Sign Language Translation Mobile Application and Open Communications Framework

Deliverable 3.2: Sign Language Recognition Component and Models

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 101017255

Project Information
Project Number: 101017255
Project Title: SignON: Sign Language Translation Mobile Application and Open Communications Framework
Funding Scheme: H2020 ICT-57-2020
Project Start Date: January 1st 2021

Deliverable Information
Title: Sign Language Recognition Component and Models
Work Package: WP 3 - Source Message Recognition, Analysis and Understanding
Lead beneficiary: Ghent University
Due Date: 30/06/2023
Revision Number: V0.3
Authors: Mathieu De Coster, Joni Dambre, Ruth Holmes, Ellen Rushe
Dissemination Level: Public
Deliverable Type: Other

Overview: This document describes the work done on the sign language recognition component, the sign language recognition models, and their integration into the component. We discuss the model architectures, the training and inference procedures, and the data with which the models are trained. We also provide an overview of the performance of these models in the form of recognition accuracy.

Revision History

Version #	Implemented by	Revision Date	Description of changes
V0.1	Mathieu De Coster, Joni Dambre, Ruth Holmes, Ellen Rushe	16/06/2023	Initial version
V0.2	Mathieu De Coster	17/06/2023	Implement feedback from consortium members on V0.1
V0.3	Mathieu De Coster	26/06/2023	Implement feedback from consortium members on V0.2

The SignON project has received funding from the European Union's Horizon 2020 Programme under Grant Agreement No. 101017255. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the SignON project or the European Commission. The European Commission is not liable for any use that may be made of the information contained therein.

Approval Procedure

Version #	Deliverable Name	Approved by	Institution	Approval Date
V0.1	D3.2	Shaun O'Boyle	DCU	16/06/2023
V0.2	D3.2	Marco Giovanelli	FINCONS	21/06/2023
V0.2	D3.2	Vincent Vandeghinste	INT	19/06/2023
V0.2	D3.2	Adrián Núñez-Marcos	UPV/EHU	21/06/2023
V0.1	D3.2	John O'Flaherty	MAC	16/06/2023
V0.2	D3.2	Horacio Saggion	UPF	18/06/2023
V0.2	D3.2	Irene Murtagh	TU Dublin	20/06/2023
V0.2	D3.2	Lorraine Leeson	TCD	19/06/2023
V0.3	D3.2	Mathieu De Coster Joni Dambre	UGent	27/06/2023
V0.2	D3.2	Jorn Rijckaert	VGTC	21/06/2023
V0.1	D3.2	Henk van den Heuvel	RU	16/06/2023
V0.1	D3.2	Myriam Vermeerbergen	KU Leuven	17/06/2023
V0.2	D3.2	Rehana Omardeen	EUD	18/06/2023
V0.2	D3.2	Mirella De Sisto Dimitar Shterionov	TiU	21/06/2023 25/06/2023

Acronyms

The following table provides definitions for acronyms and terms relevant to this document.

Acronym	Definition
SLR	Sign Language Recognition
SL	Sign Language
SLT	Sign Language Translation
CSV	Comma Separated Value
VGT	Flemish Sign Language
NGT	Dutch Sign Language
ISL	Irish Sign Language
BSL	British Sign Language
LSE	Spanish Sign Language
WP	Work Package
MT	Machine Translation
ROA	Re-train On All

Table of Contents

1. Overview

Deliverable 3.2 describes the Sign Language Recognition (SLR) models and the SLR component. The difference between model and component is the following. The models are the actual deep neural networks that predict, given input video data, the sign corresponding to the video. The component is a web service which wraps these models and is integrated into the SignON application framework (see Deliverable 2.2). With this web service, the trained models can be used to perform inference on data submitted by users of the application.

2. SLR models

In this section, the SLR models are described including the data processing, model architecture, training and inference procedures, and evaluation, for each of the five sign languages. These models are implemented in Python with the PyTorch[1] deep learning library, supported by the PyTorch Lightning[2] framework. The model training and inference code is available in the SignON GitHub organization under WP3/slr-pipeline.

2.1. Recognizing isolated signs in continuous signing

The purpose of the SLR models is to extract information from video data containing signs and to pass this information onto the Sign Language Translation (SLT) models. Within the SignON project, we train SLR models on the objective of recognizing individual signs, i.e., isolated sign recognition, extracted from continuous signing. That is, during training, every video contains a single sign. The reasoning behind this is a pragmatic choice necessitated by the limited amounts of available data (see Section 2.1.1 for specifics). However, importantly, these videos are cut from longer (manually annotated) videos containing sequences of signs with transitions and co-articulations. They introduce challenges that are often not present in isolated SLR in the scientific literature: in most cases, there the datasets contain recordings of individual signs without transitions such as in MS-ASL (Joze et al., 2018) or AUTSL (Sincan et al., 2020); these are easier to recognize. However, the presence of these transitions and co-articulations forces the models to learn features that are more representative of real-world signing. This is especially important because after training, the models are also applied to *continuous* SL input and tasked to recognize the signs they were trained on.

2.1.1. Why isolated SLR?

The SLR models are trained for the task of isolated SLR: that is, given an input video, the models are tasked to predict a single sign for that input video. The lack of annotated data is what necessitates this choice. For the corpora and datasets that are available for SignON, we are unable to extract sufficient consecutive signs to support continuous SLR (there are too many gaps between annotated signs). Hence, we start from the assumption that an isolated SLR model extracts salient SL representations (i.e., the

[1] https://pytorch.org/
[2] https://www.pytorchlightning.ai/

embeddings we describe in Section 2.1.2) from the data and that these representations can be used for downstream tasks such as SLT.

2.1.2. Why embeddings?

Labeled data is scarce in the SL processing domain. Furthermore, datasets that can be used for SLR and datasets that can be used for SLT often originate from different sources. For example, the "Content4All VRT NWS" dataset (Camgöz et al., 2021) contains data that can be used to train SLT models, but no annotations for SLR. Moreover, because SLT datasets typically contain only very few parallel utterances—the largest publicly available dataset, RWTH-PHOENIX-Weather 2014T (Camgöz et al., 2018), containing only 8257 (De Coster et al., 2023)—it is impractical to train SLR and SLT end-to-end on these data.

