Signs project - report

Yohan Pellerin, Philip Rettig, Romain Darous

March 13, 2024

Abstract

The goal of this project was, given videos in which signs of the Swedish sign language were performed, to be able to classify them. By classifying, we mean, given a video input, being able to detect which sign is performed. More precisely, we wanted to test a specific representation by trying to classify the signs using the frame per frame pose extraction (hand coordinates). We wanted to check is the representation using hand coordinates was a good representation of the signs. Thus, we wanted to perform a classification task using those to see if it captures well the underlying sign.

The provided data is a set of videos corresponding, either to an isolated sign (token), or to a full phrase in sign language linked with its meaning. For this project, we restricted it to classifying tokens only. To each video was attached a .JSON file with the frame per frame position extraction. As videos can be seen as time series of images, we naturally drove our research toward recurrent neural network models. We tried two different models, one simple model, and a deeper one made of several LSTM layers.

The main problem of our dataset was the too high number of different signs and the too small amount videos per sign, which makes impossible to directly perform sign classification. Thus, as a first task, we grouped the significant amount of tokens that we had in categories, and we tried to perform multi-class classification among those categories. The results were a bit disappointing, so we tried another approach.

Instead of grouping all the signs and videos in categories, we decided to restrict to the ten signs for which we have the biggest amount of videos and train a model to classify videos to recognize those signs. We could in then end get more than 90% of accuracy on the training set and 45% on the test set. The model fails to generalize, and one of the main limitations is the lack of enough data per sign. We still could get a classifier that can train and perform better than just random classification.

1 Introduction

Sign language serves as a fundamental mode of communication for the Deaf and hard-of-hearing communities. It encompasses a rich array of gestures, facial expressions, and body movements, each with nuanced meanings that vary significantly across cultures and regions. This visual language empowers individuals to articulate thoughts, emotions, and concepts, promoting inclusivity and mutual understanding both within and beyond the Deaf community. A long-term aspiration in this field is to develop systems capable of automatically generating signs. Achieving this goal hinges on obtaining accurate representations of signs, which serve as the foundation for generating them.

In this paper, our focus lies in evaluating a particular representation of sign language. Our approach involves utilizing this representation to classify various signs into distinct groups using recurrent neural networks. By doing so, we aim to assess the effectiveness of the representation in capturing the diversity and nuances of sign language expressions. This evaluation is crucial for advancing the development of automated sign language generation systems.

2 Background

2.1 Sign language - An international handbook

Phonetic studies of sign languages typically focus on articulation, considering the arms, hands, and fingers as complex articulators that exhibit a significant range of variation. In the "Handbook of Sign Language Linguistics" [PSW12] describes the motion in sign language as being closely linked to how signs are made, focusing particularly on the movement and positioning of hands and fingers. Sign language motion is all about the actions of the upper limbs, from the shoulders right down to the fingertips. It highlights the complex capabilities of our arms, hands, and fingers, which allow for a wide variety of movements, beyond just the basics of hand shape, movement, and where signs are made in space.

2.2 Machine learning related article

Video pose estimation consists in capturing the dynamics of human movement using sequences of frames. The research paper "LSTM Pose Machines" [YL17] introduces a new approach by incorporating Long Short-Term Memory (LSTM) units between video frames. This architecture not only learns the temporal dependencies among frames but also enhances the stability of joint predictions on moving bodies. LSTM Pose Machines excel are better in capturing the nuances of human motion, offering a significant increase in performance and speed compared to existing methods. The study provides insights into the internal dynamics of LSTM memory cells, showing how these units elevate video pose estimation performance.

In the paper "Beyond Short Snippets: Deep Networks for Video Classification" [JS16], the focus shifts towards applying deep neural network architectures to process video data over longer time periods than previously explored. The study focus on performing a video classification task, emphasizing the importance of capturing temporal relationships and global video information. The paper explores convolutional temporal feature pooling architectures and recurrent neural networks, particularly Long Short-Term Memory (LSTM) cells. The work focuses on the challenges specific to video data processing, highlighting the intricate interplay of frames, motion, and temporal context in video classification tasks.

