_time_per_word_easy,
acpse: auto_content_per_sentence_easy, atpse: auto_time_per_sentence_easy,
acpsh: auto_content_per_sentence_hard, atpsh: auto_time_per_sentence_hard,
mcpse: man_content_per_sentence_easy, mtpse: man_time_per_sentence_easy,
mcpsh: man_content_per_sentence_hard, mtpsh: man_time_per_sentence_hard,
acpse: auto_content_per_sentence_easy, mtpse: man_time_per_sentence_easy,
mcpsh: man_content_per_sentence_hard, mtpsh: man_time_per_sentence_hard

```

7.4.9 Process Subtitles

Function 9 Categorize subtitles.

```

acpwe, atpwe, acpse, atpse, acpsh, atpsh, mcpwe, mtpwe, mcpse, mtpse, mcpsh
, mtpsh = module_process_subtitles.format_sub(MAN_SUB_EXIST = manual_subtit
les_exist, AUTO_SUB_EXIST = automatic_subtitles_exist, MAN_SUB_EASY_TYPE =
man_sub_easy_type, AUTO_SUB_EASY_TYPE = auto_sub_easy_type, MAN_SUB = man_s
ub_var, AUTO_SUB = auto_sub_var)

```

All arguments of Function 9 make use of the output result variables produced by Functions 7 and 8. Function 9 creates lists of the classified subtitle variable names contained in Table 3. For every video Function 9 produces a maximum of 4 non-empty variables out of the 12 in total. The remaining variables are assigned the Nonetype object as shown on Figure 9.

Name	Type	Size	Value
LANDMARK_DATAFRAME_array	Series	(213,)	Series object of pandas.core.series module
LENGTH_OF_LANDMARK_DATAFRAME_array	int	1	213
acpse	list	268	['good evening after facing days of', 'criticism for avoiding media sc ...
acpsh	NoneType	1	NoneType object
acpwe	list	1642	['good', 'evening', 'after', 'facing', 'days', 'of criticism', 'for', ...
atpse	list	268	['00:00:00.030 --> 00:00:03.050', '00:00:03.060 --> 00:00:05.030', '00 ...
atpsh	NoneType	1	NoneType object
atpwe	list	1642	['<00:00:00.030>', '<00:00:00.780>', '<00:00:01.159>', '<00:00:02.159> ...

Figure 9 All subtitle variables.

Both acpwe and atpwe variables are of the same length (Figure 9), acpwe contains individual words (Figure 10) and atpwe contains the time (start time) each word is presented in the video (Figure 11).

Inde ▲	Type	Size	
0	str	1	Ive
1	str	1	made
2	str	1	it
3	str	1	clear
4	str	1	that

Figure 10 Example acpwe variable.

Inde ▲	Type	Size	
0	str	1	<00:00:00.030>
1	str	1	<00:00:00.210>
2	str	1	<00:00:00.480>
3	str	1	<00:00:00.630>
4	str	1	<00:00:00.960>

Figure 11 Example atpwe variable.

Since the atpwe variable contains only the start time values, the ending of the spoken word is unknown. To overcome this problem the stop time is assumed to be the start value of the subsequent word in the list. There is clearly a limitation of using the particular technique as it fails to accurately capture single word samples if pauses are contained during someone's speech. The *Tool* will keep capturing until the next presented word appears in the video which will result into a defective sliced video segment. This is one of the limitations of the *Tool* as it does not deploy audio to text alignments (P2FA) or SyncNet to which will allow it to distinguish between a person's speech and background music as YouTube ASR only provides the starting value for each word. With the current development of the *Tool* this problem is overcome using three different approaches:

1. Selecting videos which contain an uninterrupted talking face.
2. Avoiding performing slicing between time boundaries containing interruptions or long pauses of the speaker.
3. Completely discard word aligned subtitles for the particular video.

For variables which contain whole sentences such as acpse (Figure 12), their matching time variables such as atpse (Figure 13) contain both the start and stop time values (boundaries) per sentence as shown in Figure 13.

Inde ▲	Type	Size	
0	str	1	good evening after facing days of
1	str	1	criticism for avoiding media scrutiny
2	str	1	Boris Johnson whos running against
3	str	1	Jeremy Hunt to be the next prime
4	str	1	minister and Tory leader has spoken
5	str	1	exclusively to the BBC he says he would

Figure 12 Example of acpse.

Inde ▲	Type	Size	
0	str	1	00:00:00.030 --> 00:00:03.050
1	str	1	00:00:03.060 --> 00:00:05.030
2	str	1	00:00:05.040 --> 00:00:07.280
3	str	1	00:00:07.290 --> 00:00:08.750
4	str	1	00:00:08.760 --> 00:00:10.850
5	str	1	00:00:10.860 --> 00:00:13.610

Figure 13 Example of atpse.

7.4.10 Max Video Length

Function 10 Get the maximum time length of the video.

```
max_time = module_video_processing.maximum_time_of_vid(ATPSE=atpse, MTPSE=
mtpse, AUTO=automatic_subtitles_exist, MAN= manual_subtitles_exist)
if max_time == None:
    max_time = module_video_processing.maximum_time_of_vid(ATPSE=atpsh, MT
PSE=mtpsh, AUTO=automatic_subtitles_exist, MAN= manual_subtitles_exist)
else:
    pass
```

Function 10 makes use of the outputs from Function 9 and 7 to compute the maximum time of the video. The function requires at least one of the variables atpse, mtpse, atpsh and mtpse to be a list and not a Nonetype object as shown on Figure 9. The output of the function is the maximum length of the video in seconds which will be stored in max_time as a string.

7.4.11 Replace Special Characters

Function 11 Replace special characters.

```
acpwe = module_process_subtitles.remove_or_replace_special_char(INPUT_SUB_
LIST = acpwe, CHAR_TO_REPLACE = 'all', CHAR_TO_REPLACE_WITH = '')
```

Function 11 is used to replace special characters (Variable 3) of the variables listed in Table 3. This is an important step as it prevents many syntax errors due to the handling of these special characters from following processes such as slicing the samples. For instance the quote character (') which is commonly present in text is also used by python to interpret the beginning and ending of strings. The argument *INPUT_SUB_LIST* handles one of the variables (at a time) created by Function 9 which contains either sentence or word subtitles. Function 11 performs character replacement onto that variable. The output variable is identical to the value of the *INPUT_SUB_LIST* with the specified character(s) removed or replaced. The argument *CHAR_TO_REPLACE* specifies which character is to be replaced from *INPUT_SUB_LIST* and *CHAR_TO_REPLACE_WITH* the replacement character. The function is capable of substituting and replacing any character which can be interpreted by python 3.

The function makes underline use of the built in function `.replace()` which is capable of replacing characters within strings. Apart from replacing single characters, the function is capable of replacing all special characters simply by specifying the value 'all' to the argument *CHAR_TO_REPLACE*. The special characters which will be replaced are listed in Variable 3 which can be generated using the `string.punctuation()` command. Hence, the string 'all' is reserved and cannot not be

substituted. However, as the string 'all' does not contain any special characters there is no need for it to be replaced, as it is incapable of introducing unexpected system errors.

Variable 3 Special characters replaced by Function 11.

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

For the sake of simplicity the default value replaces all special characters. However, the best approach would be to replace special characters contained within the subtitles with code strings. Once all processes which require subtitle manipulations have completed, the Function 11 could be used again to replace the code strings back to the special characters.

7.4.12 Save Subtitles

Function 12 Save subtitles.

```
module_save_variables.save_sub(VAR_INPUT=acpwe, FILE_NAME=INPUT_FILE_NAME+
'_acpwe', TXT=False, JSON=True, TXT_SEPARATOR = '\n')
```

Function 12 saves the formatted subtitles (output of Function 11) locally which contain the subtitles pre-sentence or per-word. The *VAR_INPUT* argument specifies which subtitle variable to save. The *TXT* and *JSON* arguments specify two potential save formats and the *TXT_SEPARATOR* argument specifies how the list specified in *VAR_INPUT* will be assembled into a single and continuous string which will be stored in the text file.

7.4.13 Chopped Sample Folder

Function 13 Generate chopped sample folder.

```
chopped_sample_per_word_folder_dir_acpwe = module_sample_name.folder_gen(R
ANDOM_STRING = FOLDER_PATH + '/chopped_samples_per_word_acpwe', FILE_PERMI
SION = '777')
```

Function 13 automatically generates a folder in which sliced segments of the video will be contained. The argument *RANDOM_STRING* sets the name (absolute path) of the generated folder and the output 'chopped_sample_per_word_folder_dir_acpwe' contains the folder's absolute path.

7.4.14 Slicing

Function 14 Slice the video into segments.

```
word_chunk_samples_info_acpwe = module_video_processing.chop_video_per_word_or_sentence(LIST_PER_WORD = acpwe, TIMES_PER_WORD = atpwe, MAX_TIME = max_time, FILE_NAME = INPUT_FILE_NAME, CHOPPED_SAMPLE_FOLDER_DIR = chopped_sample_per_word_folder_dir_acpwe, SAVE_FILE_NAME = chopped_sample_per_word_folder_dir_acpwe + '/word_chunk_samples_info_acpwe.csv', SHIFT_RIGHT_OR_LEFT = 0, EXTEND_LEFT = -150, EXTEND_RIGHT = 150, EXTENSION = '.mkv', START_INDEX = 0, STOP_INDEX = 5, SAVE = True)
```

Function 14 makes underline use of FFmpeg to slice the video into segments per word or per sentence depending on what variables are passed (acpwe, acpse, atpwe etc.) to arguments *LIST_PER_WORD* and *TIMES_PER_WORD*. The function does not remove or edit the original file as it creates mere copies at the time boundaries specified in *TIMES_PER_WORD*. The output of Function 10 is passed to the argument *MAX_TIME*, *FILE_NAME* specifies the absolute path of the whole video and *EXTENSION* the file's extension. The argument *CHOPPED_SAMPLE_FOLDER_DIR* specifies the directory folder to save the generated segments. *SAVE_FILE_NAME* will be used as the name of the csv file to save the file names of the generated segments if the *SAVE* argument is set to True. The function slices each word or sentence in accordance to the corresponding boundaries contained in *TIMES_PER_WORD*. The *SHIFT_RIGHT_OR_LEFT* argument allows the user to apply shifting onto the slicing time values (both start and stop values). *EXTEND_LEFT* shifts only the start value whereas *EXTEND_RIGHT* shifts only the stop value. All operations which alter the slicing positions require an integer input and correspond to millisecond shifts. The *START_INDEX* and *STOP_INDEX* specify the range of words to be sliced by index according to the value passed to the argument *LIST_PER_WORD*. For instance, the default values (*START_INDEX* = 0 and *STOP_INDEX* = 0) will produce segments containing the words or sentences in the list shown on Figure 9. The function is capable of slicing any available format supported by FFmpeg.

If shifting, which exceeds the boundaries of the video is applied, the function will slice only up to the last possible value on that segment. For example if the resulting extensions and shifts from arguments *SHIFT_RIGHT_OR_LEFT*, *EXTEND_LEFT* and *EXTEND_RIGHT* result in slicing at 400 seconds when the whole video is only 390 seconds long, the slicing will occur at 390 seconds.

The output of the function is a pandas (python 3 module) dataframe as shown in Figure 14 which contains useful information of the segments such as the absolute path, duration, content etc.

word_chunk_samples_info_acpwe - DataFrame						
Index	File Name	Start Time (with extension)	End Time (with extension)	Duration (with extension)	Word Content	INDEX
0	/media/.../9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/.../LEARNING/UnlSheff/Mech/4/FYP/fyp-code/.../useful_bit/important_files/Vyz/chopped_samples_per_word_acpwe/AUTO_SEGMENT_good_0.mkv	00:00:00.030	00:00:00.930	00:00:00.930	good	0
1	/media/.../9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/.../LEARNING/UnlSheff/Mech/4/FYP/fyp-code/.../useful_bit/important_files/Vyz/chopped_samples_per_word_acpwe/AUTO_SEGMENT_eventing_1.mkv	00:00:00.630	00:00:01.309	00:00:00.679	eventing	1
2	/media/.../9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/.../LEARNING/UnlSheff/Mech/4/FYP/fyp-code/.../useful_bit/important_files/Vyz/chopped_samples_per_word_acpwe/AUTO_SEGMENT_after_2.mkv	00:00:01.009	00:00:02.309	00:00:01.300	after	2
3	/media/.../9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/.../LEARNING/UnlSheff/Mech/4/FYP/fyp-code/.../useful_bit/important_files/Vyz/chopped_samples_per_word_acpwe/AUTO_SEGMENT_facing_3.mkv	00:00:02.009	00:00:02.700	00:00:00.691	facing	3
4	/media/.../9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/.../LEARNING/UnlSheff/Mech/4/FYP/fyp-code/.../useful_bit/important_files/Vyz/chopped_samples_per_word_acpwe/AUTO_SEGMENT_days_4.mkv	00:00:02.400	00:00:02.940	00:00:00.540	days	4

Figure 14 Function 14 output, segment info dataframe.

Figure 15 shows a segment produced from Function 14. Segment names are automatically generated in the following format 'AUTO/MAN_SEGMENT_word/sentence_index.extension'. The first word is used to distinguish which subtitle type the segment belongs too, AUTO for automatically generated or MAN for manually added by the owner. The second word, SEGMENT, separated by an underscore and it is added to all segments indicating that the current file is a segment and not the file. The third word contains the word been spoken within the segment file, if a sentence is been spoken the words are separated by a hyphen (e.g. hi-there). The fourth and final value prior the extension of the file is a number which specifies the index of the word in the input of *LIST_PER_WORD*.

Keeping a record of the index could be used to track the prior and upcoming words which is important as it could be used to investigate coarticulation. The segment in Figure 15 contains only the word 'cyber' which was spoken in less than 1000 milliseconds causing the end value of the video to display 00:00. Figure 16 shows an example end-result of Function 14.

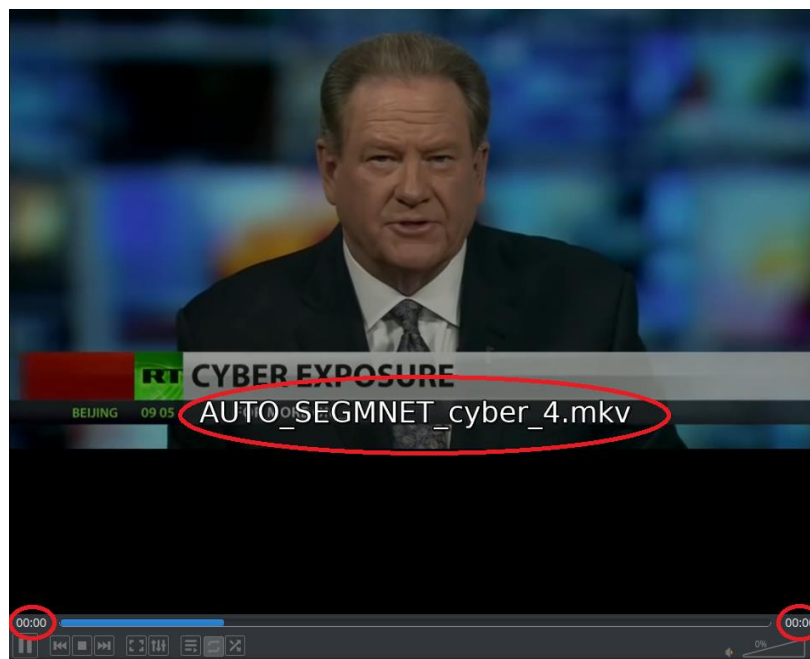


Figure 15 Example of a segmented sample produced from Function 14.

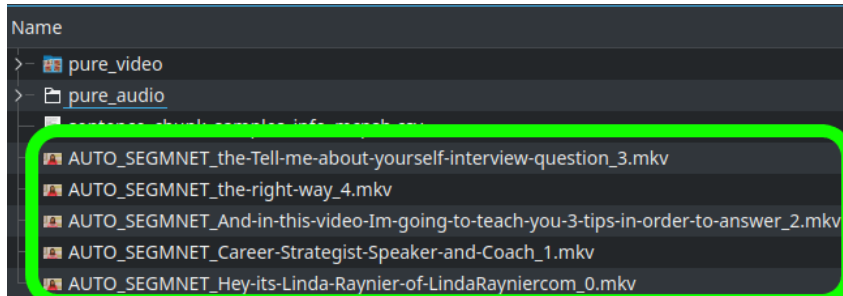


Figure 16 Example output file segments generated by Function 14.

7.4.15 Final Result Folders

Function 15 Pure audio and video folder generation.