As a side-effect, there is little overlap between the data used for SLR and the data used for SLT within the SignON project. The overlap between the vocabularies between SLR and SLT for NGT, the language for which we have the most labeled data, is estimated at less than 5%. Hence, we would only be able to predict glosses correctly for a very small portion of the translation data, even with an SLR model that has perfect accuracy. Moreover, due to the lack of labeled data, the SLR models do not achieve near-perfect accuracy. Hence, predicting glosses as output of SLR, and using these as input for SLT, would lead to a large number of errors being propagated to the SLT model. Theoretically, should we have sufficient annotated data, we would be able to predict glosses (as the "written form of signs") with high accuracy and use these in the downstream task. However, with the currently available data this is not possible, and moreover, glosses do not contain all of the information that is present in actual signing (De Coster et al., 2023), so there would still be information loss to a certain extent.

Instead, we choose to extract the latent representations (embeddings) from intermediate layers learned by the SLR model: these embeddings are less specific than the gloss outputs and contain more generic information relating to face, body, and hand morphology and movement. This could, for example, allow an MT model to learn about common confusions between signs that resemble each other. Our hypothesis is that these embeddings provide more general information than glosses. The SLT model then operates directly on these embeddings[3].

[3] See D4.4 - Second distributional intermediate representation based on embeddings - InterL-E: https://signon-project.eu/publications/public-deliverables/

While glosses are specific to each SL and often dependent on the annotator and annotation guidelines, the embeddings are less language-specific (see Section 2.4). This allows (i) to transfer embeddings from one language to another, (ii) align SLR with SLT embeddings, and (iii) recognize previously unseen signs (De Coster and Dambre, 2023).

2.2. Data

The SLR models, being deep neural networks, require large amounts of training data. To support a single training and inference pipeline, we collect data for the considered languages and process these data in the same way to obtain datasets in a uniform format.

2.2.1. Data format

The datasets are processed to obtain individual examples: every example consists of a video and corresponding metadata. Each video is stored as a separate file in a directory and the metadata are stored in one single CSV file per dataset. The metadata consists of the following fields:

- **Id**: A unique identifier for every sample
- **Gloss**: The gloss corresponding to this sign
- **start_ms**: The start time (in milliseconds) relative to the source video from which this sample was extracted
- **end_ms**: The end time (in milliseconds) relative to the source video from which this sample was extracted
- **Participant**: The unique participant identifier, indicating which person is signing this sample
- **SourceVideo**: Path of the source video file
- **SampleVideo**: Path of the video corresponding to this sign in the processed dataset
- **subset**: Either “train”, “val”, or “test”; this value indicates to which subset of the dataset this sample belongs and how it is used for training and evaluation

Listing 1 shows an example for the VGT dataset.

```
Id,Gloss,start_ms,end_ms,Participant,SourceVideo,SampleVideo,subset
0,EERST-A,15850,16040,i002,CVGT_01/CVGT_0102_i002.mp4,CVGT_0102.eaf_0
.mp4,train
1,WG-1,16047,16216,i002,CVGT_01/CVGT_0102_i002.mp4,CVGT_0102.eaf_1.mp
4,train
4,SCHOOL-A,17957,18223,i002,CVGT_01/CVGT_0102_i002.mp4,CVGT_0102.eaf_
4.mp4,train
```

Listing 1. *Subset of the metadata file for the VGT data.*

2.2.2. Data processing

The data sources consist of SL corpora (for VGT, NGT, ISL, and BSL); we describe which corpora these are in Section 2.2.3. These corpora contain videos of continuous signing which are partly annotated with glosses, translations, and metadata. These annotations are not in a unified format (De Sisto et al., 2022) and need to be processed before they can be used to train SLR models. The processing procedure consists of several steps. It is available in the SignON GitHub organization under the WP3/SLR-Dataset-Processing repository.

First, all annotation files are collected. Only the files that contain annotations and that are linked to existing video files are kept. All annotations from all files are gathered. A set of unique glosses is collected; every gloss in this set is normalized to account for annotation differences. For example, in the VGT corpus, some glosses occur both in lowercase and uppercase variants, but they refer to the same unique sign. After normalizing the glosses, only those glosses that correspond to lexical signs are kept. For example, fingerspelling annotations are removed because fingerspelling recognition is a task separate from isolated SLR. This is done based on the gloss annotations (e.g., in the VGT coprus, fingerspelling is indicated with “VS:<letters>” and in the NGT corpus with “#<letters>”). We then remove infrequent glosses (see Table 1 for specifics), as deep neural networks require sufficient training data for every category.

After collecting all annotations according to the above criteria, a dataset split is performed. This split is stratified on gloss and grouped on participant identity. As a result, the distributions of the signs in the training, validation, and test subsets are similar, and every participant only occurs in either the training, validation, or test set, but never in two or more of these subsets. This is done to avoid data leakage.

After the split, any glosses that do not occur in all three sets are removed: if a gloss is not present in the training set, the model will not learn to predict it, and if it is not present in the validation or test set, we cannot evaluate the model's performance for this gloss.

2.2.3. Datasets

The data collection efforts have previously been discussed in Deliverable 3.1. Here, we repeat information that is relevant for this deliverable: i.e., how these data were processed and used to train SLR models. Table 1 summarizes some key information for the different datasets.

***Table 1**. Dataset statistics.*

Dataset	Number of samples	Number of distinct glosses	Number of participants	Minimal number of samples per gloss
VGT	24967	292	112	20
NGT	68854	458	82	20
ISL	4013	224	37	3
BSL	2416	123	44	3
LSE	N/A (see below)	N/A (see below)	N/A (see below)	N/A (see below)

VGT

We use the VGT corpus[4] as the data source to create our VGT dataset. Using the procedure outlined in Section 2.2.2, we collect 24,967 data samples corresponding to 292 unique glosses, from 112 unique participants. Every gloss has at least 20 occurrences across the entire dataset. The dataset features a large class imbalance (see Figure 1), with more than 17% of the samples belonging to the two majority classes. A baseline for the performance of an SLR model on this dataset is the "majority class prediction baseline", where the majority class is predicted for any model input. The accuracy of this baseline is 10.39%.

[4] https://corpusvgt.be/

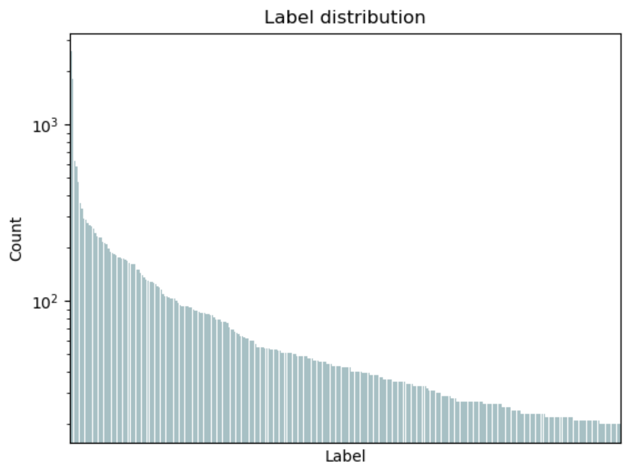

Figure 1. *Label distribution in the VGT dataset.*

The examples are distributed more evenly with respect to the participants, though there are some participants with very few samples (minimum: 1): see Figure 2.