3 Method

We've been given a dataset consisting of 27636 json files of extracted features of sign language phrases and signs. We've also been given a dataset with corresponding descriptions to the signs. The goal of this project evaluate the extracted features signs dataset. We do this by creating various classification models to determine sign language features given the extracted features (the json files). There are many different words and we need to group signs together to get a reasonable amount of classes. As a tool for this we prompt GPT-4 to label the data with the sign language features identified in the signs description. in two parts: Data collection and model selection

3.1 Data Pre-processing

To begin classifying the data using the RNN method, we first needed to prepare the data by flattening it, ensuring it was suitable for training. Originally, the data was stored in a zip file and organized using dictionaries. Each video comprises a variable number of frames, with each frame containing three types of information: hands, faces, and body. For our purposes, we opted to utilize only the hand data. Specifically, each hand consists of 21 points represented by two coordinates. Consequently, when flattened, each frame yields 84 floats. Moreover, to standardize the input for the neural network, we needed a fixed number of frames. Since we exclusively utilized single-sign videos, the frame count ranged between 40 and 400, with a predominant average around 100 frames. Thus, we made the decision to standardize each video to 100 frames. In cases where a video contained fewer than 100 frames, we employed padding techniques, adding zeros to complete the sequence.

As a result, each input video was represented as a matrix with dimensions (100, 84).

3.2 Data labeling

The book points out that the way signs move can vary a lot. This motion isn't just about following set rules for how to shape the hand or where to move it but is also influenced by the body's physical limits and how signs flow together in conversation. For instance, the height at which a sign is made can change depending on what signs come before or after it. This can make the hands move higher or lower in space, showing how sign language motion is flexible and adaptive. Moreover, the motion of a single sign can involve different parts of the arm and hand, depending on factors like the distance between the people communicating. This shows that what might seem like a straightforward sign can actually have several ways to be expressed, all depending on the situation and the signer's intentions.

With this in mind we create our first output vector. Consisting of the following features

- låg positionering
- mellan positionering
- hög positionering
- handflator orienterade inåt
- handflator orienterade utåt
- tum position inåt
- tum position utåt
- fingrar samman
- fingrar isär
- fingrar rakade
- fingrar böjda
- handform krok
- handform platt
- handform kupad
- handform fist
- rörelse oscillerande
- rörelse rotatorisk
- rörelse rak linje
- rörelse böjd linje
- handled vriden inåt
- handled vriden utåt
- handled rörelse uppåt
- handled rörelse nedåt
- handled rörelse sidleds
- fingertoppar rör vid varandra
- fingrar korsade
- handflator rör vid varandra
- handrygg mot handrygg
- flata händer mot varandra
- en hand över den andra
- fingrar snurrar runt varandra
- tummen pekar uppåt
- tummen pekar nedåt
- fingerpek mot handflata

- handflator pekar mot betraktaren
- handflator vända från betraktaren
- fingrar formar en bokstav
- fingrar formar en siffra
- handen gör en knackande rörelse

We subsequently requested GPT-4 to annotate each sign in the dataset, identifying the most relevant features from the provided list based on keywords in the given descriptions. The initial rows are displayed below. This task represented the initial phase of our data labeling process. Our objective for grouping variables in this manner was to minimize the number of potential classes for prediction, as the dataset encompassed a wide variety of signs, which would otherwise result in a significantly large output vector dimension. This describes the general process, in the discussion we motivate our variable groupings and report insights gained from various iterations.

names	word	description	phonetics
skolkurator-12712-tecken.json	skolkurator	Flata händer, uppåtriktade och inåtvända, upprepat tag.	hög_positionering, handflator_orienterade_inåt
skolkurator-12712-tecken.json	skolkurator	Flata händer, uppåtriktade och inåtvända, upprepat tag.	hög_positionering, handflator_orienterade_inåt

