Exploring Hybrid Models for ASL to Text Translation

Agnese Adorante[1], Martina Del Gaudio[2], Tommaso Giacomello [3], Kristian Gjika[4] and Clara Montemurro[5]

[1]*3158575* , [2]*3174092* , [3]*3173616* , [4]*3116656* , [5]*3179680*

May 29, 2024

This project investigates the feasibility of translating American Sign Language (ASL) from video input into written text using a variety of deep learning models. Our approach involves leveraging multiple models independently, each with distinct strengths, to tackle different aspects of this challenge. We employ: (1) transfer learning using backbones like InceptionV3, ResNet50, EfficientNetV2L, VGG16, and InceptionResNetV2, with further enhancement through GRUs, bidirectional layers, self-attention mechanisms and transformers; (2) a CNN that extracts frames from videos, compiles them into a single image, and applies a kernel across this composite image; and (3) feature extraction using Google MediaPipe followed by 2D Convolutions. The objective of this work is to compare different ASL-to-text translation models.

Introduction

American Sign Language (ASL) is a comprehensive, natural language, possessing the same linguistic properties as spoken languages. It is articulated through movements of the hands and facial expressions, and it is widely adopted by deaf communities and hard-of-hearing individuals from over 20 countries around the world.
Recent advances in deep learning have shown promise in addressing complex pattern recognition tasks, making it feasible to develop models that can accurately interpret ASL from video inputs.
However, translating ASL into written text remains a complex challenge, as sign languages have their unique grammatical structure, reliance on spatial and temporal features, and the use of body movements is not always straightforward to decode.
The ASL interpretation involves mainly two tasks, namely, word-level sign language (or "isolated sign language") recognition and sentence-level sign language (or "continuous sign language") recognition. In this work, we target at word-level recognition task for ASL.

The Dataset

The dataset used in this work is a large-scale *Word-Level American Sign Language* (WLASL) video dataset downloaded from *Kaggle*, containing more than 2000 words performed by over 100 signers with heterogeneous physical features, and different background color. This dataset combines high and low-quality videos.
In total, the WLASL dataset comprises 21,083 RGB-based videos performed by 119 signers, with each video containing a single ASL sign and lasting 2.5 seconds on average. Each sign is demonstrated by at least three different signers, which are all captured in near-frontal view, without the need for specialized equipment such as depth cameras or colored gloves.

Figure 1: *Overview of interpreters from the WLASL dataset*

The dataset includes a *WLASL_03.json* file that provides a detailed description of the associated videos. For every word in the dataset, there is an instance linked to it. Every instance is represented as a list of dictionaries,

where each dictionary contains essential information about the attributes of the video.
This structure allows for precise identification and segmentation of the relevant parts of each video; additionally, some of the included attributes indicate whether the video instance is designated for training, testing, or validation, specify the range of frames within which the gesture is performed, or characterize the signer ID.

More specifically, the file is organized with the following structure:

Gloss: The words being expressed (the glossary);
Instances: Video instances, which include:

- **fps:** Frames per second, which are approximately constant across different videos;
- **split:** The data split that needs to be performed for modelling (train, validation, test);
- **frame_start:** The frame number corresponding to the start of the gesture representing the word;
- **frame_end:** The frame number corresponding to the end of the word;
- **url:** The link to the video;
- **video_id:** A unique identifier for each videoin the dataset.

This dataset is crucial for understanding sign language sentences, but contains several inherent sources of inconsistency:

Minimal pairs: Since the meaning of signs is primarily determined by the combination of body movements, manual gestures and head positions, even apparently slight differences in handshape, movement, location, or palm orientation, like those in Figure 4, may lead to different interpretations;

Figure 2: *ASL words 'autumn' (up) and 'rural' (down)*

Scalability: The vocabulary of signs used in daily ASL is very large, often comprising thousands of different signs. In comparison, tasks like gesture and action recognition involve far fewer categories, usually only a few hundreds, resulting in a significant challenge for scalability;

Contextual ambiguities: A single sign in sign language may correspond to multiple words in spoken languages depending on the context. Additionally, nouns and verbs derived from the same root often share the same sign;

Inter-signer variations: Signers can perform the same words with different hand positions and amplitude of hand movements;

Dialects: Just as there are accents in speech, there are regional dialects in sign. The variations in ASL primarily include different speeds of signing, lexical, and phonological differences depending on the region. A distinct variety of ASL used by the Black Deaf community has also evolved in time.

