CS577 Final Project Report

End-to-end Audio-assisted Lip-Reading using LSTM

April 2019 | Serg Masís (A20427420), Srirakshith Betageri (A20414667)

1) PROBLEM STATEMENT

Lip reading is a challenge to both humans and machines alike. Primarily because of the existence of homophemes, (characters that produce the same lip movement. *B* and *P* for example). These problems can be overcome by using either neighbouring words or by using a language model.

Owing to advances in Deep Learning as well as by the presence of a large enough dataset, we're able to build a multitude of models that work together to solve the problem of audio-visual lip reading.

These models are as follows:

1. Isolating the region of the mouth in a video.
2. Extracting features from the audio.
3. Extracting features from regions of the mouth.
4. Encoding audio features through time.
5. Encoding video features through time.
6. Decoding both audio and video streams to generate text.

2) PROPOSED SOLUTION

The various tasks can be grouped together in modules as follows:

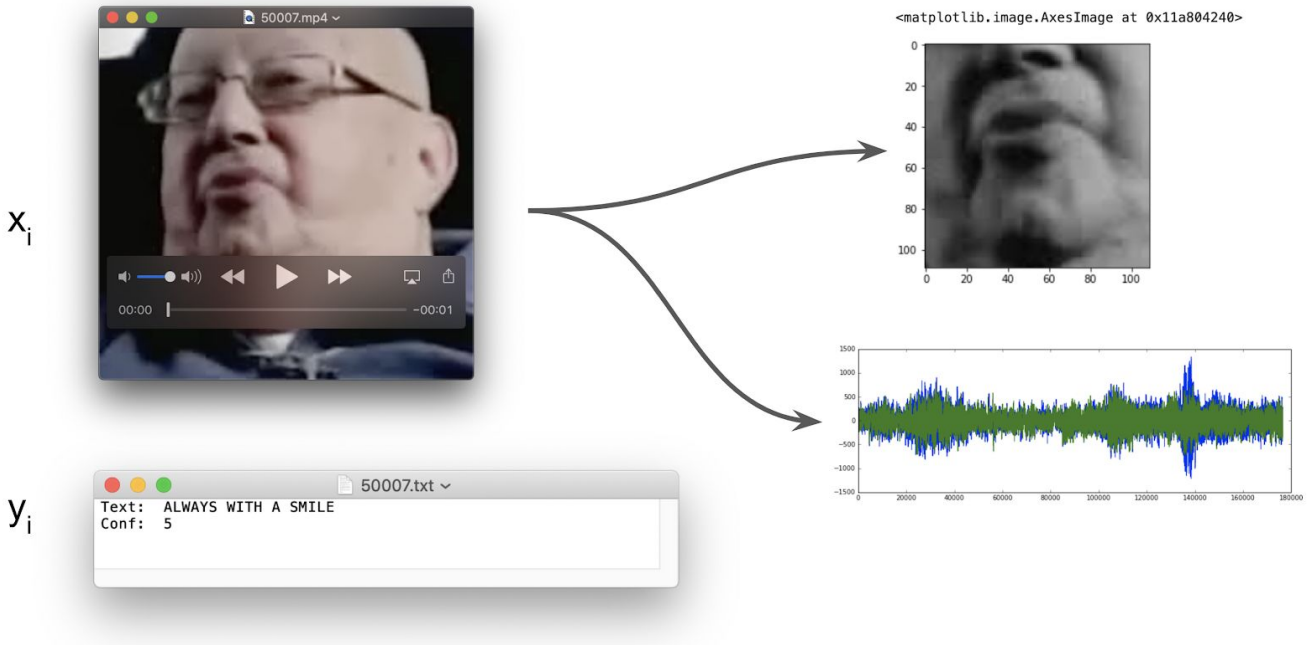
1. Watch Module, *Video Encoder*[4]:
    a. Isolate regions of the mouth : HAAR Cascades and Ensemble of Regression Trees [10] (this is done in pre-processing)
    b. Extract features from regions of the mouth: Convolutional Neural Network, SyncNet[3]
    c. Encode video features through time : Long Short Term Memory
2. Listen Module, *Audio Encoder*[5]:
    a. Extract MFCC audio features.
    b. Encode audio features through time : Long Short Term Memory

3. Spell Module, *Transducer for character decoding*[6, 7, 8]:
    a. Decode audio features using Attention Mechanism.
    b. Decode video features using Attention Mechanism.
    c. Use decoded features to obtain a prediction.