Figure 1: First two rows of dataset in iteration 1

3.3 Model selection

The interesting part of the background articles concerns the type model that is used to perform classification, after performing feature extraction. The pose extraction of the videos that we already have allows us to skip the representation learning part (done with CNNs in one of the articles, for instance). Given the information found in the research papers we had, we decided to try two different model architectures, a simple one with a single layer RNN, and a deeper one, with several LSTM units layers. Here are the more detailed descriptions of the two models :

3.3.1 Simple RNN using PyTorch

- **Input Representation**:
  - The input to the model is a sequence of hand coordinates representing sign language . Each key point of the hand is represented by a pair of x and y coordinates, resulting in a total of 84 input values (21 x-coordinates + 21 y-coordinates).
- **Architecture**:
  - The model architecture consists of two main components:
    - **RNN (Recurrent Neural Network)**: The input sequence of coordinates is fed into an RNN layer with 64 units.
    - **Dense Layer**: After processing the input sequence through the RNN, the hidden state of the last time step is passed through a fully connected (dense) layer. This layer transforms the hidden representation into the output space.
- **Output**:
  - The output of the model is a sequence of predictions, each corresponding to one of the labels (gestures) in the sign language vocabulary. Since it's a multi-label classification task (each gesture can be present or absent independently), the model applies a sigmoid function to the output of the dense layer. This allows each output to be interpreted as the probability of the corresponding label being present in the input sequence.

3.3.2 Deeper RNN using keras LSTM units

- **Input Representation**:
  - The input data is represented as a sequence of feature vectors. Each sequence has a shape of $(nb\_frames, frame\_features\_nb)$, where nb_frames represents the number of frames in the sequence, and $frame\_features\_nb$ represents the number of features in each frame.
- **Architecture**:
  - The model architecture consists mainly of LSTM units.
  - The model has several layers:
    - **LSTM Layers**: Three LSTM layers are stacked on top of each other. Each LSTM layer has a different number of units: 64, 128, and 64 respectively. The activation function used in each LSTM cell is the ReLU activation function.
    - **Dense Layers**: After the LSTM layers, the model includes two fully connected dense layers with ReLU activation functions.
    - **Output Layer**: The final dense layer has a sigmoid activation function and its size matches the number of classes in the output. This layer produces the probability distribution over the output classes, as for the previous model.

For both models, we use the same loss function and optimizer :

Loss Function and Optimization:

- For training the model, Binary Cross-Entropy Loss (*BCELoss*) is used. This loss function is commonly used for binary classification tasks and is suitable for multi-label classification by treating each label independently.
- The optimizer used is Adam, a popular choice for training neural networks.

4 Discussion

4.1 Data labeling process

4.1.1 Data labeling iteration 1 - Multi-label classification

The initial classes employed were those delineated in section 3.2 of the output vector. The available classes for labeling predominantly emphasized the shape of the hand. During the labeling process, we encountered several uncontrollable obstacles, such as hallucinations. Furthermore, the distribution of assigned labels exhibited a skewed pattern, resulting in an imbalanced dataset. This is most likely due to the fact that some hand shapes fits the description of various different signs. The dataset had the following distribution.

