Detecting temporal boundaries in sign language videos

Paul-Emile Zafar
paul-emile.zafar@telecom-paris.fr
January 2022

1. Introduction

This projects focuses on the segmentation of sign language videos. In order to annotate sign language videos, it is necessary to know the boundaries between the different signs in a video where sign language is performed. I first read and reproduced the results of the main paper this project is based upon [3], then I explored the use of a transformer architecture in the last part of the segmentation process. Both parts of the projects targets fully supervised results, using the British Sign Language Corpus Project (BSLCP) data [1].

2. Paper results

2.1. Presentation

The *Sign language segmentation with temporal convolutional networks* paper by Katrin Renz, Nicolaj C. Stache, Samuel Albanie and Gül Varol [3] presents a pipeline to label the frames that separates different signs within a sign language video. Formally, given an sequence $(x_1, ..., x_N)$, the objective is to predit the sequence $(y_1, ..., y_N) \in \{0, 1\}^N$, where 1 labels a **boundary** frame. Boundaries frames separates signs in the video.

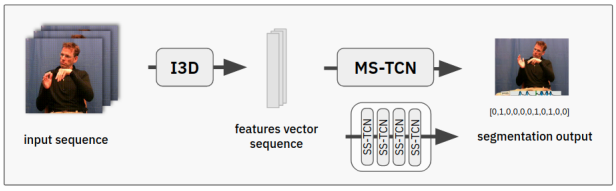

Figure 1: Summary of the method used in [3]

The presented method is based on two main steps:

- Frame by frame feature extraction using the I3D model [2], trained on the Kinetics dataset, whose output can be interpreted as a good representation of the body pose in each frame, to later be able to classify each frame as boundary or not. The output of this step is a 1024 length feature vectors sequence (one vector per input frame).
- Training of a Multi Stage Temporal Convolutional Networks (MS-TCN), a sequence-to-sequence model that acts as the segmentation model. The MS-TCN takes the features output sequence of I3D as an input and outputs a binary sequence of $\{0, 1\}$ elements.

2.2. Provided code and asset

The authors provided the full Python code for the experiments, and detailed explanations to reproduce the results. Pretrained models are also provided as a whole in order to test the demo on sample videos. In order to evaluate the method on a qualitative point of view, I ran the demo on the provided video segment (extracted from the British Sign Language Corpus Project (BSLCP) [1]), and then on other video parts found on the dataset website), as shown in Figure 2.

Figure 2: Results on a BSLCP example

From a simple visual inspection, and without specific knowledge in sign language, the segmentation does provide boundaries for all gestures in the video segment. Those boundaries frames (highlighted in blue in the result graphical display) also appear to match the moments in between active signs. Overall and qualitatively, the results are very convincing for an untrained eye.

2.3. Reproductibility

The training result were also reproducible with detailed explanations. I was able to execute the training of the MS-TCN model on the annotated BSLCP corpus. In order to

evaluate a segmentation, the paper proposes two metrics: mF1B and mF1S. Both metrics estimate the correctness of a boundary if its distance (resp. its IOU) is below (resp. above) a given average, and then average all F1 scores over several thresholds. I ran the training for two different seeds and obtained very similar scores as the paper (see Table 1)

Metric	mF1B	mF1S
Paper	$68.68_{\pm 0.6}$	$47.71_{\pm 0.8}$
Reproduced	$68.93_{\pm 0.15}$	$47.8_{\pm 0.89}$

Table 1: Original scores vs reproduced ones

The paper results though instructions and detailed code was easily reproducible, and on my side the obtained results totally fit the announced ones.