The authors of the main paper did not publish code and the data they used is not available so our implementation will have its challenges. The paper was originally written with the LRS(Lip Reading Sentences)[1] dataset. We are using the a TED talk annotated video dataset [2] instead, which was suggested as an alternative in the paper.

2.1) Data Preparation

The dataset we used, LRW3 (TED talk), consists of 4004 training videos, and 412 test videos, with a vocabulary of 17 000 words and 2000 words respectively. Further, we extracted just the regions surrounding the mouth and placed them on disk. To do this we first extracted every frame from every video (over 2.5 million frames in total). Then, we used another script to locate the mouths in every frame using a computer vision regression tree based model (Kazemi-Sullivan, 2014)[10]. Sometimes this was difficult when mouth was temporarily obscured but often if a neighboring frame had accurate mouth coordinates these coordinates could be used. An algorithm was used to fill in these gaps when possible but when it wasn't the entire video was marked as bad and not used at all in the training. We lost about 700 videos this way. For those remaining videos, we converted the frames to grayscale and stitched them back together to audio-less MP4 files and extracted the audio to MP3 files. We kept the label text files intact but they would be later tokenized in to character vectors. Similarly, the videos and audio would be split up into batches of frames and 10ms timesteps respectively.



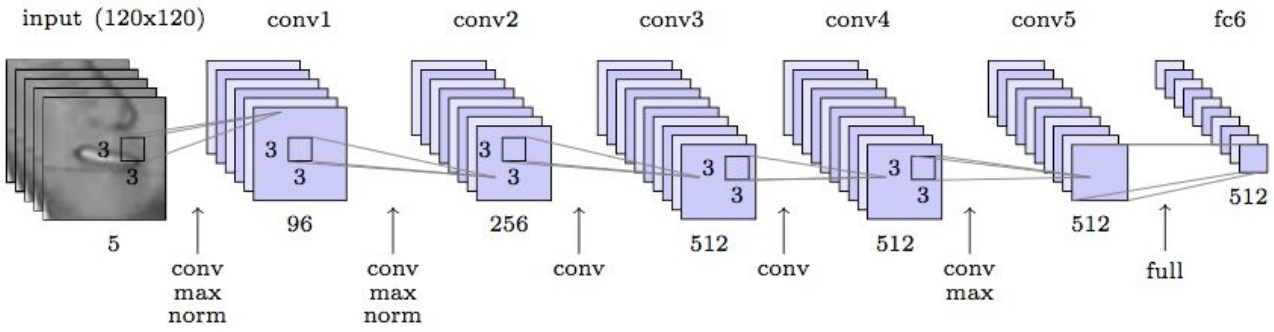
Data was pre-processed this way to make training more efficient because files can be converted to numpy arrays in then Pytorch tensors as is except for the audio and text files which need some minor transformations.

[1] "Lip Reading in the Wild (LRW) dataset." http://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrw1.html.

2.2) Model

WATCH:

Each region surrounding the mouth is fed through a Convolutional Neural Network, whose architecture is based on the VGG-M model [11].

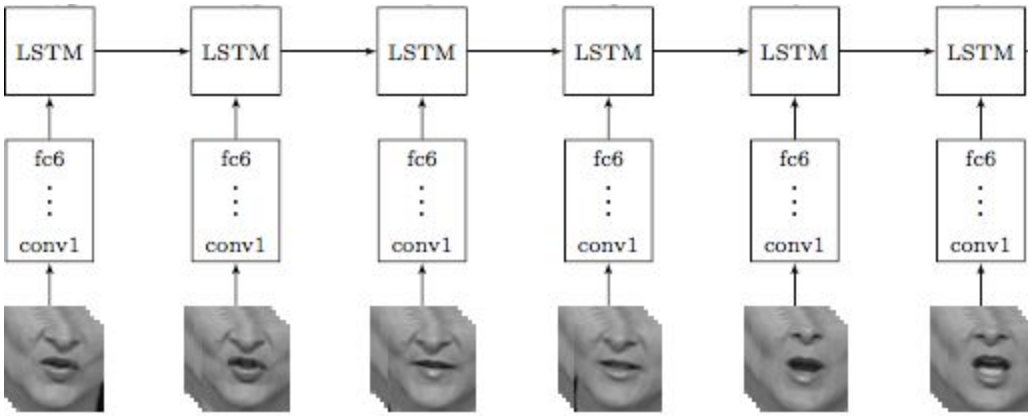


The input to the CNN is comprised of 5 frames of grayscale regions. The aim of this is to obtain a feature map of the respective regions. The frames follow a sliding window approach, this maximises the feature maps as well as provides adequate information to the encoder.