Dataset Filtering

To develop a practical ASL recognition model, the training data must include a sufficient number of classes and examples, whilst the dataset we are working on presents incongruencies due to the variable number of videos per word and the varying number of frames within each video.
Ideally, there would be around 10 videos per word if the classes were perfectly balanced. However, this is not the case; for instance, the word "book" has approximately 40 videos, while other glosses are significantly underrepresented.
To address this imbalance, we counted the number of videos available for each word. To avoid significant disparities, we excluded words with a disproportionately large number of samples and removed those with insufficient samples. This approach ensured a more balanced and robust training set for the ASL recognition model.

1 Transfer Learning

This part of the project focuses on video classification through the application of transfer learning with a variety of pre-trained convolutional neural networks (CNNs), adapting them for our ASL recognition model. We employ different CNN backbones to leverage their feature extraction capabilities:

- **InceptionV3** uses factorized convolutions and aggressive regularization to capture complex spatial hierarchies in video frames.
- **ResNet50** employs deep architecture and skip connections to learn intricate patterns without the risk of vanishing gradients.
- **EfficientNetV2L** balances accuracy and computational efficiency by scaling depth, width, and resolution systematically.
- **VGG16** is a simpler architecture with deep convolutional layers for strong baseline feature extraction.
- **InceptionResNetV2** combines Inception and ResNet strenghts for high performance with lower

computational cost through residual connections and inception modules.

We optimize these models for video data by integrating advanced layers and mechanisms:

- **Gated Recurrent Units (GRUs)** to capture temporal dependencies within video sequences.
- **Bidirectional layers** to enhance context understanding from both past and future frames.
- **Attention mechanisms** to improve the focus on relevant parts of the video frames.
- **Transformers** to leverage their powerful sequence modeling capabilities.

Finally, we replace the pre-trained CNNs' classifiers at the head with new ones tailored to our problem. These will consist of **Multi-Layer Perceptrons** (MLPs) and additional **CNN layers** to refine feature extraction and improve classification accuracy. The classification part will be shared for every proposed model, while the aforementioned integrations will be tried one by one on all CNNs. This comparative analysis aims to identify the most effective model architecture for ASL video classification.

1.1 Data Processing

To begin our process, we extract frames from the videos and format them for input into pre-trained CNNs. To maintain consistency across videos, we standardize the number of frames to 90. Specifically, we select the midpoint frame of each video and include an equal number of frames from both sides. This approach ensures that the selected frames better represent the action being performed.
Additionally, to ensure a balanced dataset, we only include words that have at least 15 videos in the training set. For our initial analysis, we focus on 10 labels to keep the task manageable. From these selected videos, we extract the relevant feature maps using the various pre-trained CNN architectures mentioned earlier. These feature maps are then used as input for the classifier.

InceptionV3

Configuration	Max Accuracy (%)
Transformer	62.07
Bidirectional	27.91
Sequential	20.93
Selfattention	18.60

Table 1: *InceptionV3 Model Performance*

ResNet50

Configuration	Max Accuracy (%)
Transformer	37.21
Sequential	18.60
Bidirectional	18.60
Selfattention	16.28

Table 2: *ResNet50 Model Performance*

EfficientNetV2L

Configuration	Max Accuracy (%)
Transformer	37.21
Sequential	18.60
Selfattention	16.28
Bidirectional	11.63

Table 3: *EfficientNetV2L Model Performance*

VGG16

Configuration	Max Accuracy (%)
Transformer	37.20
Bidirectional	13.95
Selfattention	9.30
Sequential	11.62