3. Transformer based architecture

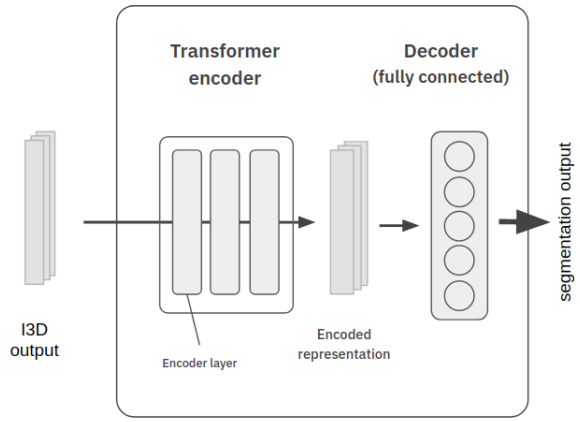

Figure 3: Transformer based adaptation

3.1. Motivation and architecture

The next objective of the project was to replace the MS-TCN by an architecture based on transformers [4]. Transformers are widely used in NLP tasks and their attention based architecture takes into account the temporal relation between the sequence elements, making it suitable for this task.

3.1.1 Encoder

Using standard transformer encoder layers for the encoder part, the chosen model architecture was made of multiple transformer encoder layers. Transformer encoder layers core attention is modeled through multiple attention heads, combined and followed by a feed forward layer.

In this context of segmentation, there is no restriction on what part of the input is accessible during inference, masks are not needed as they can be for sequential token prediction (in NLP taks for instance). Additionally, because we worked with already extracted features from the video frames, no embedding was necessary before the encoder module.

Finally, to inject position information in the input, I used the positional encoding proposed by the authors of [4], defined by:

$$PE_{(\text{pos},2i)} = \sin(\text{pos}/10000^{2i/d_{\text{model}}})$$

$$PE_{(\text{pos},2i+1)} = \cos(\text{pos}/10000^{2i/d_{\text{model}}})$$

where pos is the time position, and i the dimension index in the encoder input.

3.1.2 Decoder

Our goal here is to output the segmentation sequence whose values lie in $0, 1$. Therefore, as suggested, after a transformer-based encoder, we consider only a set of fully connected layers that will act as a decoder on the *encoded representation*.

The resulting architecture is depicted in Figure 3

3.2. Experiments

Given this model architecture, I focused on tuning the following hyper parameters:

- Number of stacked transformer encoder layers n_{layers}
- Number of attention heads n_{heads}
- Feed forward layer dimension in transformer encoder layers dim_{ff}
- Dropout in the transformer encoders

I used the mF1B and mF1s scores in order to evaluate the models for each set of hyper parameters.

When varying the number of attention heads in encoder layers, I only noticed a slight improvement of the results, but within a very restricted range. Figure 4 shows the evolution of the mF1B and mF1S for $n_{heads} \in \{2, 4, 8\}$.

Varying the number of encoder layers was more promising, as it showed real improvements in mF1S and mF1B scores. Figure 5 show the evolution of the scores (F1B and F1S scores lie in different ranges) for $n_{layers} \in \{2, 4, 8, 10\}$.

Above 10 encoder layers, I noticed important convergence issues, where the convergence of the model seemed to be delayed for several epochs, and could not manage to make it converge for a number of layers strictly above 12. Overall, the 10 layers model seemed to make the