```
chopped_sample_per_word_folder_dir_pure_audio_acpwe = module_sample_name.folder_gen(RANDOM_STRING = chopped_sample_per_word_folder_dir_acpwe + '/pure_audio', FILE_PERMISSION = '777')

chopped_sample_per_word_folder_dir_pure_audio_acpwe = module_sample_name.folder_gen(RANDOM_STRING = chopped_sample_per_word_folder_dir_acpwe + '/pure_audio', FILE_PERMISSION = '777')
```

Function 13 is run again as shown in the block of Function 15 to generate new folders named 'pure_audio' and 'pure_video' shown on Figure 16. In this folders the following end-results will be stored: 1) pure audio segmented files, 2) pure video segmented files and 3) pure video cropped segmented files. Function 15 outputs the absolute paths of the folders and stores them in the variables assigned.

7.4.16 Final Audio Files

Function 16 Extract the audio.

```
module_video_processing.extract_audio_from_list_mkv_files(INPUT_FILE_NAME = LLIST_INPUT_WORD_CHUNK_SAMPLES_ACPWE_FILE_NAMES, OUTPUT_FILE_NAME = LIST_OUTPUT_AUDIO_WORD_CHUNKED_ACPWE_NAMES, BIT_RATE = '192000')
```

Function 16 extracts the audio from the segmented files and stores them into the folder named pure_audio with the use of FFmpeg. The function's arguments *INPUT_FILE_NAME* and *OUTPUT_FILE_NAME* accept variables in the form of lists which contain the input video file names and the desired output audio names. Hence, all segments are converted with the use of a single function. The format of the audio files is specified by including the desired extension in the output file names. The *BIT_RATE* argument specifies the bit rate of the newly created files. Figure 17 shows an example output result of Function 16.

Name
🎵 AUTO_SEGMNET_the-Tell-me-about-yourself-interview-question_3.wav
🎵 AUTO_SEGMNET_the-right-way_4.wav
🎵 AUTO_SEGMNET_And-in-this-video-Im-going-to-teach-you-3-tips-in-order-to-answer_2.wav
🎵 AUTO_SEGMNET_Hey-its-Linda-Raynier-of-LindaRayniercom_0.wav
🎵 AUTO_SEGMNET_Career-Strategist-Speaker-and-Coach_1.wav

Figure 17 Example result of pure audio generated segment file from the YouTube video: <https://www.youtube.com/watch?v=kayOhGRcNt4> .

7.4.17 Remove Audio

Function 17 Remove the audio.

```
module_video_processing.remove_audio_from_list_mkv_files(INPUT_FILE_NAME =
LLIST_INPUT_WORD_CHUNK_SAMPLES_ACPWE_FILE_NAMES, OUTPUT_FILE_NAME = LIST_OUTPU
TPUT_VIDEO_WORD_CHOPPED_ACPWE_NAMES)
```

Function 17 with the use of FFmpeg produces a copy of the video stream of the segmented file names and stores the result into the pure_video folder. Similarl to Function 16, Function 17 accepts list variables for arguments *INPUT_FILE_NAME* and *OUTPUT_FILE_NAME* which contain the paths of the input and output files. The desired output format is specified from the extension included in the output file name. An example result is shown in Figure 18.

🎥 AUTO_SEGMNET_the-right-way_4.mkv
🎥 AUTO_SEGMNET_the-Tell-me-about-yourself-interview-question_3.mkv
🎥 AUTO_SEGMNET_Hey-its-Linda-Raynier-of-LindaRayniercom_0.mkv
🎥 AUTO_SEGMNET_Career-Strategist-Speaker-and-Coach_1.mkv
🎥 AUTO_SEGMNET_And-in-this-video-Im-going-to-teach-you-3-tips-in-order-to-answer_2.mkv

Figure 18 Pure video segments produced by Function 17.

7.4.18 Face Recognition and Cropping

Function 18 Crop and extract facial landmarks.

```
pure_video_cropped_word_chunk_samples_info_acpwe = module_face_detection.mu
ltiple_file_camera_face_rec_and_cropping(LIST_OUTPUT_FILE_NAME = LIST_OUTPU
T_VIDEO_WORD_CHOPPED_ACPWE_NAMES, LIST_OF_INPUT_FILE_NAME = LIST_OUTPUT_VID
EO_WORD_CHOPPED_ACPWE_NAMES, FOURCC1='M', FOURCC2='J', FOURCC3='P',FOURCC4
='G', ADD_STR_CROPPED_FILE_NAME = '_cropped', INPUT_FILE_NAME_EXTENSION = '
.mkv', OUPUT_FILE_NAME_EXTENSION = '.avi', CROPPED_WIDTH = 110, CROPPED_HEI
GHT = 105, SHIFT_RIGHT = -50, SHIFT_DOWN = 0, OUTPUT_FPS = 'same',
ENABLE_FACE_RECOGNITION_TRACKING_CROPPING = True, WHOLE_FACE_PROFILE = False
, LIPS_PROFILE = False, LOAD_FACE_LANDMARKS = True, POINT_LAND_MARK_TRACKIN
G = False, LAND_MARK_TRACKING_NUMBER = 1, LAND_MARK_LIP_TRACKING = True, CA
PTURE_FACE_LANDMARKS = True, DISPLAY_FACE_LANDMARKS = False, SAVE_LANDMARK
```