These feature maps are then fed in through to a 3 layer LSTM in order to encode the movements of the mouth. The output of each timestep is stored, so is the final hidden state of the LSTMs.

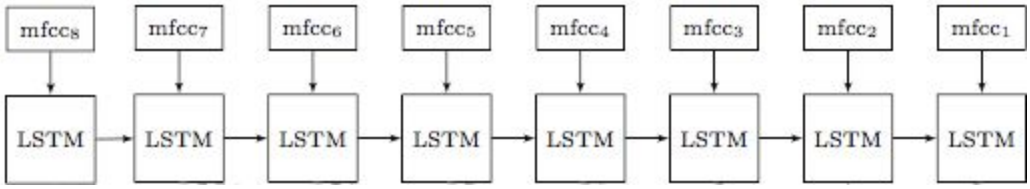
$$f_i = CNN(x_i)$$

$$h_i, o_i = LSTM(f_i, h_{i-1})$$



***LISTEN*:**

The audio, in the form of an mp3 file, is used to extract speech features. These features are then fed to an LSTM with the aim of encoding audio through time.



$$h_j,\ o_j = LSTM\left(x_j,\ h_{j-1}\right)$$

The LSTM ingests the 13 dimensional Mel-frequency cepstral coefficients (MFCC) features for every timestep (25ms window with 10ms step), to produce a state vector and output vector. The state vector is the last hidden state.

SPELL:

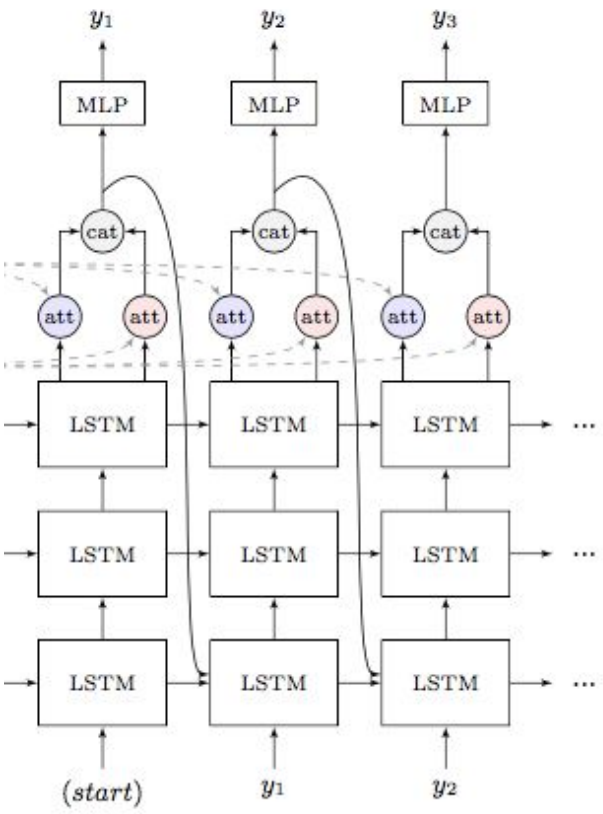
This module accepts text input during training and during inference, the previous prediction is fed back in. The text input is encoded through a 3 layer LSTM, which uses the hidden states of the LSTMs from both *Watch* and *Listen* as it's initial hidden state. This encoded text is then fed into the two attention networks. Each attention network are independently fed in either the output states of either *Watch* or *Listen*. The attention mechanism is crucial in our sequence to sequence model. An interesting aspect of this model is that the encoding is from two different data sources(audio and video) and used to predict a third data source(text).

$$h_k,\ o_k = LSTM\left(h_{k-1},\ y_{k-1},\ c^v{}_{k-1},\ c^a{}_{k-1}\right)$$

$$c^v{}_k = o^v\ Attention^v\left(h_k,\ o^v\right)$$

$$c^a{}_k = o^a\ Attention^a\left(h_k,\ o^a\right)$$

Output from the dual Attention network is concatenated and passed to an MLP which predicts a character. During inference, this predicted character is fed back to the *Spell* LSTM cell as input.