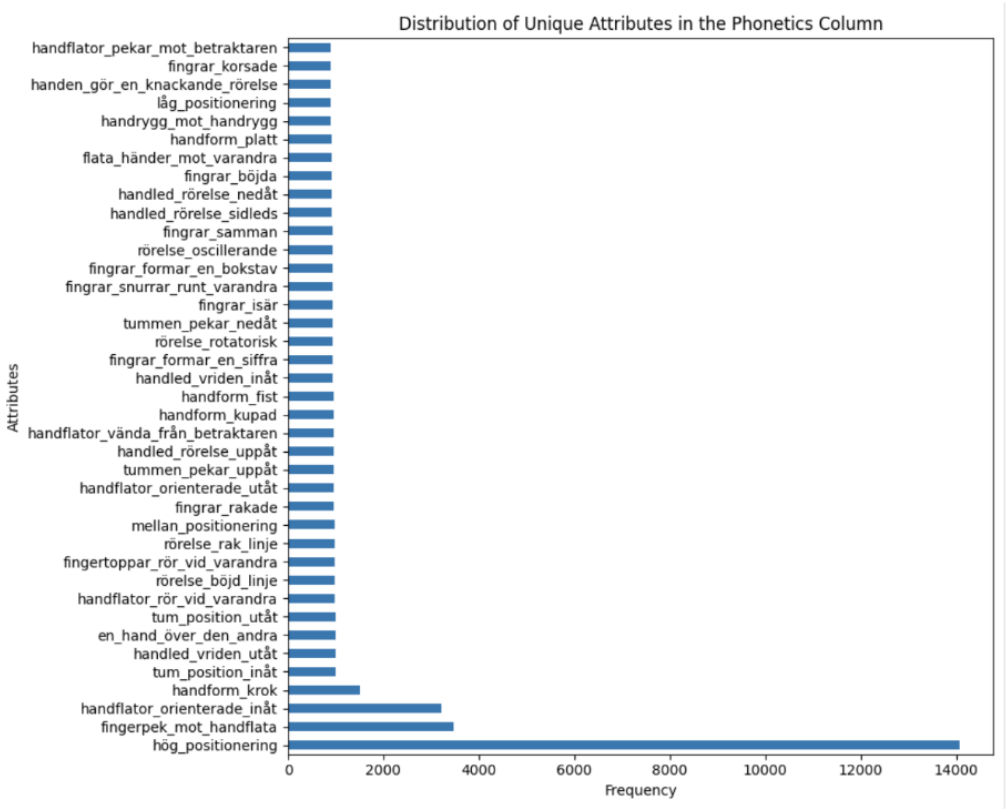

Figure 2: First two rows of dataset in iteration 1

We experimented with various groupings, yet the datasets invariably remained imbalanced. Efforts to instruct GPT-4 to constrain the variance in the frequency distribution resulted in the model attributing equal labels to all groupings on numerous occasions, which significantly impacted the relevance of the labeling process.

4.1.2 Model results iteration 1

Initially, we sought to train the model with a small dataset to verify the efficacy of the process, considering the anticipated time required for training. Subsequently, we expanded the training dataset to in-

clude 1000 inputs. However, upon applying the model, it consistently predicted only the "hög_positionering" class, which appeared to be over represented in the dataset.

One plausible explanation for this phenomenon could be the distribution of attributes. As illustrated in Figure 4, three classes exhibited significantly higher prevalence compared to others. This skewed distribution might have biased the models towards favoring these classes as predictions.

To address this issue, we opted to exclude these three over represented classes from consideration, resulting in a more balanced distribution. However, upon evaluating the model, we observed that no class had a probability exceeding 0.5. Furthermore, the most probable classes consistently remained the same two, indicating potential limitations in the model's ability to generalize across diverse classes.

4.1.3 Data labeling iteration 2, (motion) (multi-class) (Philip)

In light of these observations, we adopted a more straightforward classification strategy: multi-classification. Additionally, we shifted our focus from hand-shape-based classes to motion-based classes to achieve a smaller, more distinct set of sign categories. Subsequently, we directed GPT-4 to assign a single label per description, as opposed to multiple labels based on the most relevant keywords in the sign's description. Despite these modifications, the results remained significantly imbalanced. Although setting a fixed limit on variance this time led to a more evenly distributed dataset, concerns regarding the relevance of the labels persist.

#	names	word	description	motions	multi_motions
0	skolkurator-12712-tecken.json	skolkurator	Flata händer, uppåtriktade och inåtvända...	Handskakning	Öppen handflata uppåt; Handskakning; Rör sig i mönster
1	skolkurator-12712-tecken.json	skolkurator	Flata händer, uppåtriktade och inåtvända...	Rör sig nedåt	Handskakning; Rör sig bakåt; Rör sig i mönster