```
TRACKING_RESULTS = False, SAVE_LANDMARK_TRACKING_RESULTS_NAME = 'Record', SAVE_WHOLE = False, SAVE_CROPPED = True, DISPLAY_WHOLE = False, DISPLAY_CROPPED = False, ENABLE_CUBIC_LAND_MARK_TRACKING = False, CUBIC_LAND_MARK_POINT_TOP = 34, CUBIC_LAND_MARK_POINT_LEFT = 49, CUBIC_LAND_MARK_POINT_BOTTOM = 9, CUBIC_LAND_MARK_POINT_RIGHT = 55, FLIP = True, FLIP_ARGUMENT = 1, SHAPE_PREDICTOR_NUMBER_OF_LANDMARK_POINT_START = 48, SHAPE_PREDICTOR_NUMBER_OF_LANDMARK_POINT_STOP = 68)
```

Function 18 applies face recognition, extracts facial landmarks and creates new files containing only the cropped region of the lips as shown on Figure 19.



Figure 19 Example result produced by Function 18. The same sample shown in Figure 15 was processed to produce the result of Figure 19.

By making use of the cv2 python library the function decodes files and processes each frame separately. The function contains a wide range of options which could be used to customise the output. The function is capable of handling multiple files simultaneously and hence the function only needs to be called once. The *LIST_OF_INPUT_FILE_NAME* contains the absolute path of the files in the form of a list and the *INPUT_FILE_NAME_EXTENSION* their extension. The *LIST_OUTPUT_FILE_NAME* requires a list of the output file names and *OUTPUT_FILE_NAME_EXTENSION* their extension. The function supports *SAVE_WHOLE* and *SAVE_CROPPED* options which save the results as whole frames (same as input) or cropped frames (desired result). In case both saved arguments are set to True, the argument *ADD_STR_CROPPED_FILE_NAME* allows the user to add a string at the end of the cropped output file names to prevent some of the files been over written due to the same assigned name. In addition, by doing so, the output file names could share the same names as the input filenames with the distinction of the additional string at the end. Figures 18 and 20 show an example where the additional string is ‘_cropped’ is added to the newly created files.

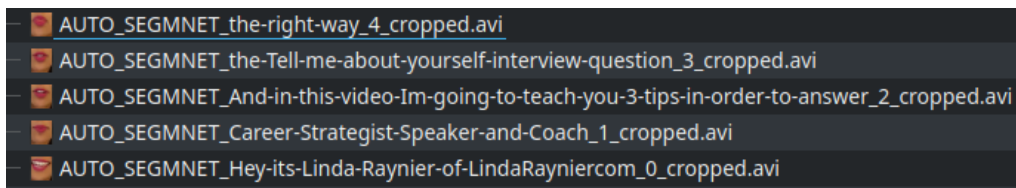


Figure 20 Cropped samples produced by Function 18.

The argument *OUTPUT_FPS* specifies the fps of the output video. The argument accepts an integer value and it is important to encode the video with the same fps as the input files. The fps of the downloaded video can be found from the output of Function 3. In addition, by assigning the string 'same' to the argument *OUTPUT_FPS* the function is programmed to detect and automatically assign the input fps to the output file.

The *FFOURCC* arguments specify the form of compression. The default value is set to 'MJPG'. Other formats could be specified (Appendix 12.7) such as 'RGBA' which is an uncompressed lossless format.

7.4.18.1 Face Cropping Options

Function 18 provides a wide range of face cropping options to allow the user to customize the dataset to their preference by containing only the necessary parts of the face. By storing the samples containing only the region of the mouth the identity of the people in the video is protected in addition to the reduction of used space. For instance, Figure 19 contains the person shown in Figure 15, without providing any other information such as the origin of the video, it is very difficult to recognise a person merely from Figure 19. This feature becomes extremely important if local recordings take place (6.2 Design Matrix approach 2) in order to meet the ethical requirements mentioned in section 5. Function 18 supports real time recording and it is activated by passing the 0 integer value to the argument

LIST_OF_INPUT_FILE_NAME.

However, this should only be used to test the various settings of the function and not for the purpose of enlarging the dataset for the reasons described in paragraph 6.2.1. The *DEMO_face_detect.py* file essentially consists of various pre-

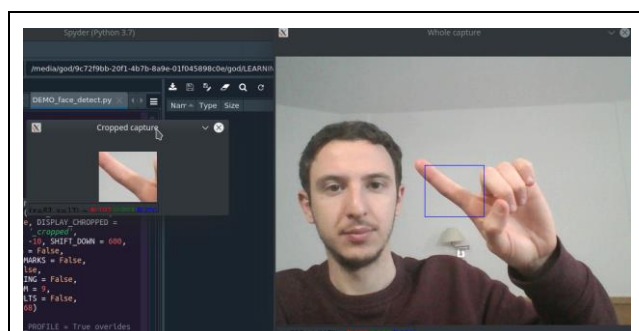


Figure 21 Disabled face recognition and extracting a fixed pre-defined square.

set example settings of Function 18 and their descriptions which could be tested. By assigning the True value to arguments *DISPLAY_WHOLE* and *DISPLAY_CROPPED* the processing and effect of other settings onto the video can be displayed in real time.

The *FLIP* argument enables the flipping of the frames and the *FLIP_ARGUMENT* sets the axis in which they will be flipped (0: vertical, >0: horizontal and <0 both).

The argument *ENABLE_FACE_RECOGNITION_TRACKING_CROPPING* enables or disables the face recognition. In Figure 21, face recognition is disabled and the function records a pre-fixed rectangular region at the centre of the display. The reticular region is programed to adjust itself to the centre regardless of the dimensions of the captured display (position is not hard coded).

However, the user has the ability to change the position of the rectangle using the *SHIFT_RIGHT* and *SHIFT_DOWN* arguments. The arguments accept integer values and displace the rectangle per pixel. The size of the desired cropped region can be adjusted through the *CROPPED_WIDTH* and *CROPPED_HEIGHT* arguments. The arguments accept integer values and adjust the rectangle per pixel.

If the *ENABLE_FACE_RECOGNITION_TRACKING_CROPPING* is set to True, face recognition is deployed with the use of `dlib.get_frontal_face_detector()`. The detector outputs the upper left and lower right coordinates of a rectangle which enclose the face of a person in the video. By using those coordinates in a combination to shifting and sizing, the cropped region can be adjusted to capture specific parts in the frame such as the region of the lips. Two pre-configured profiles have been developed in Function 18. The profiles are tweaked to crop the region of the lips (Figure 22) and of the whole face (Figure 23). Only one of the profiles can be used at a time and are set by assigning the True value to arguments *WHOLE_FACE_PROFILE* and *LIPS_PROFILE*. Figures 22 and 23 demonstrate the profiles.

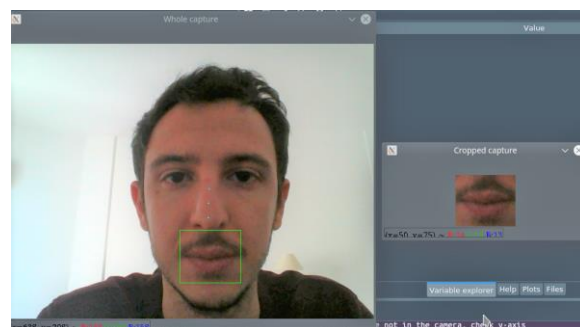


Figure 22 Crop the region of the lips profile (*LIPS_PROFILE* = True).

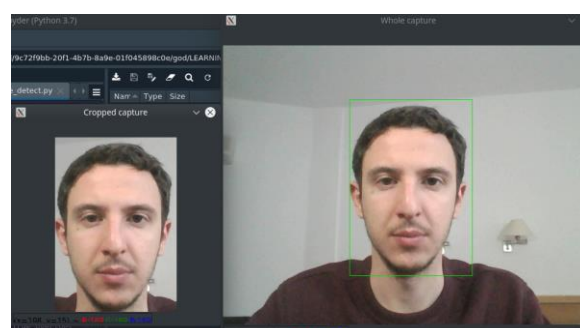


Figure 23 Crop the whole face profile (*WHOLE_FACE_PROFILE* = True).

7.4.18.2 Facial landmarks

According to the literature, facial landmarks can be used to solve lip-reading, perform face tracking and identify the speaker in a video (2) (14). The *Tool* makes use of the 68 point landmark predictor which is loaded from the file 'shape_predictor_68_face_landmarks.dat' (44). Other predictors capable of identifying more facial points could have been used. However, the particular predictor is used as it is pre-trained (does not require training on a face recognition dataset) and is freely available on GitHub [43]. By assigning the *LOAD_FACE_LANDMARKS* argument to True, Function 18 loads the facial landmarks using the function 'dlib.shape_predictor()'. The argument *CAPTURE_FACE_LANDMARKS* enables the capturing of facial landmarks

and must be set to True in order to capture the landmarks. The *DISPLAY_FACE_LANDMARKS* paints the landmark points onto the frames and should only be used for testing and demonstration purposes. The function allows the user to specify the range of landmarks to be detected. Figure 24 shows the position and numerical label of each landmark for the particular predictor. The argument *SHAPE_PREDICTOR_NUMBER_OF_LANDMARK_POINT_START* sets the start landmark and *SHAPE_PREDICTOR_NUMBER_OF_LANDMARK_POINT_STOP* the stop landmark. According to Figure 24, the range of landmarks which cover only the region of the lips start at number 49 and end at 68. Figure 26 shows the implementation of extracting landmarks at the region of the lips and Figure 25 of the whole face. This is an important feature as it prevents the identification and storage of

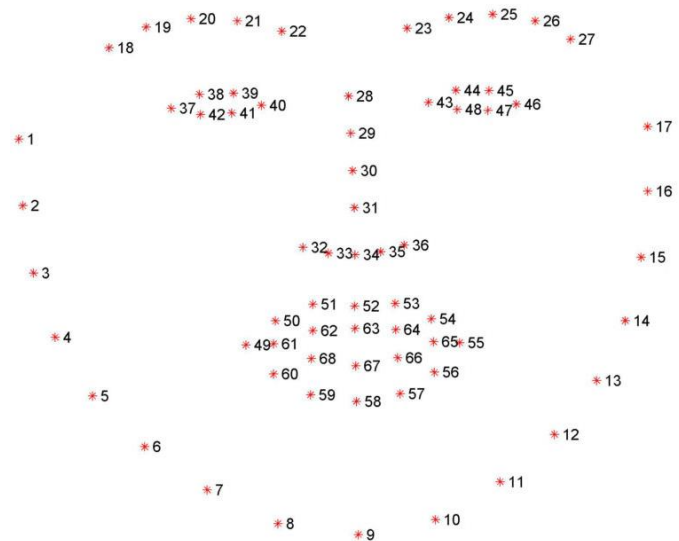


Figure 24 Label and position of each landmark (43).

The argument *SHAPE_PREDICTOR_NUMBER_OF_LANDMARK_POINT_START* sets the start landmark and *SHAPE_PREDICTOR_NUMBER_OF_LANDMARK_POINT_STOP* the stop landmark. According to Figure 24, the range of landmarks which cover only the region of the lips start at number 49 and end at 68. Figure 26 shows the implementation of extracting landmarks at the region of the lips and Figure 25 of the whole face. This is an important feature as it prevents the identification and storage of

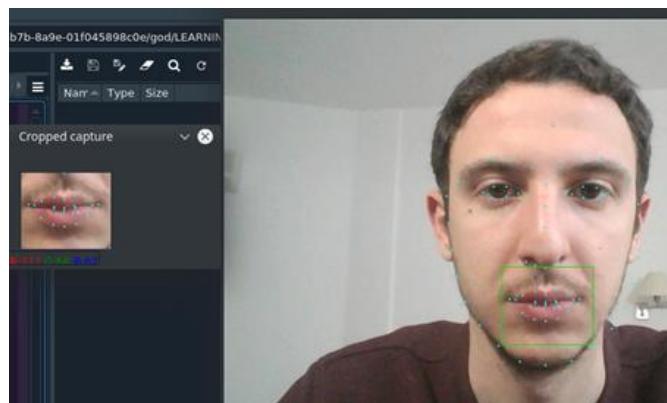


Figure 25 Extraction of whole face landmarks and tracking per landmark.

full facial landmarks which could be used to UI a person.

Landmark coordinates could also be used to track and crop parts of the face as an alternative method to face recognition points. The displacement of landmark points between consecutive frames is much smaller than the displacement of face recognition coordinates. As a result, tracking with the use of landmark points produces a

more stable result. However, this holds true only for landmark points such as point labelled 28 (Figure 24) which mimic the face's movements and are not respectively influenced by of the movements of facial muscles.

By assigning *POINT_LAND_MARK_TRACKING* equals to True the *WHOLE_FACE_PROFILE* and *LIPS_PROFILE* become redundant and tracking is performed based on the specified landmark point assigned to argument *LAND_MARK_TRACKING_NUMBER*.

The argument *LAND_MARK_LIP_TRACKING* is a pre-set profile which applies tracking and cropping at the region of the lips (shown on Figures 25 and 26) with the use of the landmark point labelled 34 on Figure24. Shifting and enlargement functions of the cropped region can also be applied in parallel, as shown in Figure 27. Landmark point tracking could also be deployed if *CAPTURE_FACE_LANDMARKS* is set to False.

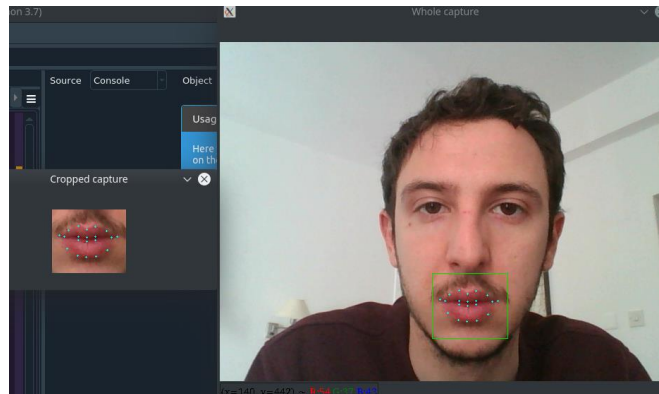


Figure 26 Extraction of lip landmarks and tracking per landmark.

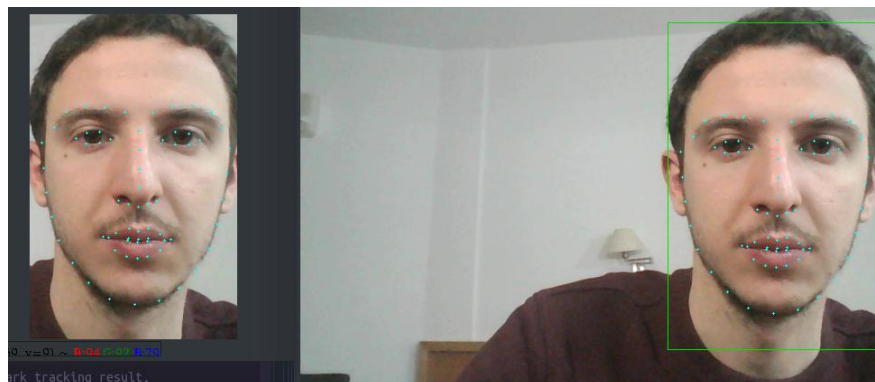


Figure 27 Exceeding the camera boundaries on the x-axis.

The final feature Function 18 supports for cropping the frames is the cubic landmark tracking. By assigning argument *ENABLE_CUBIC_LAND_MARK_TRACKING* to True, the shifting and fixed size of the cropped region is disabled and the cropped rectangular region is defined by four landmark points. All sides of the rectangle are specified from the following arguments *CUBIC_LAND_MARK_POINT_TOP*, *CUBIC_LAND_MARK_POINT_LEFT*, *CUBIC_LAND_MARK_POINT_BOTTOM* and *CUBIC_LAND_MARK_POINT_RIGHT*. These arguments accept numerical integer values from the label shown in Figure 24. Figures 28 and 29 demonstrate the behaviour of this setting and how the cropped region alters its shape.

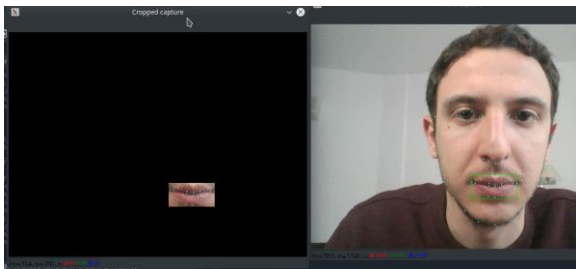


Figure 28 Cubic landmark tracking phase 1.

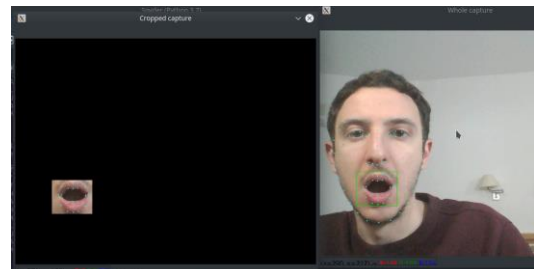


Figure 29 Cubic landmark tracking phase 2.

Finally, Function 18 supports features to save the landmark coordinates per frame locally by setting the argument *SAVE_LANDMARK_TRACKING_RESULTS* equals to True and specifying the desired file name in *SAVE_LANDMARK_TRACKING_RESULTS_NAME*.

Most figures have been produced by executing Function 18 using the demonstration file named *DEMO_face_detection.py*.

Figure 30 demonstrates the implementation of the cubic landmark tracking feature of Functions 18 into a locally stored file downloaded from Youtube. This serves as a proof of the concept that Function 18 handles real time streams and video files the same way and hence functions developed in the demo file using a live recording maybe implemented to the *Tool* without any changes or adjustments.

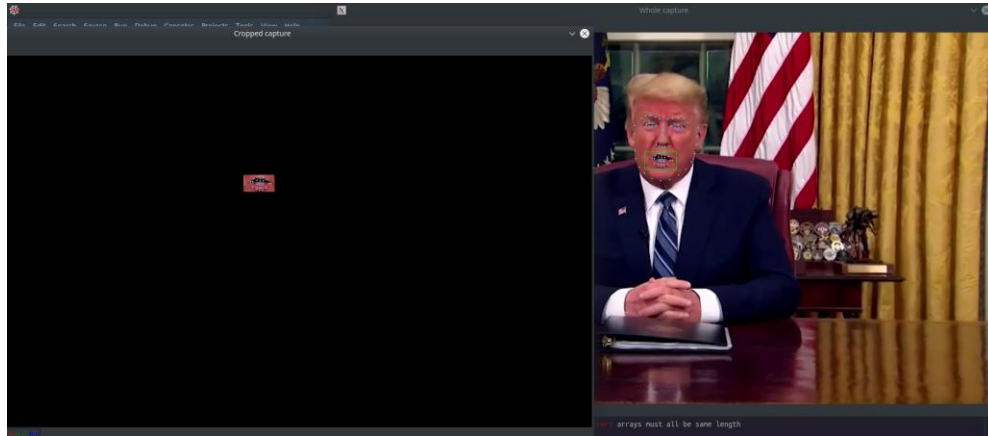


Figure 30 Implementation of Function 18 cubic landmark tracking on a video with the URL: https://www.youtube.com/watch?v=6cXdS_qVfUc.

7.4.19 End Results

When all of the *Tool's* procedures are executed in the following order the end results are:

1. The sliced and cropped videos per word or/and per sentence produced by Function 18 and stored in the folder 'pure_video' (Figure 20).
2. The sliced audio files per word or/and per sentence produced by Function 16 are stored in the folder 'pure_audio' (Figure 17).
3. The Variable 4, `dir_name_dictionary`, was created which is shown in Figure 31 contains the absolute paths of all AV files.
4. The Variable 5 `Cropped_chunk_info_dict` shown in Figure 33 which contains all of the useful information per samples such as landmark position, fps, ratio of faces detected per frame etc. was created.

Due to the size and complexity of the program the useful information about the dataset was gathered and stored in Variables 4 and 5 which have been created to help the user to find easily the useful information they need.

7.4.19.1 Variable with All File Paths

Variable 4 Variable `dir_name_dictionary`.