Table 4: *VGG16 Model Performance*

InceptionResNetV2

Configuration	Max Accuracy (%)
Transformer	34.88
Bidirectional	16.28
Selfattention	16.28
Sequential	9.30

Table 5: *InceptionResNetV2 Model Performance*

1.2 Models Performance

The metrics we decide to employ for evaluation are Sparse Categorical Accuracy and TopK Categorical Accuracy. The former measures the proportion of correctly predicted labels out of the total predictions, and is suitable for classification problems with mutually exclusive classes like ours. The latter evaluates the model's performance based on whether the true label is within the top K predicted labels, particularly useful in scenarios where there are many classes. We decided to use the top 3 labels.

Here we report the results obtained in terms of maximal accuracy reached, in decreasing order.

The integration of Gated Recurrent Units (GRUs) in our Sequential model did not yield good results, despite their suitability for temporal data. We then tried bidirectional layers, which improved performance slightly, but not significantly. Moving beyond traditional CNN and RNN approaches, we explored transformer architectures with self-attention mechanisms starting with 1D convolutions over the temporal axis. Although the self-attention mechanism surprisingly underperformed, transformers clearly outperformed all other methods. Using InceptionV3, transformers achieved a maximum accuracy of 62.07% on validation data and a top-3 categorical accuracy of 96.55%.
Here, we plot the results obtained with this model.

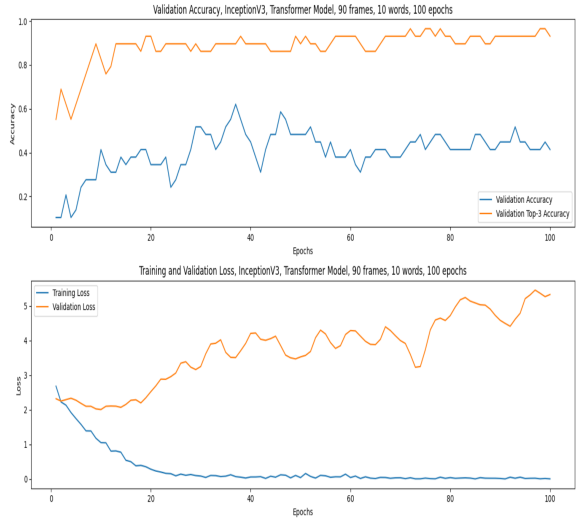


Figure 3: *Loss and accuracy for InceptionV3 + Transformer*

1.3 Results and Challenges

Despite experimenting with various architectures, our models underperformed compared to others, so we decided not to expand our analysis to more labels. Although InceptionV3 achieved a good maximum accuracy, the accuracy curve was highly noisy and the validation loss started to diverge, indicating overfitting. This variability is likely due to the low quality of the dataset, which remains the primary challenge of our study.

---

2 Streamlining Video Classification into Image Analysis

In this section, we will examine techniques to reduce the computational burden associated with demanding tasks such as video classification. Specifically, we will reformulate the problem as an image classification task and employ more cost-effective algorithms to address it. We will explore two primary architectures:

- **2D CNN**: This architecture comprises two convolutional layers (Convolution, Batch Norm, ReLU, average Pooling), followed by a Multi-Layer Perceptron (MLP) consisting of 2 dense layers (Linear, Batch Norm, RELU, Dropout);
- **MLP**: In this approach, the image is treated as an array, and we apply 3 dense layers (Linear, Batch Norm, RELU, Dropout);

2.1 From Video to Image

To convert a video into an image, we will perform the following steps:

- **Frame Extraction:** To simplify computations, we selected a set of 9 frames following the method outlined in **Section 1**. We therefore identified a central frame and included 4 frames before and four frames after it.
- **Image Creation:** Once the frames were chosen, we arranged them in a 3x3 grid to form a single composite image.
- **Data Augmentation:** Given the limited amount of data, we decided to augment our dataset using three techniques for each image, to effectively triple its size. First, we applied a random Rotation(θ) with angles chosen from $\{90, 180, 270\}$. Then, we utilized a "Speed up" method, which involves skipping certain pixels in the color channel to reduce image detail. Finally, we employed a "Horizontal Flip" technique, which reverses the intensity of pixels in the color channel.
- **Resizing:** Images are resized, ensuring uniformity in the dataset.
- **Normalization and Grayscale:** Due to significant contrast variations and differences in recording condition, we converted the images to grayscale and normalized them. This process helped mitigate unwanted noise and reduced computational demands.