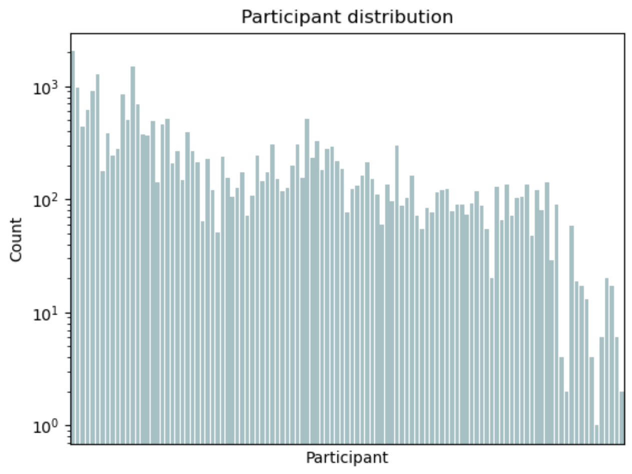

Figure 2. *Participant distribution in the VGT dataset.*

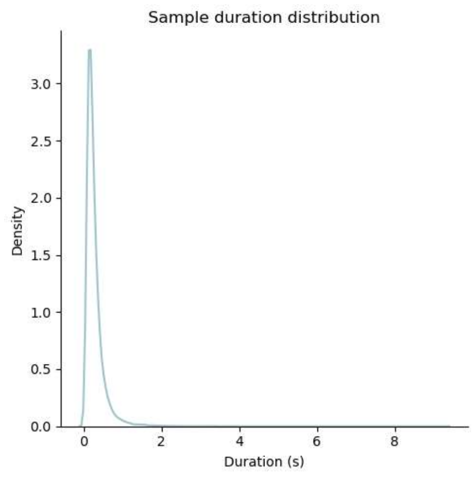

Figure 3. *Sample duration distribution in the VGT dataset.*

Because the samples are collected by segmenting videos with continuous signing, many samples are very short: see Figure 3. The mean duration is 0.38 seconds (10 frames) and the minimum duration is 0.012 seconds (1 frame). Some outlier samples have a duration of 9 seconds (225 frames): these are samples where a sign is being held for a long period of time or repeated multiple times.

NGT

We use the NGT corpus[5] as the data source to create our NGT dataset. Using the procedure outlined in Section 2.2.2, we collect 68,854 data samples corresponding to 458 unique glosses, from 82 unique participants. Every gloss has at least 20 occurrences across the entire dataset. Like the VGT dataset, the NGT dataset features a large class imbalance (see Figure 4). The majority class prediction baseline for the performance of an SLR model on this dataset is 12.76%. The participant distribution is shown in Figure 5, and the distribution of the duration of samples in Figure 6. The minimum duration is 0.001 seconds (1 frame), and the maximum duration is 17.92 seconds (448 frames). Like in the VGT dataset, these are samples where a sign is being held for a long period of time or repeated multiple times. The mean duration is 0.32 seconds (8 frames).

[5] https://www.corpusngt.nl/

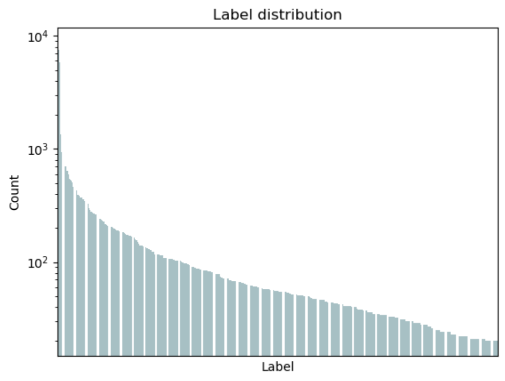

Figure 4. *Label distribution in the NGT dataset.*

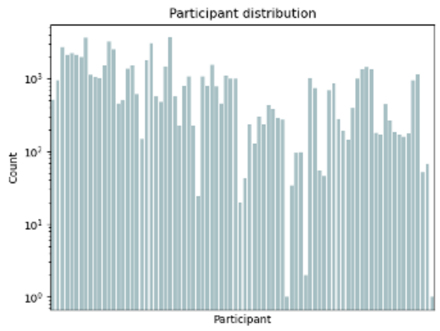

Figure 5. *Participant distribution in the NGT dataset.*

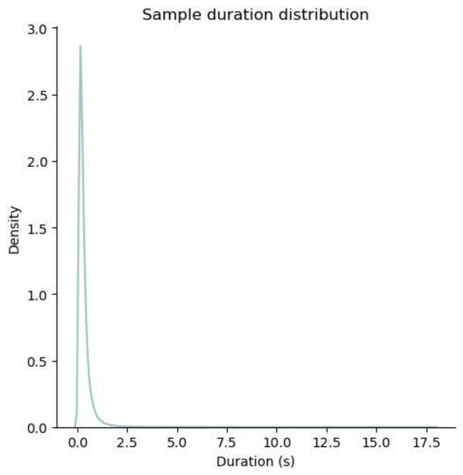

Figure 6. *Sample duration distribution in the NGT dataset.*

ISL

We use the Signs of Ireland Corpus[6] as the source to create our ISL dataset. Using the procedure outlined in Section 2.2.2, we collect 4,013 data samples corresponding to 224 unique glosses from 37 unique participants. In this case, each gloss has a minimum of only 3 occurrences across the entire dataset. It was proposed that this lower bound be set to 20 occurrences as with VGT/NGT, however this would have limited the number of trainable classes to just 50. As with the VGT and NGT datasets, the ISL dataset features a large class imbalance (see Figure 7). The majority class prediction baseline for the performance of an SLR model on this dataset is 11.06%. The participant distribution is shown in Figure 8, and the distribution of the duration of samples in Figure 9. The minimum duration is 0.04 seconds (1 frame), and the maximum duration is 3.2 seconds (80 frames). The mean duration is 0.70 seconds (18 frames).