```
dir_name_dictionary = {'whole_files_name_dir' : whole_files_name_dir, 'chopped_files_name_dir' : chopped_files_name_dir, 'chopped_pure_audio_files_name_dir' : chopped_pure_audio_files_name_dir, 'chopped_pure_video_files_name_dir' : chopped_pure_video_files_name_dir, 'chopped_cropped_pure_video_files_name_dir' : chopped_cropped_pure_video_files_name_dir}
```

Variable 4 was created with the purpose grouping all of the filenames' absolute paths into a single variable. Figure 31 shows the first layer of the dictionary variable, the key array contains the names assigned to each row. The key-names in Figure 31 are self-explanatory, further descriptions along with the functions which created them are listed in Table 4 in the Appendix 12.9.

Key	Type	Size	Value
chopped_cropped_pure_video_files_name_dir	dict	6	{'pure_video_cropped_word_chunk_samples_file_dir_acpwe': ['/media/god/9 ...]}
chopped_files_name_dir	dict	6	{'LLIST_INPUT_WORD_CHUNK_SAMPLES_ACPWE_FILE_NAMES': ['/media/god/9c72f9 ...]}
chopped_pure_audio_files_name_dir	dict	6	{'LLIST_OUTPUT_AUDIO_WORD_CHOPPED_ACPWE_NAMES': ['/media/god/9c72f9bb-20 ...]}
chopped_pure_video_files_name_dir	dict	6	{'LLIST_OUTPUT_VIDEO_WORD_CHOPPED_ACPWE_NAMES': ['/media/god/9c72f9bb-20 ...]}
whole_files_name_dir	dict	3	{'video_converted_to_mkv': ['/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589 ...]}

Figure 31 First layer of variable 4.

Figure 32 shows an example of the variable expanded at multiple layers up to the root which contains the absolute path of a file. The second layer, as show in Figure 32, classifies the files per subtitles which are listed in Table 3 (acpwe, acpse etc.). If files do not exist for particular subtitle types such as acpse etc. they are assigned with an empty list (square brackets '[]'). The third and final layer contains the absolute paths for each segment.

Key	Type	Size	Value
chopped_cropped_pure_video_files_name_dir	dict	6	{'pure_video_cropped_word_chunk_samples_file_dir_acpwe': ['/media/god/9 ...]}
chopped_files_name_dir	dict	6	{'LLIST_INPUT_WORD_CHUNK_SAMPLES_ACPWE_FILE_NAMES': ['/media/god/9c72f9 ...]}
chopped_pure_audio_files_name_dir	dict	6	{'LLIST_OUTPUT_AUDIO_WORD_CHOPPED_ACPWE_NAMES': ['/media/god/9c72f9bb-20 ...]}
chopped_pure_video_files_name_dir	dict	6	{'LLIST_OUTPUT_VIDEO_WORD_CHOPPED_ACPWE_NAMES': ['/media/god/9c72f9bb-20 ...]}
whole_files_name_dir	dict	3	{'video_converted_to_mkv': ['/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589 ...]}

Key	Type	Size	Value
LIST_OUTPUT_AUDIO_SENTENCE_ACPSE_CHOPPED_NAMES	list	0	[]
LIST_OUTPUT_AUDIO_SENTENCE_ACPSH_CHOPPED_NAMES	list	0	[]
LIST_OUTPUT_AUDIO_SENTENCE_MCPSE_CHOPPED_NAMES	list	0	[]
LIST_OUTPUT_AUDIO_SENTENCE_MCPSH_CHOPPED_NAMES	list	0	[]
LIST_OUTPUT_AUDIO_WORD_CHOPPED_ACPWE_NAMES	list	5	['/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589cde/god/LEARNING/unlSheff/ ...']
LIST_OUTPUT_AUDIO_WORD_CHOPPED_MCPWE_NAMES	list	0	[]

Index	Type	Size	Value
0	str	1	/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589cde/god/LEARNING/unlSheff/ ...
1	str	1	/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589cde/god/LEARNING/unlSheff/ ...
2	str	1	/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589cde/god/LEARNING/unlSheff/ ...
3	str	1	/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589cde/god/LEARNING/unlSheff/ ...
4	str	1	/media/god/9c72f9bb-20f1-4b7b-8a9e-01f04589cde/god/LEARNING/unlSheff/ ...

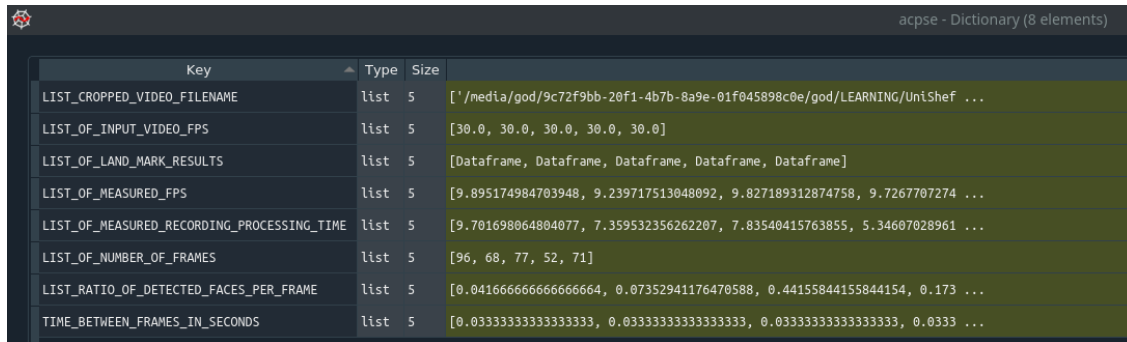
Figure 32 Contents of dir_name_dictionary variable.

7.4.19.2 Variable with All Information Per segment

Variable 5 cropped_chunk_info_dict.

```
cropped_chunk_info_dict = {'acpwe' : cropped_chunk_info_list[0], 'mcpwe': cropped_chunk_info_list[1], 'acpse':cropped_chunk_info_list[2], 'acpsh':cropped_chunk_info_list[3], 'mcpse':cropped_chunk_info_list[4], 'mcpsh':cropped_chunk_info_list[5]}
```


Variable 5 contains all the useful information for all the cropped files produced by Function 18. The first layer of the variable classifies the subtitles per type such as acpwe, mcpwe etc. If a particular subtitle type does not exist, similar to Variable 4 it is assigned square empty brackets. The second layer shown in Figure 33 contains all the useful information about the segments of the subtitle type selected.

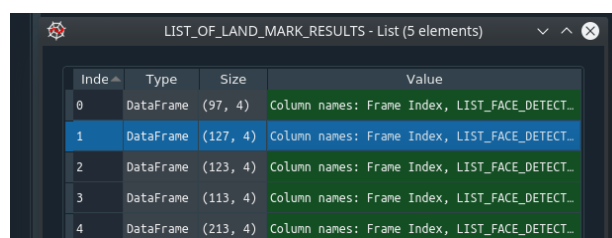


Key	Type	Size	Value
LIST_CROPPED_VIDEO_FILENAME	list	5	['/media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/UniShef ...
LIST_OF_INPUT_VIDEO_FPS	list	5	[30.0, 30.0, 30.0, 30.0, 30.0]
LIST_OF_LAND_MARK_RESULTS	list	5	[Dataframe, Dataframe, Dataframe, Dataframe, Dataframe]
LIST_OF_MEASURED_FPS	list	5	[9.895174984703948, 9.239717513048092, 9.827189312874758, 9.7267707274 ...
LIST_OF_MEASURED_RECORDING_PROCESSING_TIME	list	5	[9.701698064804077, 7.359532356262207, 7.83540415763855, 5.34607028961 ...
LIST_OF_NUMBER_OF_FRAMES	list	5	[96, 68, 77, 52, 71]
LIST_RATIO_OF_DETECTED_FACES_PER_FRAME	list	5	[0.041666666666666664, 0.07352941176470588, 0.44155844155844154, 0.173 ...
TIME_BETWEEN_FRAMES_IN_SECONDS	list	5	[0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.0333 ...

Figure 33 cropped_chunk_info_dict.

Each key consists of a list of 5 values since 5 segmented files have been inputted (in this example) into Function 18 and the output is 5 cropped segments. If a greater number of files are processed the lists will simply expand further. The order of each piece of information contained in the lists which is shown in Figure 33 corresponds to the order of the processed samples from Function 18. For instance, with reference to Figure 33, the first sample file contains 96 frames, the second 68 and etc.

The first key LIST_CROPPED_VIDEO_FILENAME contains the absolute paths of each file produced by Function 18 (the files show in Figure 20). The same information is contained in Variable 4. The second key LIST_OF_INPUT_VIDEO_FPS contains the fps of the inputted videos. The program identifies the fps of the input video and uses the result to 1) re-encode the video to the same frame rate and 2) to find the time between frames and assign it to the eighth key TIME_BETWEEN_FRAMES_IN_SECONDS. When a recording is made through the use of a local camera (live recording) the value assigned to the second key is based on the specifications of the camera. However, this value should not be used as it is inaccurate for live recordings due to the many processes (face recognition, cropping etc.) of Function 18 which delay the capture of the frames. When local recordings take place, the fourth key



Index	Type	Size	Value
0	DataFrame	(97, 4)	Column names: Frame Index, LIST_FACE_DETECT...
1	DataFrame	(127, 4)	Column names: Frame Index, LIST_FACE_DETECT...
2	DataFrame	(123, 4)	Column names: Frame Index, LIST_FACE_DETECT...
3	DataFrame	(113, 4)	Column names: Frame Index, LIST_FACE_DETECT...
4	DataFrame	(213, 4)	Column names: Frame Index, LIST_FACE_DETECT...

Figure 34 LIST_OF_LAND_MARK_RESULTS contents.

named LIST_OF_MEASURED_FPS should be used instead, which contains the actual fps of the video. The value in LIST_OF_MEASURED_FPS is calculated by measuring the number of frames stored in the sixth key LIST_OF_NUMBER_OF_FRAMES and dividing that result with the fifth key LIST_OF_MEASURED_RECORDING_PROCESSING_TIME ($FPS = \frac{frames}{seconds}$). Hence, by measuring the recording time and frames contained in a video the FPS can be found. The seventh key LIST_RATIO_OF_DETECTED_FACES_PER_FRAME contains the ration between how may frames a face is detected divided by the total number of frames in the sample, $\frac{frames\ a\ face\ is\ detected}{total\ number\ of\ frames}$. The third key LIST_LAND_MARK_RESULTS contains dataframes for each sample as shown on Figure 34. Figure 35 shows a landmark dataframe of an example sample.

Index	Frame Index	LIST_FACE_DETECTION_RESULT_PER_FRAME	X-Y Land Mark Coordinates	LAND_MARK_TIME_LENGTH_ARRAY
5	5	False	None	0.166833
6	6	False	None	0.2002
7	7	False	None	0.233567
8	8	False	None	0.266933
9	9	False	None	0.3003
10	10	False	None	0.333667
11	11	False	None	0.367033
12	12	False	None	0.4004
13	13	False	None	0.433767
14	14	True	[[293, 290, 288, 289, 294, ...	0.467133
15	15	True	[[292, 290, 289, 290, 294, ...	0.5005
16	16	True	[[291, 289, 288, 290, 296, ...	0.533867
17	17	True	[[291, 289, 288, 290, 297, ...	0.567233
18	18	True	[[293, 291, 291, 292, 297, ...	0.6006
19	19	True	[[293, 291, 290, 291, 296, ...	0.633967

Figure 35 An example segment landmark dataframe.

The first column of the dataframe, names 'Frame Index', contains the index (numerical order) of each frame in the sample and each row contains the information about the frame.

The second column, 'LIST_FACE_DETECTION_RESULT_PER_FRAME', contains the result of the face recognition algorithm for every frame. If a face is identified the True value is assigned and if not, the False value is assigned. The values assigned to the first and second columns are used to compute the LIST_RATIO_OF_DETECTED_FACES_PER_FRAME values (Figure 33). The results are important as they could be used in the future as a form of criterion (threshold) to

automatically remove any of the samples which contain a threshold lower than the expected value, $\frac{\text{frames a face is detected}}{\text{total number of frames}} < \text{threshold value}$.

The third column 'X-Y Land Mark Coordinates' contains the x and y position of each landmark per frame. Figure 36 shows the positions of the x and y coordinates per landmark when only 3 landmark points are extracted by Function 18.

Frame Index	CTION_RES	X-Y Land Mark Coordinates
0	True	[[357, 357, 358], [311, 331, 353]]
1	True	[[356, 356, 357], [311, 333, 355]]
2	True	[[357, 357, 358], [312, 333, 355]]
3	True	[[357, 358, 358], [312, 334, 356]]
4	True	[[359, 359, 360], [311, 332, 355]]

Figure 36 Notation of landmark coordinate for three loaded landmarks.

The fourth column named

'LAND_MARK_TIME_LENGTH_ARRAY' contains the time stamps of each frame/landmark (position) in the video. Hence, with the information of the landmark positions and their time values the *temporal signal* (rate of change of x or/and y against time) can be extracted. This signal can be used similarly to an audio signal to train and test ML models for lip-reading. In addition, the segmented and cropped files (Figure 20) can also be used to train and test lip-reading ML models.

7.4.19.3 Save End Results

Function 19 Save cropped_chunk_info_dict variable.