Spell consists of a 4 models. Two attention, one LSTM network and an MLP network. As input, the spell module requires the output states of the LSTMs of both Watch and Listen, as well as the final hidden states. During training, teacher forcing is employed wherein the next character from ground truth, is fed into the network. When teacher forcing is not used, the predicted value of the entire ensemble network is used as input to the lstm.

2.3) Training and Deployment

We developed the code on a Macbook Pro and a Macbook Air. In order to facilitate easy debugging and actual running of the deep learning code, we provisioned a much smaller dataset that we called the dev set. This consists of just 108 video, audio and text files. Once development was complete, we then used a Linux box running Ubuntu 16.04 to run the build the full model.

Specs:

- 3GHz Intel i5 CPU
- 16 GB DDR3 Memory
- Nvidia GTX 1070, 8GB GDDR5 Memory

Libraries:

- python==3.7.3
- python-Levenshtein==0.12.0
- pytorch == 1.0.1
- numpy=1.16.2
- Pandas==0.24.2
- scipy==1.2.1
- setuptools==41.0.0
- librosa==0.6.0

3) IMPLEMENTATION DETAILS

The end result should be a python program that can take a recorded video or live camera feed and as a speaker speaks display subtitles underneath the corresponding face in real time.

3.1) Model Training

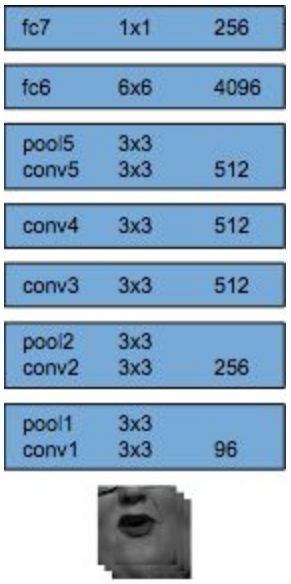
In order to facilitate similar working environments amongst the various machines, we used Conda to separate and create virtual environments.

The original dataset consists of just mp4 files and their corresponding texts. We extracted the mp3 files and saved them in the same directory, in order to facilitate easy training. We used the built in tool ***ffmpeg*** to achieve this.

Next, we indexed all the respective files in a ***Pandas*** dataframe which we then stored as a csv file. When building a custom dataset class, using PyTorch's Dataset class, this csv file is utilised in order to quickly access the training files.

WATCH:

Watch uses a CNN called SyncNet to extract the features. The authors of the original paper trained this CNN model to predict if audio and video from a given stream are in sync. In due process, the authors realised that the CNN is a good model to extract features of regions surrounding the mouth. We have used the same model, without retraining. It's architecture is given below.



SyncNet takes in 5 grayscale frames in each iteration. This is extracted from the mp4 file during time of read. Although part of the Watch module, these frames are passed as tensors, from the LRWDataset class. These 5 tensors then produce a single 512 dimensional feature map.

This 512 dimension vector is then fed to the 3 layer LSTM, which have a cell size of 256. The output of each of the LSTMs is stored, which is then fed to the Attention networks.

PyTorch has the following implementation of LSTMs, which we found to be adequate for our purposes.

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t c_{(t-1)} + i_t g_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

All weights are initialised as $u\left(-\sqrt{k}, \sqrt{k}\right)$, where $k = {}^{1}/_{hidden\ size}$

LISTEN:

13 dimensional MFCC vectors are extracted from the stored mp3 files. This is again done in the LRWDataset class, in order to reduce training time. These tensors are then fed to a 3 layered LSTM network with a cell size of 256. Once again, the output states and the final hidden output is stored, to be passed to the attention network.

SPELL:

The input to the spell module consists of the output states from Watch and Listen module. As well as the final hidden states. Further, the textual input is sent in as tensors which are then encoded as embeddings. The LSTM used here has a cell size of 512 and is 3 layered.

