GuNiLeo: An Attempt To Perform Lip Reading From Videos With STCNN

Beray Nil Atabey (*2045576*) Leonardo Biason (*2045751*) Günak Yüzak (*2048950*)

***Abstract*—Lip reading is a task that can have various usages as an accessibility feature, but it's also very complex to design: it requires a machine to be able to differentiate between the various words said by a speaker, and also to predict what the speaker said whenever words aren't spelled with a precise motion of the lips. With this paper, we propose a model based on a Spatio-Temporal CNN, capable of reading the words said by a speaker from a video clip of maximum 75 frames.**

Sapienza, AcsAi, STCNN, Lip reading, CTC Loss, Computer Vision

I. Introduction

With this paper, we describe an attempt on building a lip reading architecture, using videos up to 75 frames, using a dataset of 5000+ videos. For the model the PyTorch framework was used, along with a helper pre-trained model for landmarking the face. The model uses convolutional and recurrent neural networks, where convolution is in 3D and recurrency is bidirectional. Our model takes inspiration from the LipNet paper [1], which also employs many of the techniques that we used. The LipNet acquired a 95% accuracy metric, which is by no doubt an accomplishment. In this paper, however, we don't have an accuracy metric, as it will be explained further.

II. Implementation

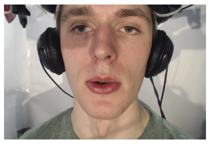
In order to create a lip reading model, the following steps have been undertaken:

1) creation and modeling of the dataset;
2) creation of the model;
3) training of the model;
4) evaluation of the model.

III. Creation and Modeling of the Dataset

The project adopted an already existing dataset from the University of Sheffield: such dataset is called Lombard Grid [2].

Fig. 1. Example of frame (on the left) and label (on the right)



```
{
    "video_name": [
        {
            "duration": 0.01,
            "offset": 0,
            "phone": "SIL_S"
        }
    ]
}
```

Frame Label

A. Structure of the Dataset

The dataset comprehends two fundamental parts: the data part and the labels. The data consists in a set of 5000+ videos, where each video records frontally a person saying a specific sentence. This sentence is made up of 6 parts, which are in order: COMMAND, COLOR, PREPOSITION, LETTER, DIGIT, ADVERB. The possible values are the following:

- **Command**: `bin`, `lay`, `place`, `set`;
- **Color**: `blue`, `green`, `white`, `red`;
- **Preposition**: `at`, `by`, `in`, `with`;
- **Letter**: `a-z` except `w`;
- **Digit**: `0-9`;
- **Adverb**: `again`, `now`, `please`, `soon`.

There are 4 words for command, color and preposition, 25 values for letter, 10 values for digit and 4 words for adverb, thus having 64000 different combinations. Each video comes with its pair, alignment data which after this we will refer as labels. Each label is transcribed in a `json` file, which has the following format:

- `video_name`: a list containing all the phonemes and the timing of said phonemes of a clip;
  - `duration`: the duration in seconds of the phoneme;
  - `offset`: the beginning of the phoneme, with respect to the beginning of the video;
  - `phone`: an encoding of the phoneme.

An example of data-label pair can be observed in Figure 1, where a frame of a sample video has

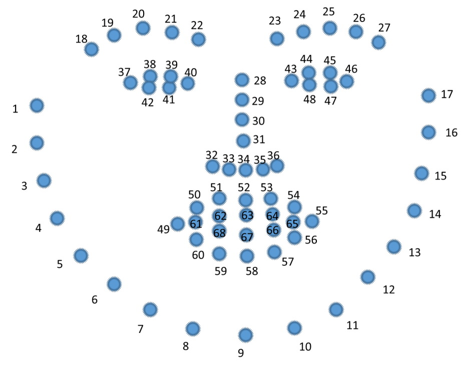


Fig. 2. Face landmarking points

been reported, alongside an example of possible label. The label can be interpreted as follows: there is a phoneme, the silent "s", which starts at $t = 0$ and lasts for $0,01s$.

B. Pre-processing of the Dataset

The pre-processing of the dataset took place in two steps: first, the pre-processing of the data, then, the pre-processing of the labels.

Pre-processing of the Videos: Since we want to read the lips, the only area that concerns us is the mouth and the close surroundings. The rest of the video can be cropped to avoid unnecessary memory-wise and computational power-wise waste. In order to locate the mouth, the `dlib` [3] library's face detection and landmark detection algorithm has been employed. The face detection is implemented in `dlib` itself, however for the landmark an external model had to be used. We now proceed to describe the undertaken steps for the pre-processing of the data:

1) First, the face detection algorithm is ran on the video in order to find the coordinates of the face;
2) Such coordinates are given as input into the landmark model, which determines the important landmarks of the face via a sequence of 68 points. Points from 49 to 68 determine where the mouth is (such points are shown in Figure 2);
3) From these points, it's possible to find the points that are on the left,right, up and down most of the mouth, which correspond to the maximum coordinates that a mouth can take in a specific video;
4) From these coordinates a margin distance is added, and after that the video gets cropped around the mouth. Clearly, every person's mouth size is different from one another, so the resize procedure reshapes the video to a fixed size of 150×100 pixels;
5) Finally, the resized video is then gray scaled and normalized for a better performance of the model.