```
module_save_variables.save_pandas_dict_results(VAR_INPUT = cropped_chunk_info_dict, FILE_NAME = FOLDER_PATH + '/cropped_chunks_info_dict', CSV=True, TXT=True)
```

Function 16 is the final function which is being executed. The function attempts to save all the information which is contained in the variable `cropped_chunk_info_dict` locally and thus be able to access the results even after the program has terminated. Unfortunately, due to the structure of the variable which consists of multiple dictionaries and dataframes (as shown in Figures 33, 34 and 35), the stored format of the variable is difficult to read. In future development, the information should be added into an SQL dataset in which the format of the variable can be preserved. Function 19 supports two save formats, as a form of a csv file or as a text file. Saving the results as a csv file will preserve more information and the structured format than as a text file.

8 TESTING

The software has gone through essential functionality tests to ensure it is compatible with most videos on YouTube. The *Tool* has been tested on 25 different YouTube videos from different channels (Appendix 12.5). The videos covered a range of different lengths, frame rates and resolutions in order to ensure compatibility with various different settings. The *Tool* was capable of automatically downloading, extracting and modifying the content successfully for 20 of them. The remaining 5 videos either required changes in the default settings or failed due to errors in the program which should be patched in future versions of the software. Most errors suggest that the youtube-dl program was unable to download the video based on the default file format code specified (Function 3) as it is not supported by the particular videos. For instance the default settings requests from the server a video with a resolution of 1080p when the maximum available resolution is only 720p etc.

The slicing of videos was tested for a range of different times across each video including their beginning and end. Through initial experimentation it was found that FFmpeg was inaccurate at slicing the video with millisecond accuracy which is a vital property. According to the FFmpeg documentation (45), if the name argument '-i' (Function 20) appears first, FFmpeg will slice the video at the closest *key-frame* (whole image) to the specified time value. In media files such as mkv, mp4 etc. compression is applied in order to reduce the files size. Due to compression not all frames are *key-frames*, hence to ensure the segmented video produced has not been corrupted, FFmpeg slices only at the *key-frames*. To enforce slicing at specific time boundaries, the start time argument '-ss' in the command must appear first before the name argument '-i' (Function 5).

Function 20 FFmpeg example.

```
Slice at key frames only: ffmpeg -i file_name -ss start_time -to stop_time  
-c copy file_name  
Force slicing at specific time: ffmpeg -ss start_time -i file_name -to  
stop_time -c copy file_name
```

Figure 37 demonstrates the result of enforcing slicing at a *non-key-frame*. In order to overcome the problem of blurry images and non-working files while ensuring millisecond accuracy the files were converted into raw video format (y4m) (46). Raw video is a lossless format as it removes the compression and makes each frame in the video a key-frame (46). This resulted into both accurate slicing and non-corrupted files at the expense of producing very large files (46).

Through experimentation it was found that when the downloaded files of mp4 format were copied into an mkv format they did not result into corrupted files as shown on Figures 37 and 38 and the files' size was preserved. As a result the default value was set to convert files into mkv format. However, the developed Function 6 allows the *Tool* to convert from any video format to any other format which is supported by FFmpeg. This gives the power to the user to configure their dataset on their preference and in the occurrence of slicing errors the user could easily switch back to the safe raw video format.

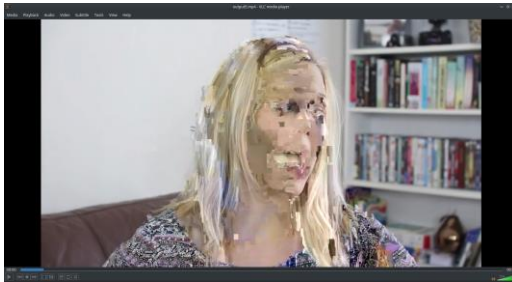


Figure 37 Sliced from mp4 format.

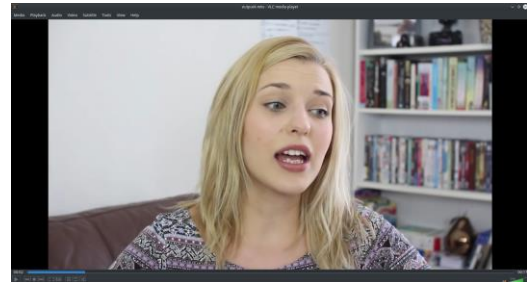


Figure 38 Sliced from mkv format.

The *Tool* was tested for all possible subtitle types available on YouTube such as manually added from the owner, automatically generated, subtitles per word or per sentence. Even though some videos contain both automatically generated subtitles and subtitles added by the user, Youtube-dl downloads the lateral format regardless of which subtitle type is requested. Errors imbedded within youtube-dl can only be resolved from the maintainers of youtube-dl.

During execution of the program update messages are printed to the console which inform of the current execution stage along with warning messages and of any potential errors as shown on Figure 39. The printed messages assist in debugging the program.

```
Completed segment 0 out of 352
Completed segment 1 out of 352
Completed segment 2 out of 352
Completed segment 3 out of 352
Completed segment 4 out of 352
/bin/sh: 2: on-the-tiny-door-back-there_2.mkv: not found
/bin/sh: 2: miniseries-books_3.mkv: not found
/bin/sh: 2: over-here-on-the-side-table_4.mkv: not found
Unable to read camera feed
Saved /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_Hello-everyone\hand-welcome-back_0_cropped.avi
Warning video: /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_Hello-everyone\hand-welcome-back_0_cropped.avi does not contain any frames
Saved /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_to-the-attic-crawl-space_1_cropped.avi
Unable to read camera feed
Saved /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_We-have-been-focussing-a-lot\non-the-tiny-door-back-there_2_cropped.avi
Warning video: /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_We-have-been-focussing-a-lot\non-the-tiny-door-back-there_2_cropped.avi does not contain any frames
Unable to read camera feed
Saved /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_as-well-as-the-various\nminiseries-books_3_cropped.avi
Warning video: /media/god/9c72f9bb-20f1-4b7b-8a9e-01f045898c0e/god/LEARNING/Unisheff/Mech/4/FYP/fyp-code/useful_bit/important_files/Twf/chopped_samples_per_sentence_mcpsh/pure_video/AUTO_SEGUNET_as-well-as-the-various\nminiseries-books_3_cropped.avi does not contain any frames
Unable to read camera feed
```

Figure 39 *Tools* procedure updates and warning messages.

Whenever the face exceeds the boundaries captured by the camera alerts are printed to the console as a form of alert as shown on Figure 40. The axes the face is off is also printed (Figure 40).

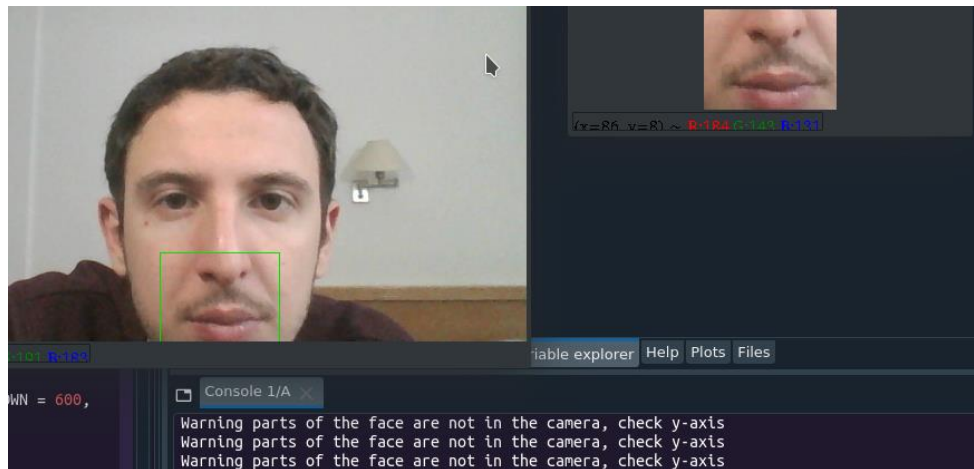


Figure 40 Exceeding the camera boundaries on the y-axis.

The *Tool* is designed to apply face recognition and extract landmark points on frames containing a single head. In many cases a video may contain frames with none or multiple faces. Figure 41 shows how the program handles this exceptions. Column 1 contains the INDEX of the list, column 2 specifies each frame, column 3 whether or not a frame is identified on that particular frame, column 4 the extracted facial landmark points and column 5 the time each frame is displayed in the segment.

86	86	False	None	2.90067
87	87	False	None	2.9
88	88	False	None	2.93333
89	89	False	None	2.96667
90	90	False	None	3
91	91	True	[[774, 773, 773, 774, 774, 777, 780, 785, 792, 802, 811, 819, ...	3.03333
92	92	True	[[775, 774, 774, 774, 775, 778, 781, 786, 794, 803, 813, 820, ...	3.06667
93	93	True	[[776, 775, 775, 775, 776, 778, 782, 787, 795, 804, 813, 820, ...	3.1
94	94	2 faces_found	[[777, 776, 776, 776, 777, 779, 783, 787, 795, 804, 814, 821, ...	3.13333
95	95	True	[[602, 603, 604, 607, 613, 620, 628, 637, 645, 650, 653, 655, ...	3.16667
96	nan	None	[[778, 777, 777, 777, 777, 780, 783, 788, 796, 806, 815, 822, ...	3.2

Figure 41 Face recognition results per frame.

If a face is not detected in the frame, the corresponding cell in column 3 (Figure 41) is assigned with False and on column 4 with None. If a single face is detected the True Boolean value is assigned to the cell of column 3 and the x-y landmark positions are assigned to column 4. If more than one face is detected in a

frame, the number of faces detected will be shown in column 4 as shown on index 94. However, when multiple faces are detected, the landmark predictor extracts facial landmarks from all of the faces. Due to this reason facial landmarks occupy two cells in column 4 as shown on Figure 41 (rows 94 and 95 in the particular case) causing its size to be enlarged by one additional value. As a result, all values of column 4 are shifted downwards from the point in which multiple faces are found and onwards (Figure 41). This causes column 4 to be over extend. As a result, the row of index 96 is assigned with landmarks while at the same time it does not contain a frame according to column 2 since it is assigned the 'nan' value. In the particular case, due to the shifting, the landmark coordinates of index 96 correspond (in the particular case) to the previous frame of index 95. The same behaviour holds applies for any number of identified faces above one. Currently the *Tool* does not support any means of identifying which landmarks correspond to each person and hence segments which contain multiple faces should be removed. In order to support this process, apart from specifying the number of detected faces in column 3, if more than a single faces is detected a warning message with the frame and number of faces detected will be printed to the console as shown on Figure 42.

42	42	3 faces_found	[[237, 236, 236, 236, 237, 240, 244, 250, 258, 266, 274, 281, ... 1.4
43	43	3 faces_found	[[696, 696, 698, 700, 706, 714, 722, 730, 737, 743, 746, 748, ... 1.43333
44	44	3 faces_found	[[237, 236, 236, 236, 237, 239, 243, 249, 256, 265, 273, 280, ... 1.46667
45	45	3 faces_found	[[697, 698, 700, 703, 708, 715, 722, 730, 738, 744, 746, 749, ... 1.5
46	46	3 faces_found	[[238, 237, 236, 236, 236, 238, 243, 248, 256, 264, 273, 280, ... 1.53333
47	47	3 faces_found	[[699, 700, 701, 704, 709, 716, 724, 732, 740, 745, 748, 751, ... 1.56667
48	48	3 faces_found	[[238, 237, 236, 236, 236, 239, 243, 248, 256, 265, 273, 280, ... 1.6
49	49	3 faces_found	[[699, 700, 701, 704, 709, 716, 724, 731, 739, 745, 748, 751, ... 1.63333

Format Resize Background color Column min/max Save and Close Close