We aim to use the same training methods, as presented in the paper. We train by minimizing the ErrorRate [1]. Specifically we aimed to reduce the *Word Error Rate* (CER). We leveraged popular Levenshtein distance formula to this end which adds the insertions, deletions and substitutions necessary to make one word like another. Then, you divide this by number of words.

$$\textbf{Word Error Rate} = \frac{\textbf{Insertions + Deletions + Substitutions}}{\textbf{Number of Words in Reference Transcript}}$$

3.2) Instructions for using programs

3.2.1) Required components

- audioread 2.1.6
- cffi 1.12.3
- Cython 3.0a0
- decorator 4.4.0
- dlib 19.17.0
- fuzzywuzzy 0.16.0
- imutils 0.5.2
- joblib 0.13.2
- librosa 0.6.0
- llvmlite 0.28.0
- numba 0.43.1
- numpy 1.14.2
- opencv-python 4.1.0.25
- pandas 0.24.2

- Pillow 6.0.0
- pip 19.1
- pycparser 2.19
- python-dateutil 2.8.0
- python-Levenshtein 0.12.0
- pytz 2019.1
- PyYAML 5.1
- resampy 0.2.1
- scikit-learn 0.20.3
- scipy 1.0.1
- setuptools 41.0.1
- six 1.11.0
- sounddevice 0.3.13
- sox 1.2.8
- torch 0.4.1
- torchvision 0.2.0
- wheel 0.33.1

3.2.2) File structure

```
/src
├/python-programs
|     ├ demo1.py (the demo for live testing using your device's front-facing camera)
|     ├ demo2.py (the demo for live processing of video file)
|     ├ setup.py (can be used to generate a .so of the estimator class)
|     ├ talkpredict.py (IMPORTANT: most of the prediction functionality is here)
|     ├ model.py (IMPORTANT: for decoding the model)
|     └ decoder.pyx (IMPORTANT: for decoding the results of the model)
├/jupyter-notebooks
|     ├ 1_video_to_frames.ipynb (was used to test 3D points and gaussian yawn estimation)
|     ├ 2_frames_to_lips.ipynb (was used to test 3D points and gaussian yawn estimation)
|     ├ JL_Dataset.ipynb (was used to test 3D points and gaussian yawn estimation)
|     └ SM_scratchpad.ipynb (was used to analyze batch processed videos ~ only a sample here)
└/training (to train the model)
      ├ LipReading.py (to run the training)
      ├ LRWDataset.py (to prepare the dataset)
      ├ watch/
           └ Watch.py (watch module)
      ├ listen/
           └ Listen.py (listen module)
      ├ spell/
           └ Spell.py (spell module)
      └ attention/
           └ Attention.py (attention module)
```

3.2.3) Running programs

To run most of the programs and jupyter notebooks you need the models and the dataset. We have made all models available via Google Drive and a sample of the dataset is available as well. We can't make the 11 gigabytes of the full dataset we used available but the 108 test samples we uploaded should do for demonstration purposes.

For the **python programs**, you put the models in a models folder in this directory and then:

```
python demo1.py
python demo2.py /path/to/video.avi
```

They both work exactly the same except the first one uses the video from your camera and the second one from a video file (path sent as an argument). These are the keys they all use:

- **'p':** Show facial landmarks
- **'b':** Show bounding box
- **'m':** Show mouth info
- **'n':** Show nothing
- **'r':** Refresh/clear the frame of all info back to defaults
- **'l':** Save log file
- **'q':** Quit the program
- **'h':** Help ~ show this information

For **jupyter notebooks**, they can be opened with anaconda and run if all the required libraries are installed and the datasets located in the expected locations.

For **training** files, they expect the Syncnet model for the CNN weights, and a path to a dev/ directory with the CSV with the paths to the dataset files and a root dir for this dataset files. This is hardcoded in the LipReading.py file. Once this has been properly set it can be run like this:

```
python LipReading.py
```

4) Implementation issues

4.1) Research Issues

We were wondering if the models would be robust to the loss of an entire datastream, as in, whether the loss of the video stream would hamper the prediction when the model just relies on audio. Likewise, how would the model react when the audio is cut off and it has to rely only on the video stream.