Figure 3: First two rows of dataset in iteration 1

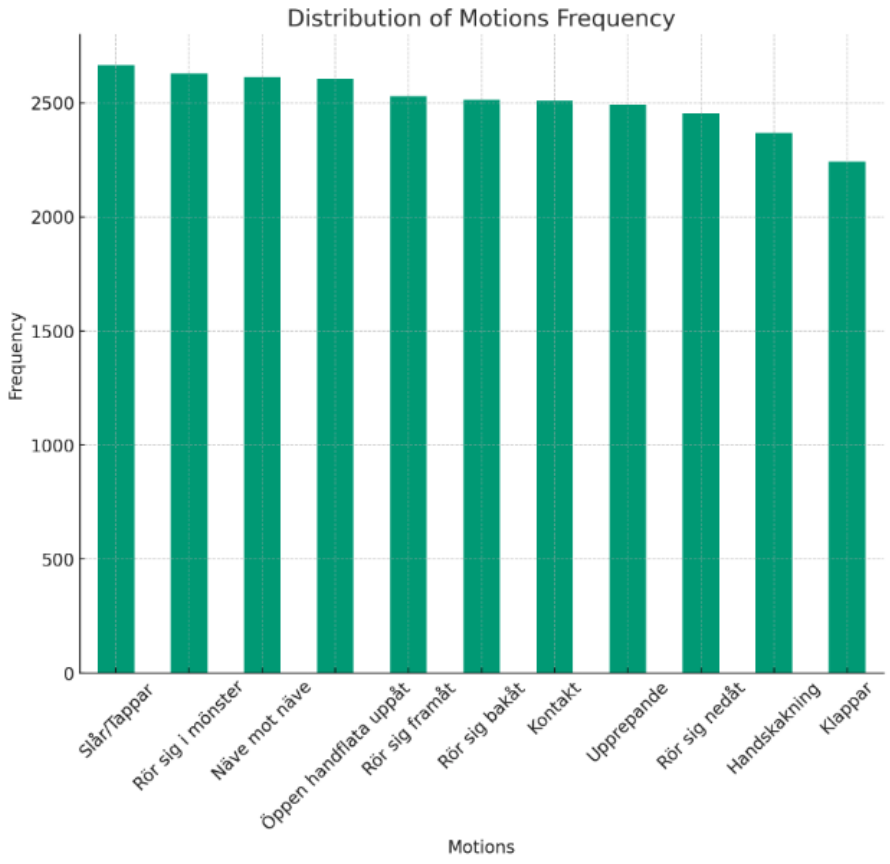

Figure 4: First two rows of dataset in iteration 2

4.1.4 Model results iteration 2

As with the first data labeling, we trained the model using 1000 inputs. Notably, the distribution of classes this time was fairly balanced, ensuring no single class was overly dominant. Despite this improvement, the achieved accuracy remained around 10%. However, it's worth noting that the model consistently predicted the same class almost every time. Given that there were 11 classes, the 10% accuracy roughly corresponds to the proportional presence of each class in the dataset.

4.2 Explanation of the results

In both classification attempts, the model struggled to classify the data accurately. To understand this, we formulated several hypotheses.

Firstly, we considered the possibility of inaccuracies in the labeling process. The labeling was performed by ChatGPT-4, and due to the sheer volume of data, we couldn't manually verify the accuracy or relevance of each label. If the labeling was incorrect, meaning that signs were grouped inappropriately, achieving successful classification would be extremely challenging. We explored this hypothesis further at the end (see section 5).

Secondly, we speculated that the data might not adequately represent the various signs. Even when we trained the models with a higher number of frames (200), the results remained consistent. Additionally, the imbalance in the number of data samples per sign could also pose a problem, with some signs being underrepresented. The dataset's size is primarily determined by the number of distinct signs present.

Another potential explanation is that our choice of model may not have been suitable, despite selecting it based on insights from various literature sources.

5 Further studies

The last approach that we tried was to restrict the dataset to the ten signs that are the most represented among the tokens of the dataset. The goal now is to perform sign classification as follows :

- Input : Frame per frame hand key point coordinates,
- Output : Probability distribution among all the ten signs and use of a decision rule to classify and link the video to a sign. Here, the output consists in a single sign prediction, contrary to before where categories were not exclusive.