```

sample. The number of faces detected on frame with index FRAME_INDEX = 40 is len(faces) = 3
Warning, len(LIST_LANDMARK_RESULT_PER_INPUT) is neither equal or smaller by 1. This could be due to the fact that multiple faces are found in at least 1 f
sample. The number of faces detected on frame with index FRAME_INDEX = 41 is len(faces) = 3
Warning, len(LIST_LANDMARK_RESULT_PER_INPUT) is neither equal or smaller by 1. This could be due to the fact that multiple faces are found in at least 1 f
sample. The number of faces detected on frame with index FRAME_INDEX = 42 is len(faces) = 3
Warning, len(LIST_LANDMARK_RESULT_PER_INPUT) is neither equal or smaller by 1. This could be due to the fact that multiple faces are found in at least 1 f
sample. The number of faces detected on frame with index FRAME_INDEX = 43 is len(faces) = 3
Warning, len(LIST_LANDMARK_RESULT_PER_INPUT) is neither equal or smaller by 1. This could be due to the fact that multiple faces are found in at least 1 f
sample. The number of faces detected on frame with index FRAME_INDEX = 44 is len(faces) = 3

```

Figure 42 Warning message of multiple faces found.

9 FUTURE IMPROVEMENTS

The range of additional features and potential improvements are endless. However, the remaining features which have been implemented by other *pipelines* should be prioritised. These features are alignment of audio signal to text with the use of P2FA and identification of the speaking head with the use of SyncNet.

Due to time limitations many features have been excluded from the program. The following list contains recommended features and improvements for future development:

- Include data augmentation functions which will allow the enlargement of the dataset by adding additional edited versions of the same files. An example would be to add modified duplicates with a reduced amount of brightness.
- Improve time performance and reduce the complexity of the code.
- Implementation of automatic transcript validation and automatic subtitle generation by implementing pre trained voice recognition algorithms such as DeepSpeech (Mozilla's ASR model) on the audio samples (47).
- Develop a MySQL dataset which will include all necessary information for each file into a modular and easy to read format for multiple URLs. Such information includes size, total number of files, directory of each file, number of occurrences of each word, position of each word in the sentence and more. Essential functionalities such as automatic updating, version control and recovery of lost files should be included.
- Include additional options to merge sequential words or sentences together. There for allow more slicing options such as chop for every two words instead of each word.
- Add a GUI to make the program more approachable and user-friendly.
- Implement more sophisticated face recognition model and a shape of a greater number of landmark points.

10 LICENSE

Among the many varieties of open-source licences which have been approved by the *open source initiative (OSI)*. The code of the project is licensed under the MIT licence and is hosted on GitHub account 'openmindednewby' in the repository 'Tool-for-creating-lip-recognition-datasets'.

The MIT licence is one of the most popular licences in the open source community. It allows freedom to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software without any warranty or liability to the owner for damages.

11 SUMMARY

The work conducted resulted to the creation of a non GUI program (*Tool*) capable of automatically downloading and processing AV content from YouTube. Many features have been developed such as:

- Identifying the number of heads in a frame.
- Extracting a desired number of facial land mark points.
- Automatically cropping desired parts of the face such as the lips.
- Splitting of the video into chunks per word or per sentence.
- Shifting and extending slicing boundaries.
- Handling of various subtitle formats.

The *Tool* has been developed sufficiently to be capable of developing a dataset suitable for training and testing DNN. As a result the objective of the project has been met. However, the lack of implementing alignment between AV content to the text (P2FA) and speaker identification (SyncNet) make the current state of the program inferior in comparison to the *pipelines* used to create LRW, LRS2 and LRS3. The exclusion of these features was based on the design approach of the project which had ultimately led to the achievement of a viable program. Future work should focus on implementing alignment and speaker identification features.

The functional structure of the *Tool* allow it the download and processing of any YouTube video as well as AV content stored locally. YouTube videos could be downloaded in a range of resolution qualities up to 4k and frame rates up to 60fps. Hence, by implementing the lacking features of other *pipelines* (P2FA and SyncNet) the *Tool* has the potential to build more divers, larger and higher quality datasets than the current state of the art LRW, LRW2 and LRS3.

The *Tool's* function to download online content should be used with caution and preferably only on non-copyrighted material due to the variations of copyright legislations between countries. The tool could also be used on any AV content stored locally as it was designed to process local media files. Hence, downloading YouTube content may be replaced with local recordings. Additionally, the *Tools* also produces audio files per word or sentence which could be used to create an ASR dataset.

The study did not carry out any local recordings despite the development of features to protect the privacy of participants.

The source code of the *Tools* is made available for free under the MIT licence and it is made available through the GitHub in the Appendix 12.4.

12 APPENDIX 1.

12.1 MIT Licence

Copyright 2020 Demetrios Loizides

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

12.2 Future Software Architecture

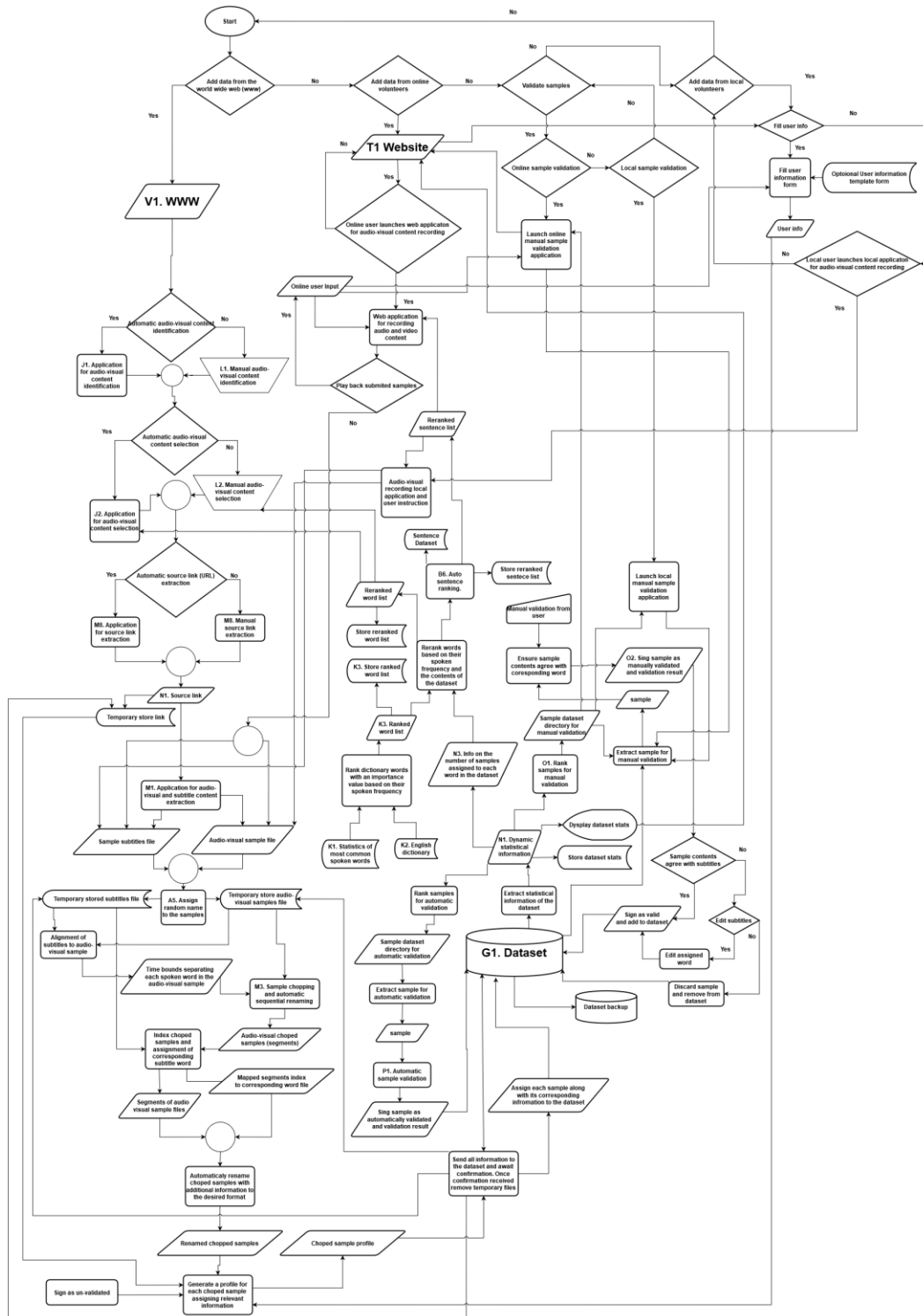
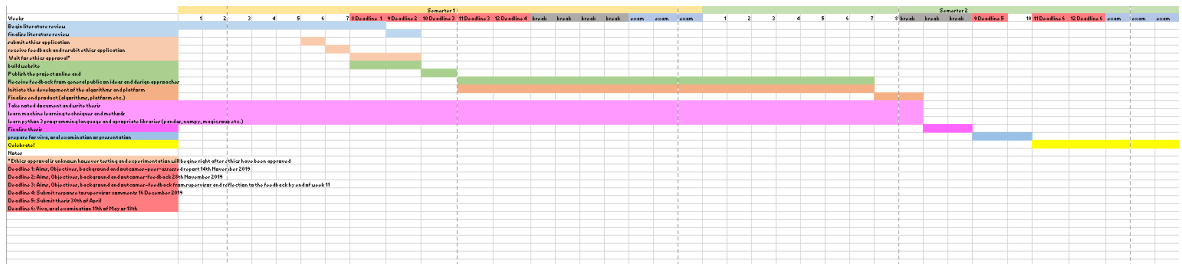


Figure 43 Pipeline which incorporates all methods considered.

12.3 Projects Gant chart



12.4 Link to Source Code

The latest version of the software can be downloaded from the link '<https://github.com/openmindednewby/Tool-for-creating-lip-recognition-datasets>' or by searching the GitHub repository 'openmindednewby / Tool-for-creating-lip-recognition-datasets'.

12.5 List of Tested URLs

URLs which work without editing the default settings apart from INPUT_URL and NNAME variables.

1. https://www.youtube.com/watch?v=6cXdS_qVfUc
2. <https://www.youtube.com/watch?v=v2Q3eoUldcE>
3. <https://www.youtube.com/watch?v=kayOhGRcNt4>
4. https://www.youtube.com/watch?v=YHCZt8LeQzI&fbclid=IwAR2e436VcxEBWWnnz48W2vPU4iTfFpxggIA9U7uIOFP1XCA1sdp4h_qnmLI
5. <https://www.youtube.com/watch?v=a1KxhhmqT8U>
6. <https://www.youtube.com/watch?v=dRFbwjwQ4VE>
7. https://www.youtube.com/watch?v=PpV_5-tCS-c
8. <https://www.youtube.com/watch?v=DhYeqgufYss>
9. <https://www.youtube.com/watch?v=HqI0jbKGaT8&pbjreload=10>
10. <https://www.youtube.com/watch?v=kR-WCDa4NSc>
11. <https://www.youtube.com/watch?v=PjQ-AfRNG18>
12. <https://www.youtube.com/watch?v=PACH0XKozuU>
13. <https://www.youtube.com/watch?v=ie6lRKAdvuY>
14. <https://www.youtube.com/watch?v=w2PQEzDawMw>
15. <https://www.youtube.com/watch?v=5v-wyR5emRw>
16. <https://www.youtube.com/watch?v=MmFuWmzeiDs>
17. <https://www.youtube.com/watch?v=3obig1XeOlw>
18. <https://www.youtube.com/watch?v=Xdzo2dVqNH0>
19. <https://www.youtube.com/watch?v=m8ZUvBeKZEY>
20. <https://www.youtube.com/watch?v=uiU5GutVms4>

URLs which either require changes to the default settings or improvements to the code.

1. <https://www.youtube.com/watch?v=ZTK8XJUXqy8>
2. <https://www.youtube.com/watch?v=1mHjMNZZvFo>

3. <https://www.youtube.com/watch?v=ZO44B271tfk>
4. https://www.youtube.com/watch?v=z0hrMg1j_d4
5. <https://www.youtube.com/watch?v=aeT3YOYsvMs>

12.6 Ethical Approval Form

The ethical approval form which was produced through the university of Sheffield online platform can be found here https://ethics.ris.shef.ac.uk/ethics_applications/30339.pdf.

12.7 Link to FOURCC Codecs

The following link contains all the available FOURCC codecs which could be used in Function 18 <http://www.fourcc.org/codecs.php>

12.8 Demo File Notice

While recording live streams in the file DEMO_face_detect.py, closing the window which displays what is been recorded does not terminate the application as the window reopens immediately. In order to terminate the execution appropriately the display window must be selected and closed by pressing the 'Q' key stroke. If the display window is terminated through alternative methods such as with the use of a keyboard interruption Ctrl+C, this could cause the execution of python to terminate without terminating the internal process used to handle the camera. This will cause the program to fail upon its next execution as the camera is still been reserved by the previous process. In order to overcome this problem and release the camera, the python kernel should be restarted.

12.9 Variable 4 Layer 1 Key-Names Descriptions

Table 4 Description of Variable 4 contents.

File Name	Description
chopped_cropped_pure_video_files_name_dir	Absolute paths of the end result files produced by Function 18 and shown on Figure 20.

chopped_files_name_dir	Absolute paths of the sliced video files produced by Function 14 and shown on Figure 14.
chopped__pure_audio_files_name_dir	Absolute paths of the sliced audio files produced by Function 16 and shown on Figure 17.
chopped_pure_video_files_name_dir	Absolute paths of the sliced audio files produced by Function 17 and shown on Figure 18.
whole_files_name_dir	Absolute paths of the whole files downloaded by Function 4 and shown on Variable 4.

13 REFERENCES

1. Automatic Speech Recognition (ASR) . [Online] technopedia, 2019. [Cited: 09 12 2019.] <https://www.techopedia.com/definition/6044/automatic-speech-recognition-asr>.
2. Zisserman, Joon Son Chung and Andrew. The Oxford-BBC Lip Reading in the Wild (LRW) Dataset. [Online] 2016. [Cited: 22 01 2020.] <https://www.robots.ox.ac.uk/~vgg/publications/2016/Chung16/chung16.pdf>.