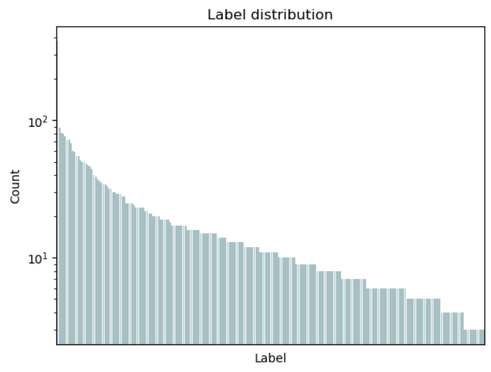

Figure 7. *Label distribution in the ISL dataset.*

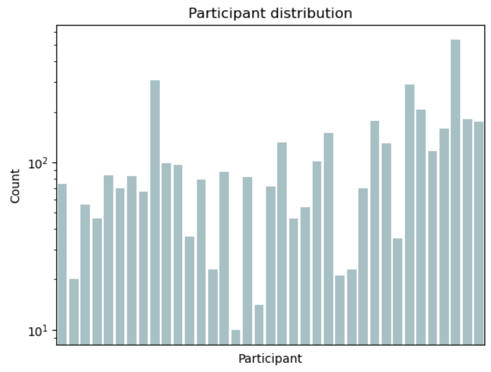

Figure 8. *Participant distribution in the ISL dataset.*

[6] http://www.tara.tcd.ie/bitstream/handle/2262/1597/ITT?sequence=1

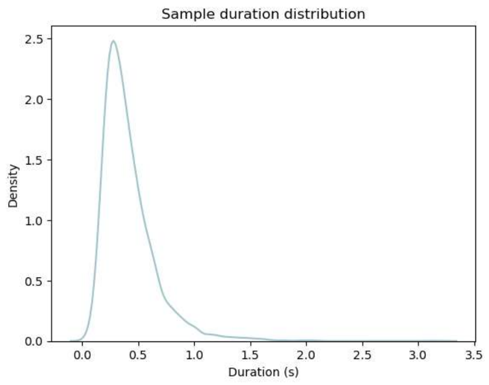

Figure 9. *Sample duration distribution in the ISL dataset.*

BSL

We use the BSL Corpus Project[7] as the source to create our BSL dataset. Specifically, the narrative activity is used in which signers tell a personal story. Using the procedure outlined in Section 2.2.2, we collect 2,416 data samples corresponding to 123 unique glosses from 44 unique participants. In this case, each gloss has a minimum of only 3 occurrences across the entire dataset. It was proposed that this lower bound be set to 20 occurrences as with VGT/NGT, however this would have limited the number of trainable classes to just 21. As with all other datasets, the BSL dataset features a large class imbalance (see Figure 10). The majority class prediction baseline for the performance of an SLR model on this dataset is 13.8%. The participant distribution is shown in Figure 11, and the distribution of the duration of samples in Figure 12. The minimum duration is 0.04 seconds (1 frame), and the maximum duration is 4.8 seconds (120 frames). The mean duration is 0.41 seconds (10 frames).

[7] https://bslcorpusproject.org/

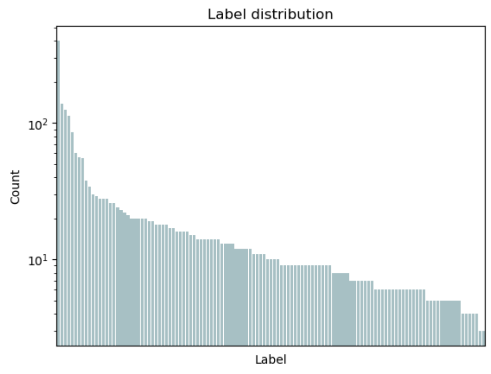

Figure 10. *Label distribution in the BSL dataset.*

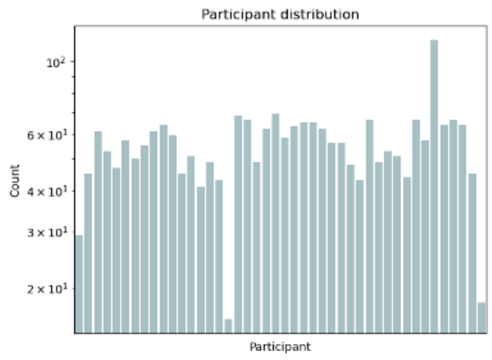

Figure 11. *Participant distribution in the BSL dataset.*

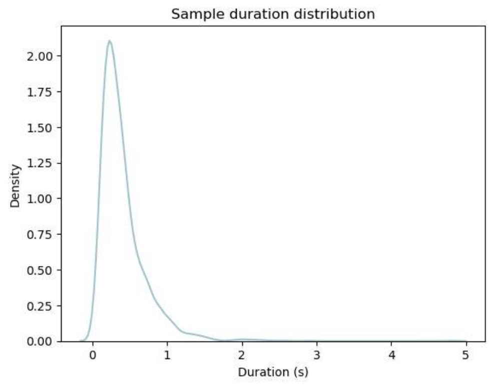

Figure 12. *Sample duration distribution in the BSL dataset.*

LSE

Out of the datasets collected in the SignON project for LSE, there were no datasets which contained parallel gloss annotations for videos with sufficient examples per video to train an isolated SLR model. Hence, the proposed pipeline of training isolated SLR models could not be applied for LSE. However, we still support this language in the SignON application through other means. We describe in Section 2.4.5 and Section 3 how LSE inputs are processed in the pipeline and the SignON application.

2.3. Pipeline

2.3.1. SLR pipeline architecture

The SLR pipeline is implemented as a Python software package. It can be used to train SLR models, to evaluate them, and to perform inference with them. Inference can be performed in an online fashion for use in the SLR component (see Section 3) or in an offline fashion to process batches of videos for training SLT models. The SLR pipeline is deterministic and can handle arbitrary SL video datasets, as long as they are converted to the format described in Section 2.2.1.

Listing 2 shows an example invocation of the pipeline to train a model on the VGT dataset. The hyperparameters of the model are set with command line arguments and are stored along with model checkpoints. Training progress can be logged to either Tensorboard[8] or Weights and Biases[9].

```
python -m train --run_name vgt_scratch --gpus 1
--variable_length_sequences --batch_size 128 --log_dir /logs
--model_name PoseFormer --learning_rate 0.0003 --num_attention_layers
4 --num_attention_heads 8 --d_pose 134 --source_aspect_ratio 0.5652
--d_hidden 192 --num_classes 292 --data_kind Mediapipe --num_workers
4 --data_dir /data/corpusvgt
```

Listing 2. *Invocation of the training command for the SLR pipeline.*

[8] https://www.tensorflow.org/tensorboard
[9] https://wandb.ai/

Once this model is trained, it can be evaluated on all available data using the command shown in Listing 3. Only the checkpoint path and an output file need to be provided. This command will write predictions for all files to the given CSV file.

```
python -m test /logs/vgt_scratch/model_best.ckpt
/predictions/vgt_scratch.csv
```

Listing 3. *Invocation of the evaluation command for the SLR pipeline.*

The same model can also be used for offline inference, i.e., to extract embeddings for one or more videos. The command for this is shown in Listing 4.

```
python -m predict /logs/vgt_scratch/model_best.ckpt /data/new_videos
/data/model_predictions --embedding_kind spatial
```

Listing 4. *Invocation of the inference command for the SLR pipeline.*

To perform online inference, the pipeline can be used as a library in a larger Python project (see Section 3).