4.2) Dataset Issues

The authors originally wrote the paper with the LRS2 dataset, which is a recording of BBC Television broadcasts. However, we were able to obtain only the LRS3 dataset which is a recording of TED talks. The difference between these datasets is drastic, with LRS2 having close to 5000 hours of video content while LRS3 only has 400 hours.

The original authors also had a fully trained SyncNet model, for 120x120 regions, that they leveraged to initialise the weights for the CNN in the Watch module. These weights are not publicly available and hence we had to change our CNN model to ingest full face features as opposed to just the regions surrounding the mouth. We think this greatly impacted our accuracy.

4.3) Design Issues

Since we try to emulate the papers methods as much as possible, designing the deep learning networks wasn't a challenge except when details were omitted. This was particularly challenging with the spell and attention modules where the particulas of the seq2seq [12] architecture was not detailed and did not match what was outlined in their diagrams. We had to fill in the gaps using our intuition of how it could be put together. We had no sample code to compare so it took us a while to get the architecture pieced together in such a way it would even run.

4.4) Programming Challenges

Ensuring the dimensions match between the various modules was a challenge, particularly because the frames/tensors from Listen needn't have the same dimensions as Watch.

Batch Processing of the respective tensors/frames that are fed to the respective modules need to be of an uniform size and due to the nature of the problem and the dataset, padding the tensors proved to be tricky. It was tricky as we had to ensure that the extra padding doesn't skew the weights and hence the results in the wrong direction.

Development was done primarily on machines with CPUs, thus extra attention had to be paid to move the tensors to the required device. PyTorch does not implicitly move all tensors to GPU.

4.5) Testing Challenges

Testing was plagued with similar problems as programming. The biggest hurdle we faces was during the transition from GPU to CPU. Further, we felt that generating some of the tensors on the fly was causing a data flow bottleneck, with the GPU waiting for data to be moved from the CPU. This increased training time significantly.

5) Results and discussion

5.1) Test Accuracy

Our test accuracies were calculated with two metrics:

Word Error Rate (WER)	Character Error Rate (CER)
43.1%	34.6%

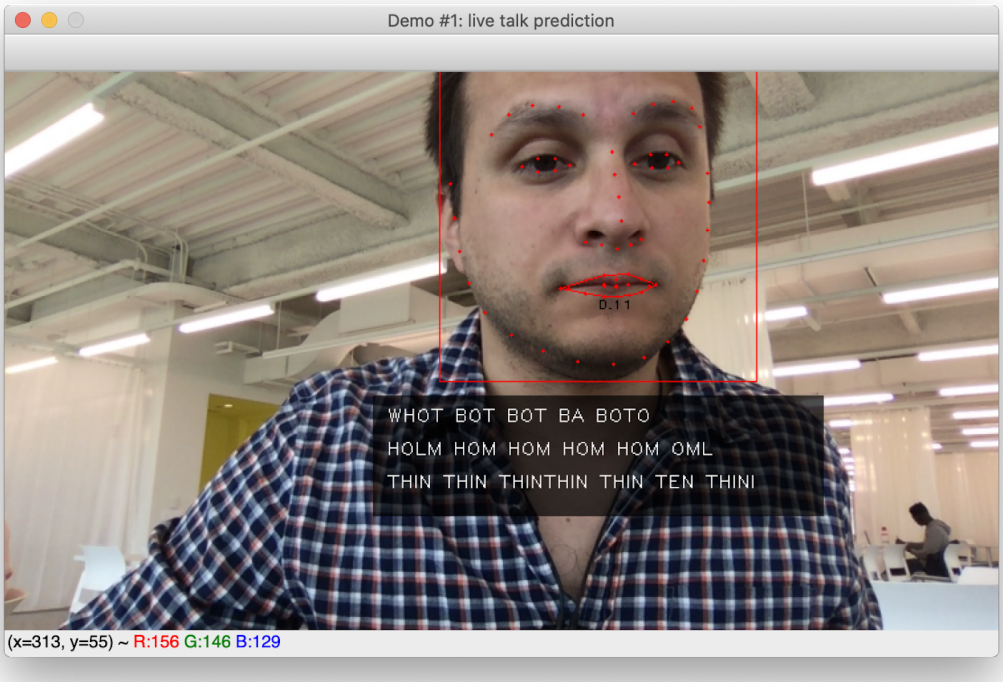
These numbers are quite high when you compare specifically with the *Audio and lips* section of the table in the paper which ranges between 13.9-42% and 7.9-29.8% respectively depending on the SNR.