3. Yannis M. Assael, Brendan Shillingford], Shimon Whiteson, Nando de Freitas. YouTube. [Online] 06 January 2017. [Cited: 09 12 2019.] <https://www.youtube.com/watch?v=YTkqA189pzQ&list=PLXkuFIFnXUAPlrXKgtlpctv2NuSo7xw3k&index=2>.
4. Yannis Assael, Brendan Shillingford, Prof Shimon Whiteson and Prof Nando de Freitas. LipNet: How easy do you think lipreading is? Youtube. [Online] 04 November 2016. [Cited: 05 12 2019.]

<https://www.youtube.com/watch?v=fa5QGremQf8&list=PLXkuFIFnXUAPlrXKgtlpctv2NuSo7xw3k&index=1>.

5. Joon Son Chung, Andrew Senior, Oriol Vinyals, Andrew Zisserman. *Lip Reading Sentences in the Wild*. Department of Engineering Science, University of Oxford DeepMind. 2017.

6. Balog, Trond Linjordet and Krisztian. *Impact of Training Dataset Size on Neural Answer Selection Models*. University of Stavanger, Stavanger, Norw. 2019.

7. Lip Reading Datasets LRW, LRS2, LRS3. [Online] [Cited: 19 04 2020.] http://www.robots.ox.ac.uk/~vgg/data/lip_reading/.

8. Lip Reading in the Wild and Lip Reading Sentences in the Wild Datasets . [Online] BBC Reaserch & Development, 2020. [Cited: 19 04 2020.] <https://www.bbc.co.uk/rd/projects/lip-reading-datasets>.

9. Yannis M. Assael, Brendan Shillingford, Shimon Whiteson, Nando de Freitas. LipNet: End-to-End Sentence-level Lipreading. [Online] 05 November 2016. [Cited: 2019 12 05.] <https://arxiv.org/pdf/1611.01599.pdf>.

10. Paul Duchnowski, Uwe Meier and Alex Waibel. <https://www.isca-speech.org/iscaweb/index.php/archive/online-archive>. [Online] 22 September 1994. [Cited: 02 12 2019.] <https://pdfs.semanticscholar.org/16ce/d6a1e6ee2599ce21c2601334cc17a0bf102f.pdf>.

11. *Lipreading with long short-term memory*. M. Wand, J. Koutnik and J. Schmidhuber. 2016, IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6115-6119.

12. S. Gergen, S. Zeiler, A. H. Abdelaziz, R. Nickel and D. Kolossa. Dynamic stream weighting for turbo-decoding-based audiovisual ASR. [Online] 2016. [Cited: 09 12 2019.] <https://pdfs.semanticscholar.org/3087/e40e076b2a7bbb1e7a6efb7779a941283853.pdf>.

13. Vincent, James. Can deep learning help solve lip reading? [Online] THE VERGE, 07 November 2016. [Cited: 05 12 2019.] <https://www.theverge.com/2016/11/7/13551210/ai-deep-learning-lip-reading-accuracy-oxford>.

14. Joon Son Chung, Andrew Senior, Oriol Vinyals, Andrew Zisserman. *Lip Reading Sentences in the Wild*. Department of Engineering Science, University of Oxford. Oxford : DeepMind, 2017.
15. Barker, Martin Cooke and Jon. reaserchgate. [Online] University of Sheffield, 02 April 2006. https://www.researchgate.net/publication/6658688_An_audio-visual_corpus_for_speech_perception_and_automatic_speech_recognition_L/link/5e329ff5458515072d6e8194/download.
16. Barker, Martin Cookea and Jon. An audio-visual corpus for speech perception and automaticspeech recognition (L). [Online] 02 April 2006. [Cited: 01 03 202.] <https://staffwww.dcs.shef.ac.uk/people/J.Barker/assets/cooke-2006-jasa-ecbf8f7ef7cb429e9621317bfc64a67002a4c465be3c1a3f6144eed058ee634.pdf>.
17. Najwa Alghamdi, Steve Maddock, Jon Barker, Ricard Marxer, And Guy Brown. The Audio-Visual Lombard Grid Speech corpus. [Online] The University of Sheffield, 2017. <http://spandh.dcs.shef.ac.uk/avlombard/>.
18. Common Voice. [Online] mozilla, 2019. [Cited: 17 12 2019.] <https://voice.mozilla.org/en/datasets#get-started>.
19. Common Voice. [Online] mozilla, 01 05 2020. <https://voice.mozilla.org/en/datasets>.
20. Reki, Ahmed. Achraf Ben-Hamadou. [Online] Google Sites. [Cited: 17 12 2019.] <https://sites.google.com/site/achrafbenhamadou/-datasets/miracl-vc1>.
21. Chalapathy Neti, Gerasimos Potamianos, Juergen Luetlin, Iain Matthews, Herve Glotin, Dimitra Vergyri, June Sison, Azad Mashari and Jie Zhou. AUDIO-VISUAL SPEECH RECOGNITION. [Online] 12 October 2000. [Cited: 09 12 2019.] <http://www.cs.cmu.edu/~iainm/papers/ws00avsr.pdf>.
22. Triantafyllos Afouras, Joon Son Chung, Andrew Zisserman. *LRS3-TED: a large-scale dataset for visual speech recognit*. Visual Geometry Group, Department of Engineering Science, University of Oxford, UK. 2018.

23. *Deep Audio-Visual Speech Recognition (LRS2)*. Triantafyllos Afouras, Joon Son Chung, Andrew Senior, Oriol Vinyals, Andrew Senior. 2018.
24. Zisserman, Joon Son Chung and Andrew. *Out of time: automated lip sync in the wild*. Visual Geometry Group, Department of Engineering Science, University of Oxford. Oxford : s.n., 2016.
25. Wei Liu¹, Dragomir Anguelov², Dumitru Erhan³, Christian Szegedy³, Scott Reed⁴, Cheng-Yang Fu¹, Alexander C. Berg. SSD: Single Shot MultiBox Detector. [Online] 22 Dec 2016. [Cited: 01 03 2020.] <https://arxiv.org/pdf/1512.02325.pdf>.
26. Fair Use. [Online] YouTube. [Cited: 12 03 2020.] <https://support.google.com/youtube/answer/6396261>.
27. Measuring Fair Use: The Four Factors. [Online] Stanford University Libraries, 2020. [Cited: 12 03 2020.] <https://fairuse.stanford.edu/overview/fair-use/four-factors/>.
28. Exceptions to copyright. [Online] gov.uk, 12 June 2019. [Cited: 12 03 2020.] <https://www.gov.uk/guidance/exceptions-to-copyright#text-and-data-mining-for-non-commercial-research>.
29. YouTube Help. [Online] YouTube. [Cited: 12 03 2020.] https://support.google.com/youtube/answer/2797449?hl=en&ref_topic=2778546.
30. McCoy, Julia. The 10 Best Image Search Engines. [Online] SEJ, 08 April 2019. <https://www.searchenginejournal.com/best-image-search-engines/299963/>.
31. Bypassing Android's Smart Lock using a Photo. [Online] F-Secure, 15 Nov 2017. <https://www.youtube.com/watch?v=CKDn1748dFI>.
32. Kniberg, Henrik. Making sense of MVP (Minimum Viable Product) – and why I prefer Earliest Testable/Usable/Lovable. [Online] Crisp's Blog, 25 01 2016. [Cited: 03 03 2020.] <https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp>.
33. Nanouks, Paul. Understanding the Linux mindset. *Linux.com*. [Online] The Linux Foundation, 15 February 2010. [Cited: 04 10 2019.] <https://www.linux.com/tutorials/understanding-linux-mindset/>.

34. Dell Manual. [Online] [Cited: 22 04 2020.]
https://downloads.dell.com/manuals/all-products/esuprt_laptop/esuprt_inspiron_laptop/inspiron-15-7559-laptop_reference%20guide_en-us.pdf.
35. *The Influence of Video Sampling Rate on Lipreading Performance*. Rothkrantz, Alin G. Chińu and Leon J.M. Man-Machine Interaction Group Delft University of Technology, Delft, The Netherlands : s.n.
36. Salus, Peter H. A Quarter Century of Unix. [Online] 1994. [Cited: 14 10 2019.]
<https://wiki.tuhs.org/lib/exe/fetch.php?media=publications:qcu.pdf>.
37. *The Bell System Technical Journal*. Douglas, McIlroy. American Telephone and Tele-graph Company, J. D. deBu, 600 Mountain Ave New Providence, NJ 07974 USA : American Telephone and Tele-graph Company, J. D. deBu, 1978, Vols. 57, NO. 6, PART 2. http://emulator.pdp-11.org.ru/misc/1978.07_-_Bell_System_Technical_Journal.pdf.
38. Plauger, Brian W. Kernighan and P.J. THE ELEMENTS OF PROGRAMMING STYLE. [Online] 1978. [Cited: 14 10 2019.]
http://www2.ing.unipi.it/~a009435/issw/extra/kp_elems_of_pgmngsty.pdf.
39. Raymond, Eric Steven. The Art of Unix Programming. *The Art of Unix Programming*. [Online] 2003. [Cited: 14 10 2019.]
https://homepage.cs.uri.edu/~thenry/resources/unix_art/.
40. Gonzalez, Ricardo Garcia. youtube-dl Supported Sites. [Online] youtube-dl developers , 2020. [Cited: 20 04 2020.] <https://ytdl-org.github.io/youtube-dl/supportedsites.html>.
41. mkdir(1) - Linux man page. [Online]
<https://linux.die.net/man/1/mkdir>.
42. Penn Phonetics Lab Forced Aligner Toolkit (P2FA). [Online] University of Pennsylvania Department of Linguistics, 2008.
https://babel.ling.upenn.edu/phonetics/old_website_2015/p2fa/readme.txt.
43. *Facial point annotations*. Department of Computing, Imperial College London, Intelligent Behaviour Understanding Group (iBUG).
44. AKSHAYUBHAT. *shape_predictor_68_face_landmarks.dat*. 2015.

45. Seeking. [Online] FFmpeg, 2018.
<https://trac.ffmpeg.org/wiki/Seeking>.
46. Stoyanov, George. Encode Raw Video. [Online] GitHub, 18 Jun 2018 .
[Cited: 21 04 2020.]
<https://github.com/stoyanovgeorge/ffmpeg/wiki/Encode-Raw-Video>.
47. mozilla. DeepSpeech. *GitHub*. [Online] [Cited: 26 10 2019.]
<https://github.com/mozilla/DeepSpeech>.
48. *Impact Wear Testing Machine*. Bayer, R G, Engel, P A and Sirico, J L.
1972, Wear, Vol. 19, pp. 343-354.
49. Stachowiak, G W and Batchelor, A W. *Engineering Tribology*.
Amsterdam : Elsevier, 2005.
50. [Online] <https://opensource.org/licenses>.
51. Licenses & Standards. *Open Source Initiative*. [Online] [Cited: 04 10 2019.] <https://opensource.org/licenses>.
52. MAC Technology Overview. *documentation archive*. [Online] Apple Inc, 16 09 2015. [Cited: 08 10 2019.]
https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/SystemTechnology/SystemTechnology.html#//apple_ref/doc/uid/TP40001067-CH207-BCICAIFJ.
53. Dolya, Aleksey. Linux Journal. *Interview with Brian Kernighan*. [Online] Linux Journal, 29 Jul 2003. [Cited: 08 10 2019.]
<https://web.archive.org/web/20171018090033/https://www.linuxjournal.com/article/7035>.
54. McIlroy, M. Douglas. M. Douglas McIlroy Biography. *Department of Computer Science, Dartmouth College* . [Online] Trustees of Dartmouth College, May 2019. [Cited: 09 10 2019.]
<https://www.cs.dartmouth.edu/~doug/biography>.
55. MyoWare Muscle Sensor . *Advancer technologies*. [Online] Blogger, 2016. [Cited: 11 10 2019.]
<http://www.advancertechnologies.com/p/myoware.html>.
56. MyoWare Muscle Sensor. *sparkfun*. [Online] SparkFun Electronics. [Cited: 11 10 2019.] <https://www.sparkfun.com/products/13723>.
57. Kaminski, Brian E. AdvancerTechnologies/MyoWare_MuscleSensor. *github*. [Online] 26 Jul 2016. [Cited: 11 Oct 2019.]

https://github.com/AdvancerTechnologies/MyoWare_MuscleSensor/raw/master/Documents/AT-04-001.pdf.

58. Krzyzanowski, Paul. *C Programming Style*. *RUTGERS School of Arts and Sciences*. [Online] 1998. [Cited: 14 10 2019.] <https://www.cs.rutgers.edu/~pxk/rutgers/notes/content/Cstyle.pdf>.