The overall structure of the SLR pipeline is illustrated in Figure 13. First, the video data are processed using MediaPipe Holistic (an external software package) to extract keypoints. Keypoint cleaning is performed as a form of data standardization (see Section 2.3.2). These cleaned keypoints form the input to the model (see Section 2.3.1). The model is trained to predict glosses, but during inference embeddings are extracted.

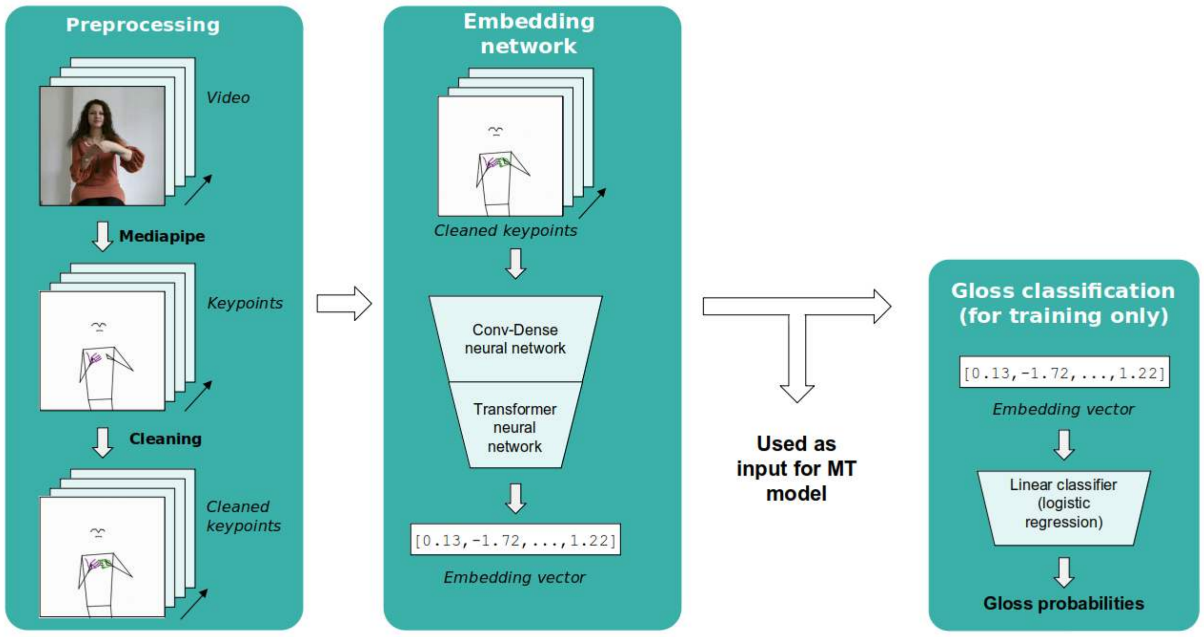

Figure 13. *The SLR pipeline.*

2.3.1. Model architecture

The model architecture, which is detailed below, was obtained by tuning layers and making architectural choices based on the properties of the data and on intermediate results, to improve the accuracy score on the validation subsets of the used SLR datasets.

The architecture—illustrated in Figure 14—consists of five stages, which together form a deep neural network that is optimized end-to-end. In the first stage, residual depthwise 1D convolutions learn to recognize local temporal patterns for every input feature individually. In the second stage, an embedding is learned for every frame in the input sequence independently, learning non-linear relationships between individual keypoint coordinates. The third stage processes the resulting frame embeddings temporally with a limited receptive field using a stack of residual depthwise 1D convolutions, detecting local temporal patterns within the frame embedding sequence. The fourth stage learns global temporal information using self-attention: the receptive field is the entire sequence. Self-attention uses so-called “CLS pooling” (Devlin et al., 2018): a single vector summarizes the entire sequence. Finally, this vector is used as the input to a classification layer.