Pre-processing of the Labels: Rather than dealing with the phonetics as presented in the `json`'s, we tried a different approach. We extracted what was being said in the videos from their file name, since the file name is made up by taking the first letter from the 6 spoken words. Every word in the categories are different in the same category by the first letter so no confusion is made this way. We then craft our label as a list of the characters in the said sentence, including the spaces.

IV. Architecture of the Model

The model used for this project takes large advantage of the Spatio-Temporal Convolutional Neural Network concept [4] (from now on referred to as STCNN). The choice of using such type of neural network is because of its capabilities of retaining information in the long short-term, which is very helpful when it comes to analysing multiple videos and making predictions on the long run.

Here follows the detailed architecture of our model, with all its layers:

- STCNN and 3D Max Pooling ×3: a STCNN layer has the following equation

$$[\text{stconv}(\mathbf{x},\mathbf{w})]_{c'tij} = \sum_{c=1}^{C}\sum_{t'=1}^{k_t}\sum_{i'=1}^{kw}\sum_{j'=1}^{kh} w_{c'ct'i'j'}x_{c,t+t',i+i',j+j'} \tag{1}$$

- Bi-directional GRU unit ×2;
- CTC loss function;
- log-Softmax activation function.

As input, the model takes a clip of size (100, 150, 75, 1), and it returns a tensor of shape (75, 37). The possible output, once encoded, is a list composed by letters from a-z, numbers from 0-9 or blank spaces "␣". A summary of the model is shown in Table I.

The model supports 75 frames videos, which is a video of 3 seconds with a constant frame rate of $25f$. The videos that had more than 75 frames are discarded from the dataset, and less then 75 frames are extended with 0 matrices.

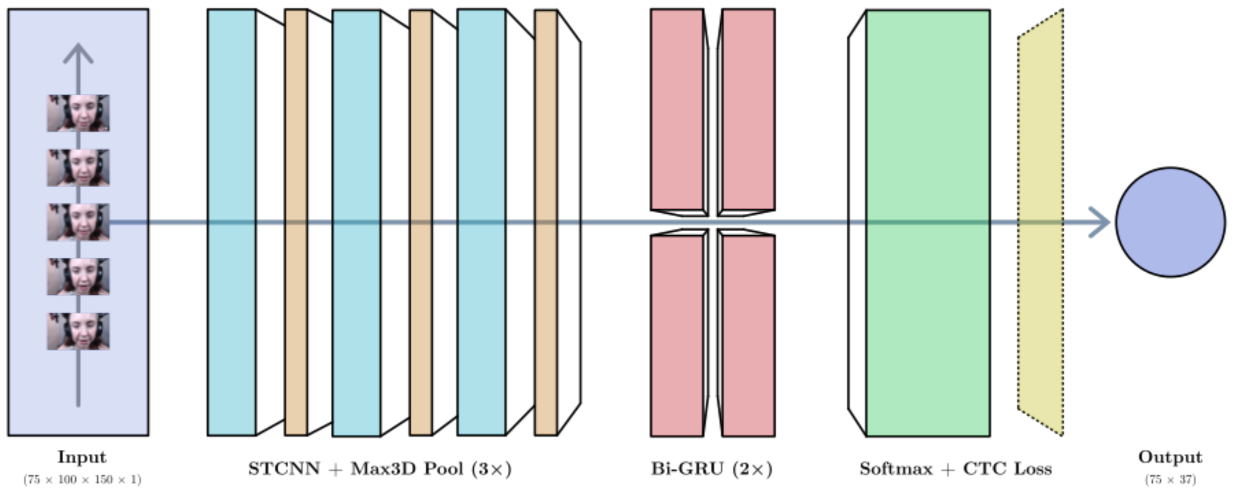


Fig. 3. Visual representation of the model

A. *Hyperparameters*

The model was trained with the k-fold cross validation technique, where $k = 5$, and with the following hyperparameters:

- 10 **epochs**;
- a **learning rate** of 10^{-4};
- a **dropout** of 0.5.

We opted for the **AdamW** optimizer, because of its computational efficiency. Given that the training of the model would've required various resources and time, it seemed a reasonable option.