Method	SNR	CER	WER	BLEU†
Lips only				
Professional‡	-	58.7%	73.8%	23.8
WAS	-	59.9%	76.5%	35.6
WAS+CL	-	47.1%	61.1%	46.9
WAS+CL+SS	-	42.4%	58.1%	50.0
WAS+CL+SS+BS	-	39.5%	50.2%	54.9
Audio only				
Google Speech API	clean	17.6%	22.6%	78.4
Kaldi SGMM+MMI*	clean	9.7%	16.8%	83.6
LAS+CL+SS+BS	clean	10.4%	17.7%	84.0
LAS+CL+SS+BS	10dB	26.2%	37.6%	66.4
LAS+CL+SS+BS	0dB	50.3%	62.9%	44.6
Audio and lips				
WLAS+CL+SS+BS	clean	7.9%	13.9%	87.4
WLAS+CL+SS+BS	10dB	17.6%	27.6%	75.3
WLAS+CL+SS+BS	0dB	29.8%	42.0%	63.1

We didn't train a model on only the lips or only the audio so we don't have metrics to compare to this but we note that ours aren't as impressive because they were trained on 1/12th of the data and with likely differences in our models. We tried to reach the Oxford researchers for answers regarding several discrepancies and omissions in the paper for clarification to no avail. We also wonder why text was meant to be tokenized by character when we've seen other very accurate models work with words or at least trigrams. This paper

actually stands out because it tokenizes by character yet allegedly yields relatively high accuracy. Had we had more time, we would had try to train it on a word based corpus comprising the entire dataset.

5.2) Empirical Testing



We wrote demo programs to demonstrate the model live but given the complexity and heaviness of this model (300Mb+) and significant lag in frame rate the predictions made are mostly gibberish although occasionally it will predict a word or two although strangely enough repeat it 3-6 times. We suspect that somehow given the streaming nature of the frames and audio spectrum, it is getting replicated more than once in the model and therefore getting predicted multiple times with slight variations. This is an implementation flaw we couldn't look in to further. However, in batch mode processing recorded videos, predictions are much better more or less matching error rates produced during training.

5.3) Strengths and Weaknesses

Due to the multi-modal nature of the model, we expect it to work well even under noisy conditions, as well as when the speaker's mouth is obstructed.

We also feel that the model can be retrained to predict words and characters from any language. It can also be used to identify the speaker in a multi-speaker environment.

This model has only been tested on small sample sizes of both video and audio. Due to the nature of recurrent neural networks and LSTMs, there is a high possibility that the model fails to predict longer sentences or when the model has been used for a long period of time.

6) Bibliography

1. Chung, J.S., Senior, A.W., Vinyals, O., & Zisserman, A. (2017). *Lip Reading Sentences in the Wild*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3444-3453. (ISBN: 978-1-5386-0457-1).
2. Afouras, T., Chung, J.S., & Zisserman, A. (2018). *LRS3-TED: a large-scale dataset for visual speech recognition*. CoRR, abs/1809.00496.
3. Chung, J.S., & Zisserman, A. (2016). Out of Time: Automated Lip Sync in the Wild. ACCV Workshops.
4. K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In Proc. BMVC.,
5. J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio. End- to-end continuous speech recognition using attention-based recurrent nn: first results. arXiv preprint arXiv:1412.1602, 2014.
6. J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. In Advances in Neural Information Processing Systems, pages 577–585, 2015.
7. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. Proc. ICLR, 2015.
8. W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals. Listen, attend and spell. arXiv preprint arXiv:1508.01211, 2015.
9. S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In Advances in Neural Information Processing Systems, pages 1171–1179, 2015.
10. Kazemi, Vahid and Josephine Sullivan. “One millisecond face alignment with an ensemble of regression trees.” *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014): 1867-1874.
11. Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.
12. Sutskever, I., Vinyals, O., & Le, Q.V. (2014). Sequence to Sequence Learning with Neural Networks. *NIPS*.