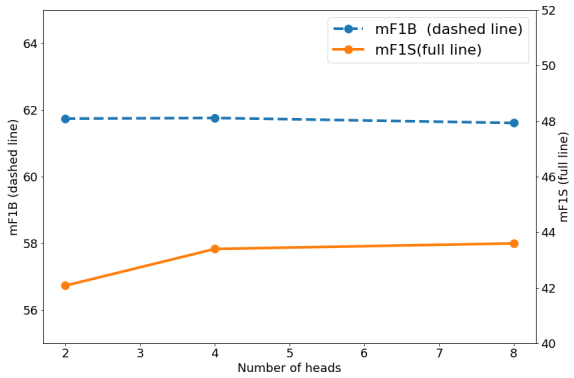

Figure 4: $n_{layers} = 8, n_{heads} \in \{2, 4, 8\}$, $dim_f = 1024$

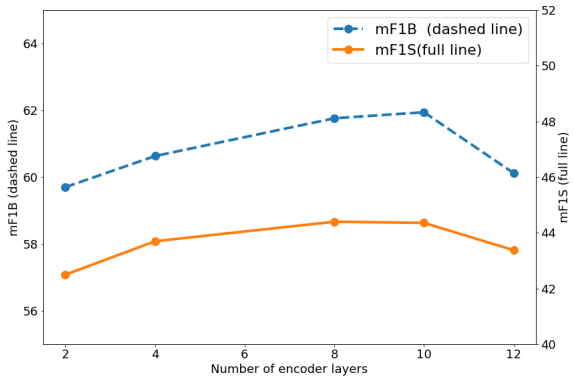

Figure 5: $n_{layers} \in \{2, 4, 8, 10\}$, $n_{heads} = 4$, $dim_f = 1024$

best results, and we can attribute the 12 layers loss in performance to the difficulty to make the model converge properly.

I then experimented with the the dimension of the encoders feed forward layers. Increasing this hyper parameter also improved the performance of the model, but also made it significantly more expensive to train. Overall the best results were obtained with 2048 neurons in the feed forward layers, and further experimenting with 4096 neurons did not noticeably improved the results and made training time longer, so this direction was no further explored.

Changing the dropout value in the encoder layers did not improve results, nor reduced overfitting, so a value of 0.4 was kept. Regarding the decoder part, I did not notice improvements when complexifying the decoder architecture (adding more layers, with more or less neurons). I thus kept a single final decoding layer for most the experiments described below.

The final model was trained with an Adam optimizer with a learning rate of 10^{-4}. The learning rate was adapted during training with an exponential scheduler of coefficient 0.9.

Globally, the best results were obtained with the following parameters:

- 4 attention head per transformer encoder layer
- 10 encoder layers
- 2048 neuron per encoder feed forward layer
- Dropout $p = 0.4$

Metric	mF1B	mF1S
Result	62.79	43.79

Table 2: Best results obtained with the new architecture

3.3. Issues

Convergence was a main issue faced when increasing the model complexity (mainly with the number of encoder layers) as we can wee in Figure 6 where 12 layers were used. The model seemed to hardly converge during the first few epochs. Moreover, when trying to investigate the results, the problem of video segmentation task, made it difficult to estimate the flaws and to have a grasp at what needed to be changed in order to improve the results.

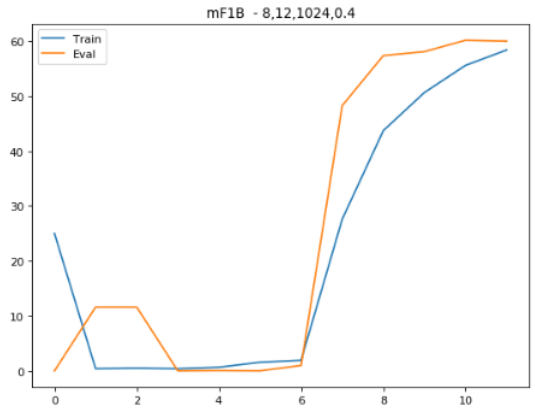

Figure 6: mF1B score across epochs: $n_{layers} = 12, n_{heads} = 8, dim_f = 1024$

4. Technical details

All the code of the original paper and the results obtained in this project were implemented using Python and its libraries, and all neural network layer implementation and training used the PyTorch library. All experiments were conducted on one Tesla K80 GPU.

References

[1] R. Rentelis A. Schembri, J. Fenlon and K. Cormier. British sign language corpus project: A corpus of digital video data and annotations of british sign language. `http://www.bslcorpusproject.org`, 2017.

[2] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

[3] Katrin Renz, Nicolaj C. Stache, Samuel Albanie, and Gül Varol. Sign language segmentation with temporal convolutional networks, 2021.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.