5.1 Pre-processing of the data

The pre-processing of the data is slightly different from above and described here :

- Loading .JSON files as done before and only keep the token videos (isolated signs),
- Count how many videos per sign we have,
- Find the ten signs that have the biggest amount of different token videos,
- For every sign, storing the image per image hand position as features, keeping only one image out of four to limit the number of frames per video.

5.2 Building the training, validation and test sets

5.2.1 Training set

To build the training set, we just gathered the frame per frame hand key point coordinates for the signs that we want to classify.

To build the label, we created one-hot vectors of length ten, each of them corresponding to one of the signs.

5.2.2 Test set

For the test set, we saved from 1 to 3 videos (and their features) of each sign that we kept apart from the training set. The amount of video per sign depends on the different trials we did, to find a good balance between training and test size.

5.2.3 Validation set

When used, the validation set corresponds to the last datapoints of the training set (from 30 to 50 in our case).

5.2.4 Padding

Same as before, to have the same number of frames for all our videos, we added a padding. However, it will be managed differently as we will change a bit the model.

5.3 Resampling

Our classes are unbalanced. Thus, to artificially balance them, we used resampling. We tried to different methods :

- **Oversampling** : we took the class with the biggest amount of datapoints and resampled the others so that they all have the same number of datapoints. It results in a dataset with more datapoints than before.
- **Undersampling** : we took the class with the lowest amount of datapoints and sampled every class so that all classes have this same number of datapoints. It results in a dataset with less datapoints than before.

At the end, we also **shuffled** the data.

5.4 Noise on data

When using resampling, espcially oversampling, we get several time the same datapoints. To atificially diversify our data, we added a Gaussian noise of mean 0 and standard deviation 0.001 to all the datapoints, to se if it could offer better generalization perspectives.

5.5 The model

The model used here is very similar to the one made of LSTM units presented before. However, as we observed that the loss was exploding while training, we slightly modified it :

- We replaced the ReLU activation function by the tanh activation function for the LSTM units,
- In the cases we had a validation set, we added early stopping with a patience of 5 to limit overfitting,
- The output activation function is now a softmax function to perform classification, as it is now a one label classification among the ten available.
- Videos don't have the same length, so we have padding. We added a Mask layer before the first LSTM layer so that the padding is not taken into account while training the model.

Here is the update structure of the model :

- **Input Representation**:
  - The input data is represented as a sequence of feature vectors. Each sequence has a shape of $(nb\_frames, frame\_features\_nb)$, where nb_frames represents the number of frames in the sequence, and $frame\_features\_nb$ represents the number of features in each frame.
- **Architecture**:
  - **Masking Layer**: This layer is used to handle variable-length input sequences. It masks out certain timesteps where the input is equal to a specified padding value. This is crucial when dealing with sequences of different lengths.
  - **LSTM Layers**: Three LSTM layers are stacked on top of each other. Each LSTM layer has a different number of units: 64, 128, and 64 respectively. The activation function used in each LSTM cell is the hyperbolic tangent ($tanh$).
  - **Dense Layers**: After the LSTM layers, the model includes two fully connected dense layers with ReLU activation functions. These layers allow the model to learn complex patterns from the LSTM outputs.
  - **Output Layer**: The final dense layer has a softmax activation function and its size matches the number of classes in the output. This layer produces the probability distribution over the output classes.
- **Loss Function and Optimization**:
  - The model uses the categorical cross-entropy loss function ($categorical\_crossentropy$), which is suitable for multi-class classification problems.
  - The Adam optimizer is used with a clipnorm parameter set to 1.0.
- **Training and Evaluation**:
  - The model's fit method is defined to train the model. It takes training data (X_train, Y_train) and optionally validation data (X_val, Y_val). During training, it utilizes TensorBoard for logging and EarlyStopping to prevent overfitting.

5.6 Results

We tried several training parameters. Here are displayed the two best results we get, one using oversampling, the other one using undersampling :

5.6.1 Using oversampling

- 330 video features in the training set, that is, 33 videos per label,
- Use of a validation set of size 40 and early stopping,
- Number of epochs : 100,
- Stopping after 37 epochs,
- Training time : 7 min.