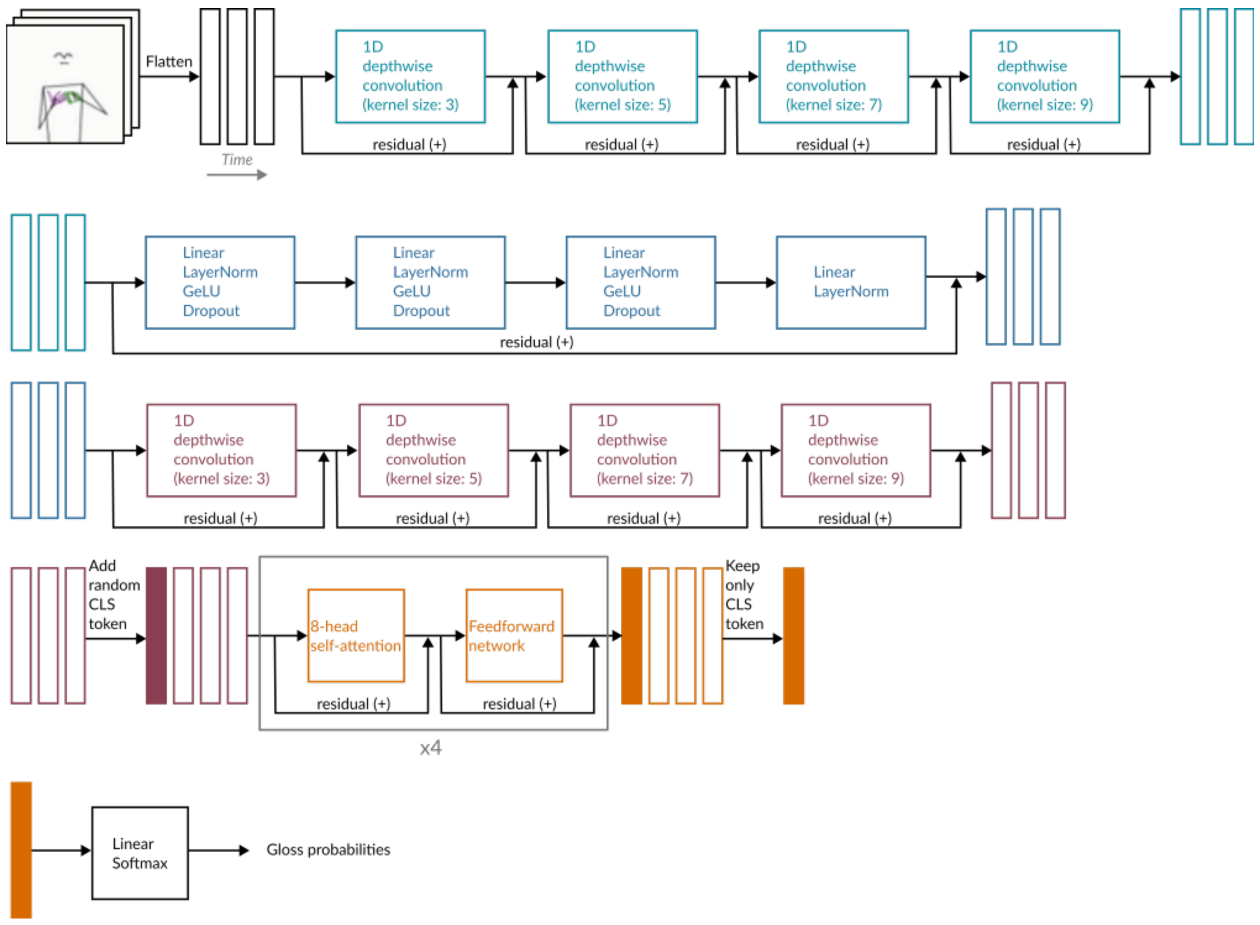
Flatten
Time
1D depthwise convolution (kernel size: 3)
1D depthwise convolution (kernel size: 5)
1D depthwise convolution (kernel size: 7)
1D depthwise convolution (kernel size: 9)
residual (+)
Linear LayerNorm GeLU Dropout
Linear LayerNorm
Add random CLS token
8-head self-attention
Feedforward network
x4
Keep only CLS token
Linear Softmax
Gloss probabilities

Figure 14. *The SLR model architecture, divided into the five stages. Every row corresponds to a single stage.*

2.3.2. Model input and output requirements

The SLR models use keypoints extracted with MediaPipe Holistic[10] as input. This pose estimator was chosen because it supports full body pose estimation in almost real-time. The keypoints corresponding to the upper body and the hands are used. Experiments were performed with the addition of face keypoints (more specifically a subset of the lip and eyebrow keypoints, which were found useful in the classification of signs in a recent SLR Kaggle competition[11]), but these did not yield satisfactory results. Because MediaPipe Holistic fails to predict keypoints in some cases, e.g., when a hand could not be detected, temporal imputation is performed. Temporal imputation means the replacement of missing values by leveraging the fact that these missing values occur in a sequence. In case one or more frames

[10] https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md
[11] https://www.kaggle.com/competitions/asl-signs

are missing, linear interpolation is used to impute these. We search for the nearest previous and subsequent non-missing frames, and perform linear interpolation to fill in the blanks.

We furthermore perform a shift to the center of the chest and a scaling to the distance between the shoulders as a form of data normalization (which is essential for the performance of machine learning models). In this way, we account for camera position and distance. The hands and body are processed separately. These normalized and imputed keypoints form the input to the model. The effects of normalization are illustrated in Figure 15.

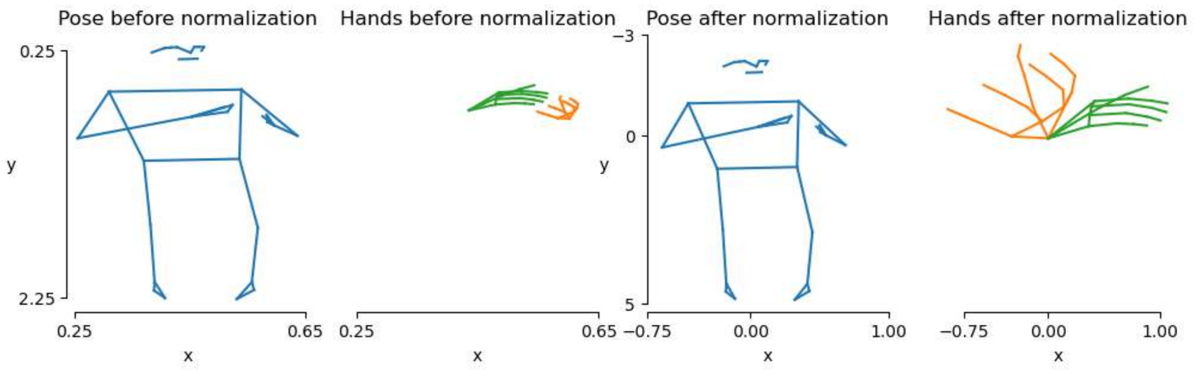

Figure 15. *The normalization procedure changes the scale and values of features, a crucial step before training machine learning models.*

The models are trained to predict glosses, and hence the outputs of the models are *K* values, where *K* is the number of glosses in each of the datasets. However, for the SLR component and for training the SLT models, we use internal embeddings from our model (see Section 2.1.2). The SLR models can provide spatial, temporal, and class embeddings. Spatial embeddings are the output of the frame embedding block of the network (see Section 2.3.1). One embedding is returned per frame. Temporal embeddings are the output of the sequence embedding block of the network. At this point, the network has processed the entire input sequence. One embedding is returned per frame. Class embeddings are extracted from the same block as temporal embeddings, but summarize the entire input sequence into one vector. In the current SLR models, 192-dimensional spatial embeddings are used. This is visualized in Figure 16.