The loss function $\mathcal{L}$ used in the model is a **Connectionist Temporal Classification (CTC)** Loss function, which employs log probabilities in order to detect a sequence of characters within a piece of audio or video (in our case, it will be on videos).

V. Failure Cases

Although this project lead to a functioning model, the creation of the model was difficult and many prototypes failed before. Some conclusions can be drawn. Why didn't it work? Why did some issues arise? We will thus try to answer to such questions.

First and foremost, why didn't it work as expected? Our biggest problem was understanding CTC loss. Many times in the implementations we faced an error, and even when the model seemed the work without any errors, the loss values were either negative or not a number (`nan`). It took us quite a long time to figure out what exactly was the problem.

The key reason why it did not work is that we did not know that CTC Loss demanded a log-Softmax activation function over a normal Softmax layer. This led to the loss values being completely unusable.

Another problem that we faced was with the implementation of the CTC Loss function in PyTorch, where it demanded, from the log-Softmax layer, an additional specific output, which was a character called `blank`. This is an important part of functionality of CTC loss.

We fixed these issues, yet unfortunately we did not have enough time and resources to train a new working model. During the training of a model with the correct settings, we observed that the loss values assumed feasible values.

This leads to think that, with enough time and resources, the model would eventually train well enough in order to have a satisfiable accuracy. We used a small number of epochs because of the necessity of this model, but the experiment could be easily replicated by using a wider number of epochs and by, possibly, augmenting the number of items in the dataset.

VI. Conclusions

During the making of this model, various issues have been faced. However, this gave us the opportunity to learn various aspects of how to build an efficient model and what mistakes should be avoided:

1) It is extremely important to paralellize the Data Loader, especially if, like in our case, the process-

TABLE I
VALUES AND DIMENSIONS OF THE MODEL

Layer	Size	Stride	Padding	Input	Dimensions
STCNN	$3 \times 5 \times 5$	(1, 2, 2)	(1, 2, 2)	$(1 \times 75 \times 100 \times 150)$	$(C \times T \times H \times W)$
Max3D Pool	$1 \times 2 \times 2$	(1, 2, 2)	/	$(8 \times 75 \times 50 \times 75)$	$(C \times T \times H \times W)$
STCNN	$3 \times 5 \times 5$	(1, 1, 1)	(1, 2, 2)	$(8 \times 75 \times 25 \times 37)$	$(C \times T \times H \times W)$
Max3D Pool	$1 \times 2 \times 2$	(1, 2, 2)	/	$(16 \times 75 \times 25 \times 37)$	$(C \times T \times H \times W)$
STCNN	$3 \times 5 \times 5$	(1, 1, 1)	(1, 2, 2)	$(16 \times 75 \times 12 \times 18)$	$(C \times T \times H \times W)$
Max3D Pool	$1 \times 2 \times 2$	(1, 2, 2)	/	$(32 \times 75 \times 12 \times 18)$	$(C \times T \times H \times W)$
Reshape	$(C \times T \times H \times W) \implies (T \times C \times H \times W)$				
Flatten	/	/	/	$(75 \times 32 \times 6 \times 9)$	$(T \times C \times H \times W)$
Bi-GRU	256	/	/	(75×1728)	$(T \times (C \times H \times W))$
Bi-GRU	256	/	/	(75×512)	$(T \times P)$
Linear	37	/	/	(75×512)	$(T \times P)$
log-Softmax	37	/	/	(75×37)	$(T \times L)$

C: Channels T: Time H : Height W : Width P : Placeholder L : Label size

ing of the videos happened on run time. Being able to parallelize the video processing tasks could save time and resources, which, on larger scale models, would be very helpful;

2) The `cuda` framework, just as any other framework that allows to run code on a GPU, is essential when performing model-training related tasks. Unfortunately for us, the `dlib` library, which was essential for our model, did not have any support for the `cuda` framework and had no options for running its code on GPUs. This led to a non-parallelized Data Loader, which increased the resources used and the time consumed;
3) The PyTorch framework and the possibility of coding every aspect of the model from scratch gave us a good opportunity to better understand how to write a model and how to train it. While many times we had issues with it, the time spent understanding the errors and troubleshooting them was worth for learning how to properly code a model. In particular, by tinkering with the CTC Loss we got a way to better understand the used algorithms and how they behave with other components of the model;

REFERENCES

[1] Y. M. Assael, B. Shillingford, S. Whiteson, and N. de Freitas, "Lipnet: End-to-end sentence-level lipreading," 2016.

[2] U. of Sheffield, "The audio-visual lombard grid speech corpus," last accessed 23 June 2024. Available here.

[3] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[4] Z. He, C.-Y. Chow, and J.-D. Zhang, "Stcnn: A spatio-temporal convolutional neural network for long-term traffic prediction," in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, 2019, pp. 226–233.