Figure 5: Training metrics using noise free training set

- Prediction precision on the train set : 57.9 %
- Prediction precision on the test set : 40.0 %

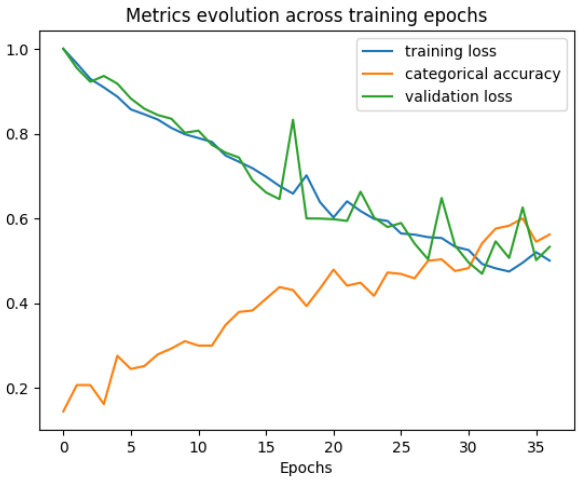

Figure 6: Training metrics using noised training set

- Prediction precision on the train set : 58.3 %
- Prediction precision on the test set : 45.0 %

5.6.2 Using undersampling

- 60 video features in the training set, that is, 6 videos per label,
- No validation set,
- Number of epochs : 150,
- Training time : 3 min.

Figure 7: Training metrics using noise free training set

- Prediction precision on the train set : 100.0 %
- Prediction precision on the test set : 40.0 %

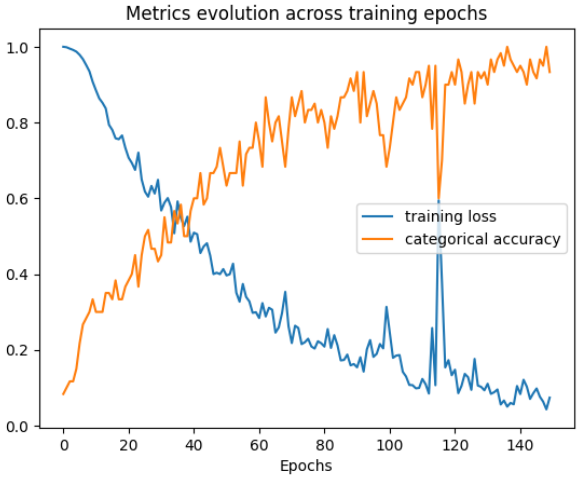

Figure 8: Training metrics using noised training set

- Prediction precision on the train set : 98.3 %
- Prediction precision on the test set : 45.0 %

5.6.3 Conclusion

Using undersampling **makes the training a lot quicker**, and we get in the end a better training accuracy and requires less data to get the same test accuracy, without even using a validation set and early stopping (we couldn't do it because of the smaller amount of data we had for training).
Even if the model does not generalize very well, we could still get results that showed that our model could learn to classify the signs to some extent. We would probably need a more complete dataset better capture the underlying structure of the videos, and thus better classify signs.
It also turns out that using a noised dataset slightly improves generalization perspectives, with the two resampling techniques.

5.7 Limitations

Resampling has its limits and we need more token per sign to better classify signs and have more consistent training, validation and test sets. The amount of data would also need to increase with the number of signs we want to predict. Here we classified only 10 signs, the total amount of signs the sign language has is huge.
All the challenges we had regarding the dataset make difficult to state whether the representation of videos using pose extraction is relevant or not, as we have other non negligible error factors.

References

[JS16] Alex Lee Joe Smith, Emily Johnson. *Beyond Short Snippets: Deep Networks for Video Classification*. Journal of Advanced Machine Learning, 2016.

[PSW12] Roland Pfau, Markus Steinbach, and Bencie Woll. *Sign language - An international handbook*. De Gruyter Mouton, 2012.

[YL17] Zhouxia Wang Yue Luo, Jimmy Ren. *LSTM Pose Machines*. SenseTime, 2017.