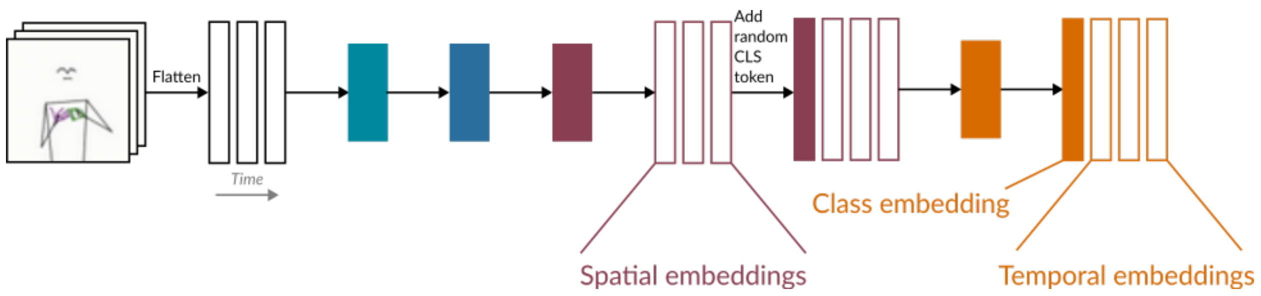

Figure 16. *The spatial, temporal, and class embeddings are extracted from different parts of the model. Refer to Figure 14 for the complete model architecture and the contents of the constituent blocks.*

2.3.3. Training procedure

The SLR models are sign classification models[12], so they are trained to minimize the categorical cross-entropy loss and to maximize classification accuracy. The models are trained with early stopping. Several hyperparameters are tuned to obtain optimal performance with the given data and model architecture. Notable hyperparameters include:

- Learning rate
- Learning rate schedule
- Embedding size
- Number of self-attention layers
- Number of self-attention heads
- Label smoothing
- Weight decay

The optimal hyperparameter values for every model are stored along with their checkpoints in the GitHub repository under WP3/slr-pipeline.

Data augmentation is used to mitigate the impact of the lack of data. We perform the following augmentations after extracting keypoints with MediaPipe and cleaning them:

- Randomly shifting the hand keypoints across time with a small offset
- Randomly flipping the keypoints horizontally
- Randomly rotating the hands independently by small angles
- Introducing random noise to the keypoints

[12] Typically, we refer to such models as "classifiers." To avoid confusion with the linguistic term "classifier handshape," we will continue to use "classification models" in this document.

- Random temporal cropping of subsequences
- Randomly removing hands from the frames to simulate missing keypoints

These augmentations were evaluated individually (by recording their impact on the validation set accuracy) and their hyperparameters were tuned to obtain optimal results.

Once optimal hyperparameters have been found, we fix them and we retrain the models on the training and validation set data. The test set accuracy is then the final measure we report to evaluate our models by.

To further reduce the impact of the lack of data, we also investigated knowledge transfer and knowledge sharing between languages. We hypothesize that certain low level features are shared between languages (e.g., hand shapes, movements, body poses…). Knowledge transfer is implemented in the form of transfer learning: we first train an SLR model on one dataset and then transfer (part of) it to another dataset. In this case, we transfer the entire model except for the final classification layer. We first freeze all transferred weights and train only the new, randomly initialized, classification layer. The performance at this point is an indication of how well the embeddings map to the new dataset and to the different language. When the loss has converged, we fine-tune the model until the early stopping criterion is met (i.e., the validation loss has not reduced for 20 epochs). At this point, we have properly tuned the downstream model and can look at the benefits of transfer learning.

Knowledge sharing is implemented as multilingual training: we concatenate multiple datasets and train a joint classification model. To avoid confusions between signs that are similar but from different languages, we furthermore learn a small language embedding and give this as additional input to the classification model. We obtain better results with transfer learning than with multilingual training, so we only report on the former.

2.4. Results

We report the accuracy for every language on the training, validation, and test set. We only report the current best scores for every dataset. We also report the test set accuracy when we Re-train On All (ROA) training and validation data using the best hyperparameters found on the validation set.

2.4.1. VGT

The VGT model was first trained on NGT, after which the weights were transferred to VGT and fine-tuned. The accuracy scores are given in Table 2.

Table 2. *Accuracy scores for VGT.*

	Training set	**Validation set**	**Test set**	**Test set (ROA)**
No transfer	75.25%	50.07%	47.70%	49.07%
NGT transfer (no fine-tuning)	56.60%	49.37%	48.23%	48.40%
NGT transfer (fine-tuning)	76.76%	52.37%	50.80%	48.83%

2.4.2. NGT

The NGT model was trained from scratch, i.e., without pre-training on another language. The accuracy scores are given in Table 3.

Table 3. *Accuracy scores for NGT.*

	Training set	**Validation set**	**Test set**	**Test set (ROA)**
No transfer	70.88%	47.26%	51.91%	52.66%

2.4.3. ISL

The ISL model was first trained on NGT, after which the weights were transferred to ISL and fine-tuned. The accuracy scores are given in Table 4.

Table 4. *Accuracy scores for ISL.*

	Training set	**Validation set**	**Test set**	**Test set (ROA)**
No transfer	81.78%	25.98%	22.13%	26.27%
NGT transfer (no fine-tuning)	64.86%	28.21%	25.85%	26.27%
NGT transfer (fine-tuning)	74.12%	30.10%	30.07%	27.45%

2.4.4. BSL

The BSL model was first trained on NGT, after which the weights were transferred to BSL and fine-tuned. The accuracy scores are given in Table 5.

Table 5. *Accuracy scores for BSL.*

	Training set	**Validation set**	**Test set**	**Test set (ROA)**
No transfer	32.61%	24.42%	18.76%	23.55%
NGT transfer (no fine-tuning)	52.82%	30.69%	25.35%	31.54%
NGT transfer (fine-tuning)	65.21%	32.78%	28.14%	31.54%

2.4.5. LSE

Because no separate models were trained for LSE due to a lack of data, we cannot report any accuracy figures for this language. However, we use the frozen NGT model to extract embeddings for LSE in the SLR component (see Section 3). As can be seen from Tables 2, 4, and 5, using frozen embeddings yields similar or better accuracy (depending on how much data is available) as training from scratch. Therefore, we can assume that the extracted features will be useful for LSE, too.

3. SLR component

The SLR component wraps the trained models in a web service that is integrated into the SignON framework. This web service allows for online inference using the models as part of the sign-to-text and sign-to-sign translation pipelines that SignON supports. The web service is implemented in Python using the Flask[13] framework. It is Dockerized[14] to allow for easy local and remote development and deployment. The code can be found in the SignON GitHub organization under WP3/slr-component.

The SLR component web service has a single endpoint which supports POST requests. This endpoint receives a video file as a blob[15] along with metadata containing the source language of this video (“VGT”, “NGT”, “BSL”, “ISL”, or “LSE”). This metadata is used to forward the video to the correct SLR model. Note that for LSE, the NGT model is used to extract embeddings (see Sections 2.2.3 and 2.4.5).

First, the video is processed to extract keypoint information with MediaPipe. These keypoints are imputed and normalized in the same way as during model training. The keypoints are passed to the SLR model, which extracts the SL representation (embeddings) and returns it as a JSON list of numbers.