59. George A. Mille. *The Magical Number Seven, Plus or Minus Two* Some Limits on Our Capacity for Processing Information. *Harvard University*. [Online] 15 04 1955. [Cited: 16 10 2019.] <http://www2.psych.utoronto.ca/users/peterson/psy430s2001/Miller%20GA%20Magical%20Seven%20Psych%20Review%201955.pdf>.

60. About the GNU Operating System. *GNU Operating System*. [Online] 15 12 2018. [Cited: 17 10 2019.] <https://www.gnu.org/gnu/about-gnu.html>.

61. Lockney, Dan. *Silent Speech*. *NASA TECHNOLOGY TRANSFER PROGRAM*. [Online] 2005. [Cited: 23 10 2019.] <https://ntts-prod.s3.amazonaws.com/t2p/prod/t2media/tops/pdf/TOP2-131.pdf>.

62. Braukus, Michael. *NASA News*. *nasa.gov*. [Online] 17 March 2004. [Cited: 23 10 2019.] https://www.nasa.gov/home/hqnews/2004/mar/HQ_04093_subvocal_speech.html.

63. Wiggers, Kyle. *Researchers develop offline speech recognition that's 97% accurate*. *VentureBeat*. [Online] VentureBeat, 22 October 2018. [Cited: 25 10 2019.] <https://venturebeat.com/2018/10/22/researchers-develop-offline-speech-recognition-thats-97-accurate/>.

64. Boyd, Clark. *The Startup*. *Speech Recognition Technology: The Past, Present, and Future*. [Online] 10 Jan 2018. [Cited: 25 10 2019.] <https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf>.

65. Hamilton, Andrew. *Voice Recognition Technology? In a Lift? IN SCOTLAND? DIGIT*. [Online] DIGIT. [Cited: 25 10 2019.] <https://digit.fyi/microsoft-voice-recognition-accuracy/>.

66. Kovo, Yael. *NASA TV*. *Technology Opportunity: Sub-Audible Speech Recognition Based on Electromyographic Signals*. [Online] NASA Ames Reserch Center, 7 Aug 2017. [Cited: 2019 10 24.] <https://www.nasa.gov/old1/ames->

partnerships/technology/technology-opportunity-sub-audible-speech-recognition-based-on-electromyographic-signals.

67. *EMG-Based Speech Recognition Using HiddenMarkov Models With Global Control Variables*. Lee, Ki-Seung. 3, s.l. : IEEE, 2008, Vol. 55.

68. Maier-Hein, Lena. Speech Recognition Using Surface Electromyography. [Online] Juli 2005. [Cited: 2019 10 30.] <https://www.cs.cmu.edu/~tanja/Papers/DA-MaierHein.pdf>.

69. Zwass, Vladimir. Speech recognition TECHNOLOGY. [Online] ENCYCLOPAEDIA BRITANNICA, 2019. [Cited: 31 10 2019.] <https://www.britannica.com/technology/speech-recognition>.

70. Rogers, Kara. Skeletal muscle. *BioNinja*. [Online] ENCYCLOPAEDIA BRITANNICA, 2019. [Cited: 25 10 2019.] <https://www.britannica.com/science/skeletal-muscle>.

71. —. Electromyography. [Online] ENCYCLOPAEDIA BRITANNICA, 2019. [Cited: 14 11 2019.] <https://www.britannica.com/science/electromyography>.

72. *Principal envelope model*. Jia Zhang, Xin Chen. 2019, Journal of Statistical Planning and Inference.

73. Vincent, James. Google's AI can now lip read better than humans after watching thousands of hours of TV. [Online] 24 November 2016. [Cited: 03 12 2019.] <https://www.theverge.com/2016/11/24/13740798/google-deepmind-ai-lip-reading-tv>.

74. *Lip Reading Sentences in the Wild*. Joon Son Chung, Andrew Senior, Oriol Vinyals and Andrew Zisserman. 2016, Vol. 1, pp. 1-8.

75. Barker, Jon. The GRID audiovisual sentence corpus. [Online] 18 March 2013. [Cited: 04 12 2019.] <http://spandh.dcs.shef.ac.uk/gridcorpus/>.

76. Zeb, Yasir. LipNET: A Lipreading Technology Revealed By The Researchers. [Online] rs.news, 31 August 2016. [Cited: 05 12 2019.] <https://pdfs.semanticscholar.org/291c/0e453503a704c0fd932a067ca054cc7edad6.pdf>.

77. *Confusions among visually preceived consonants*. Fisher, G. 11, 1968, Journal of Speech, Language and Hearing, Vol. 4, pp. 796-804.

78. *Perceptual dominance during lipreading*. Basala, R. D. Easton and M. 6, 1982, Perception & Psychophysics, Vol. 32, pp. 562-570.
79. *Connectionist temporal classification*. A Graves, S. Fernandez, F. Gomez and J. Schmidhuber. 2006, Labelling unsegmented sequence data with recurrent neural networks, pp. 369--376.
80. Gregersen, Erik. *ENCYCLOPAEDIA BRITANNICA - Judea Pearl* . [Online] [Cited: 13 12 2019.] <https://www.britannica.com/biography/Judea-Pearl#ref1174347>.
81. Erik Gregersen. *ENCYCLOPAEDIA BRITANNICA - Bayesian analysis* . [Online] [Cited: 13 12 2019.] <https://www.britannica.com/science/Bayesian-analysis>.
82. CC0 1.0 Universal. [Online] Creative Commons. [Cited: 17 12 2019.] <https://creativecommons.org/publicdomain/zero/1.0/legalcode>.
83. Awni Hannun*, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. N. Deep Speech: Scaling up end-to-end speech recognition. [Online] Baidu Research – Silicon Valley AI Lab, 19 December 2014. [Cited: 17 12 2019.] <https://arxiv.org/pdf/1412.5567.pdf>.
84. Dim, Foti. DeepSpeech on Windows WSL. [Online] Medium, 06 December 2017. [Cited: 18 12 2019.] <https://fotidim.com/deepspeech-on-windows-wsl-287cb27557d4>.
85. mozilla. Project DeepSpeech. [Online] GitHub, 18 12 2019. [Cited: 18 12 2019.] <https://github.com/mozilla/DeepSpeech>.
86. Livingstone, Frank A. Russo and Steven R. YouTube. [Online] Department of Psychology, Ryerson University, Toronto, Canada, Department of Computer Science and Information Systems, University of Wisconsin-River Falls, Wisconsin, WI, United States of America , 16 May 2018. [Cited: 19 12 2019.] <https://www.youtube.com/watch?v=0rvNpbucZOg>.
87. Dump, Brain. The Black Magic of Deep Learning - Tips and Tricks for the practitioner . [Online] EnVision , 21 February 2017. [Cited: 19 12 2019.] <https://nmarkou.blogspot.com/2017/02/the-black-magic-of-deep-learning-tips.html>.

88. Machine Learning Repository. [Online] University of California, Irvine. [Cited: 20 01 2020.] <https://archive.ics.uci.edu/ml/datasets.php?format=&task=&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=list>.
89. Stanford. [Online] [Cited: 22 01 2020.] <http://web.stanford.edu/class/archive/cs/cs166/cs166.1146/lectures/09/Small09.pdf>.
90. Seebibyte. [Online] University of Oxford. [Cited: 22 01 2020.] <http://www.robots.ox.ac.uk/~vgg/projects/seebibyte/index.html>.
91. Kazemi, V., Sullivan, J. One millisecond face alignment with an ensemble of regression tree. [Online] 2014. [Cited: 24 01 2020.] http://www.nada.kth.se/~sullivan/Papers/Kazemi_cvpr14.pdf.
92. Microsoft Research. [Online] Microsoft, 2018. [Cited: 24 01 2020.] <https://msropendata.com/categories>.
93. Lip Reading Sentences 3 (LRS3) Dataset. [Online] Visual Geometry Group, 2018. [Cited: 18 02 2020.] https://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrs3.html.
94. Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License. [Online] Creative Commons. [Cited: 18 02 2020.] <https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>.
95. Common Voice. [Online] Mozilla. [Cited: 20 02 2020.] <https://voice.mozilla.org/en>.
96. Rosebrock, Adrian. Facial landmarks with dlib, OpenCV, and Python. [Online] oyunagesearch, 03 April 2017. [Cited: 21 02 2020.] <https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/>.
97. dLIB C++ Library. [Online] 14 Dec 2019. [Cited: 32 02 2020.] <http://dlib.net/>.
98. Face detector. [Online] Dlib. [Cited: 21 02 2020.] http://dlib.net/face_detector.py.html.
99. Introduction to the Second Edition of the Oxford English Dictionary. [Online] Oxford University Press, 2020. [Cited: 25 02 2020.] <https://public.oed.com/history/oed-editions/introduction-to-the-second-edition/>.

100. Dictionaries, Oxford. Concise Oxford English Dictionary: 11th edition revised 2008 . [Online] [Cited: 25 02 2020.] <https://www.amazon.co.uk/Concise-Oxford-English-Dictionary-revised/dp/0199548412>.
101. *Hearing lips and seeing voices*. McGurk, H., MacDonald. 1976, Nature, Vol. 264, pp. 746-748.
102. *Confusability of phonemes grouped according to their viseme classes in noisy environments*. Lucey, P., Martin, T., Sirdharn. 2004, Proc. of Australian Int. Conf. on Speech Science & Tech, pp. 265-270.
103. *Unsupervised random forest manifold alignment for lipreading*. Pei, Y., Kim, T.K., Zha, H. 2013, Proceedings of the IEEE International Conference on Computer Vision, pp. 129-136.
104. *The 1994 htk large vocabulary speech recognition system*. In: *Acoustics, Speech, and Signal Processing*, . Woodland, P.C., Leggetter, G., Odell, J., Valtchev, V., Young, S.J. 1995 , ICASSP-95., International Conference on 1995.
105. *An analysis of incorporating an external language model into a sequence-to-sequence model*. A. Kannan, Y. Wu, P. Nguyen, T. N. Sainath, Z. Chen, and R. Prabhavalkar. 2017.
106. Andrew Zisserman and Joon Son Chung Visual Geometry Group, Department of Engineering Science, University of Oxford. Out of time: automated lip sync in the wild. [Online] 2016. [Cited: 01 03 2020.] <https://www.robots.ox.ac.uk/~vgg/publications/2016/Chung16a/chung16a.pdf>.
107. Chung, J.-S. and Zisserman, A. SyncNet. [Online] Github, 2016. [Cited: 01 03 2020.] https://github.com/joonson/syncnet_python.
108. MODALITY corpus. [Online] Multimedia Systems Department of Gdansk University of Technology, 2020. [Cited: 01 03 2020.] <http://www.modality-corpus.org/>.
109. youtube-dl. [Online] youtube-dl developers, 2020. [Cited: 03 03 2020.] https://ytdl-org.github.io/youtube-dl/supportedsites.html?fbclid=IwAR1wXIShygG8X7zrw3fqyl_NiBaG3YT_orORFutCTuNaXVKo-Sx3AquvTRQ.

110. Joshua Y. Kim¹, Chunfeng Liu¹, Rafael A. Calvo^{1*}, Kathryn McCabe², Silas C. R. Taylor³, Björn W. Schuller⁴, Kaihang Wu. *A Comparison of Online Automatic Speech Recognition Systems and the Nonverbal Responses to Unintelligible Speech*. s.l. : University of Sydney, Faculty of Engineering and Information Technologies² University of California, Davis, Psychiatry and Behavioral Sciences³ University of New South Wales, Faculty of Medicine⁴ Imperial College London, Department of Computing, 2019.
111. Hassanat, Ahmad Basheer. *Visual Words for Automatic Lip Reading*. Department of Applied Computing , University of Buckingham United Kingdom . 2009.
112. *Study of Influence of Word Lip Reading by Change of Frame Rate*. Takeshi Saitoh¹, Ryosuke Konishi. 2010. International Conference on Audio-Visual Speech Processing Hakone, Kanagawa, Japan.
113. Kamoun, Emna. Image Registration: From SIFT to Deep Learning. [Online] SICARA, 16 Jul 2019. [Cited: 26 04 2020.] <https://medium.com/sicara/image-registration-sift-deep-learning-3c794d794b7a>.
114. 111+ Linux Statistics and Facts – Linux Rocks! [Online] hostingtribunal. <https://hostingtribunal.com/blog/linux-statistics/>.