The SLR component can be updated with new models by uploading the model checkpoint file to the correct directory on the server hosting the component, and updating the references to the checkpoint files. If architectural changes are made, then the included SLR code must also be updated.

The SLR component itself is called from the WP3 dispatcher and also returns the response to this dispatcher. The request procedure is shown in Figure 17 and the request handling inside the SLR component is shown in Figure 18. For further details about the SignON framework architecture, please refer to Figure 1 of D2.4[16].

[13] https://flask.palletsprojects.com/
[14] https://www.docker.com/
[15] Blobs are raw data that can be sent along with HTTP requests. For more information, see the Mozilla Web documentation: https://developer.mozilla.org/en-US/docs/Web/API/Blob.
[16] D2.4 - Intermediate release of the Open SignON Framework: https://signon-project.eu/publications/public-deliverables/

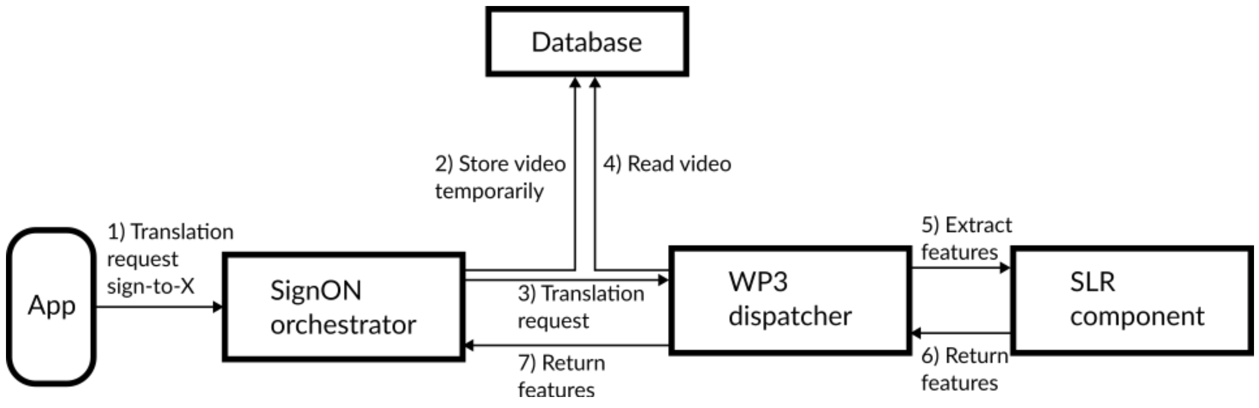

Figure 17. *How the SLR component is called from the application. After step 7, the WP3 dispatcher forwards the translation request with the SLR features to the WP4 dispatcher, which performs translation, and forwards its results to the WP5 dispatcher, which formats the output message. Communication between dispatchers is mediated by a RabbitMQ message broker.*

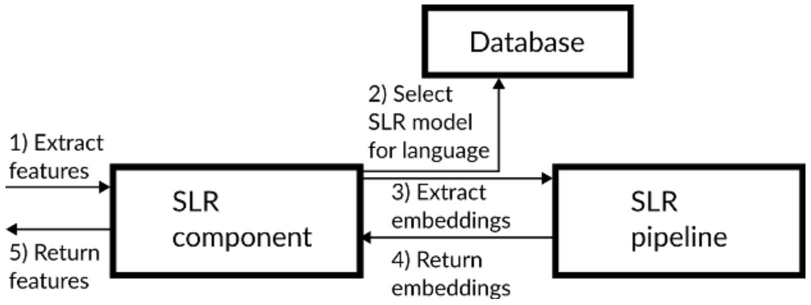

Figure 18. *Processing of the request inside the SLR component. The component forwards the request to the SLR pipeline (see Section 3.2) after selecting the SLR model that corresponds to the source message language. The SLR pipeline extracts embeddings and returns those to the SLR component.*

4. Conclusion and future work

This document describes the SLR models, the pipeline with which they are trained, evaluated, and used for inference, and the SLR component which integrates these models into the SignON application framework. The datasets which are used for the five sign languages supported in SignON, i.e., VGT, NGT, BSL, ISL, and LSE, are also described.

The proposed SLR pipeline can be used to train new models (whether it is for new SLs or for already supported ones) or to update (re-train) the existing models with additional data (when such data is collected and processed).

In the remainder of the project, we will investigate fingerspelling recognition techniques and continuous sign language recognition. In both cases, the lack of labeled data for the sign languages in the SignON project limits the immediate applicability of the techniques that will be developed. However, when more data are collected in the future, these models will be able to be integrated into the SignON framework through the SLR pipeline and component.

Furthermore, we will investigate more robust pose estimation techniques tuned specifically for the challenging nature of SL data. Improving the pose estimation techniques is expected to have an immediate impact on the SLR performance.

Once sufficient data is collected for LSE, we will be able to apply the SLR models to these data to train language-specific models instead of using the embeddings learned from NGT as we are currently doing. This will be possible, because we have illustrated and discussed in this deliverable that the same model can be applied to different SLs, even in very low-resource situations, such as ISL and BSL.

References

Camgöz, N. C., Hadfield, S., Koller, O., Ney, H., & Bowden, R. (2018). Neural sign language translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7784-7793).

Camgöz, N. C., Saunders, B., Rochette, G., Giovanelli, M., Inches, G., Nachtrab-Ribback, R., & Bowden, R. (2021, December). Content4all open research sign language translation datasets. In *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)* (pp. 1-5). IEEE.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

De Coster, M., & Dambre J. (2023). Querying a Sign Language Dictionary with Videos using Dense Vector Search. In *SLTAT2023, the Eighth International Workshop on Sign Language Translation and Avatar Technology*.

De Coster, M., Shterionov, D., Van Herreweghe, M., & Dambre, J. (2023). Machine translation from signed to spoken languages: State of the art and challenges. *Universal Access in the Information Society*, 1-27.

De Sisto, M., Vandeghinste, V., Gómez, S. E., De Coster, M., Shterionov, D., & Saggion, H. (2022). Challenges with sign language datasets for sign language recognition and translation. In *LREC2022, the 13th International Conference on Language Resources and Evaluation* (pp. 2478-2487).

Joze, H. R. V., & Koller, O. (2018). MS-ASL: A large-scale data set and benchmark for understanding american sign language. *arXiv preprint arXiv:1812.01053*.

Sincan, O. M., & Keles, H. Y. (2020). AUTSL: A large scale multi-modal turkish sign language dataset and baseline methods. *IEEE Access*, *8*, 181340-181355.