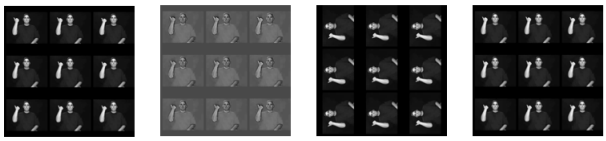
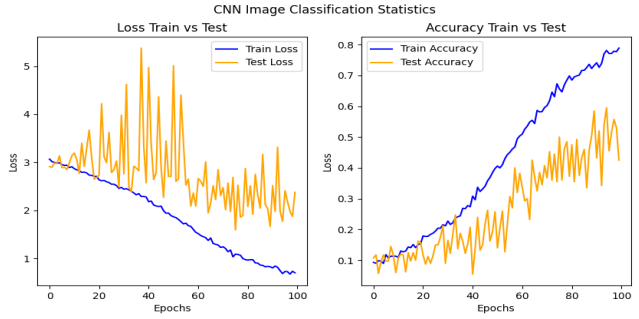


Figure 4: *Example of used dataset*

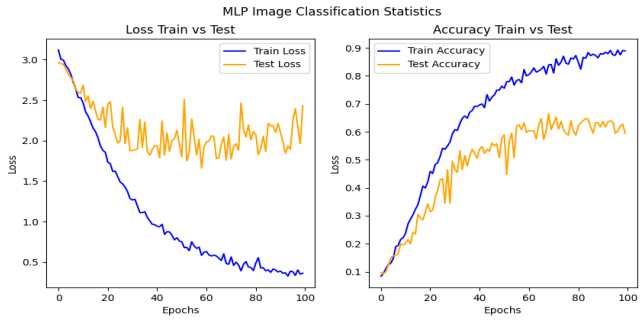
2.2 Model Performances

In this section, we present the effectiveness of the aforementioned architectures. These models were trained using a dataset consisting of images representing the 20 most frequently expressed labels.
The images were processed in batches of 10 and trained for 100 epochs. Prior to inputting the images into the neural network, they were resized from (1,768,768) to (1,256,256). This resizing step was undertaken to reduce computational burden while maintaining the integrity of the dataset's structure.
We begin our analysis with the **2D CNN**, which achieved a maximum testing accuracy of 55%, and a training accuracy of 80%.

However, given the imbalanced nature of the dataset's classes, we evaluated the model's performance based on other metrics, in order to avoid relying solely on accuracy. More specifically, we inspected Precision, Recall, F1, and ROC-AUC score. Notably, the F1 score is close to 0.6, and appears simi- lar to Precision and Recall, indicating a good trade-off between these



last two metrics. Additionally, a ROC- AUC score of 0.8 confirms the validity of this model compared to a random classifier.
Regarding the **MLP**, we achieve surprisingly a similar accuracy testing accuracy of 60% a training accuracy of 90%:



Moreover, the other scores follow the previous trend, with F1, Accuracy, and Precision all around 0.6. On the other hand, we observe a significant improvement regarding the ROC-AUC score, achieving a score of 0.9.

2.3 Alternative Architecture

To address the video classification challenge from a different perspective, we implemented an alternative approach.
Following the same method as previously described, we extracted frames from the video. However, instead of combining them into a single grayscale image, we appended each selected frame, after normalization and conversion to grayscale, into the color channel. This resulted in a composite image of dimensions (9,256,256). Subsequently, we constructed a 2D CNN architecture similar to the previous one, but designed to accommodate multiple channels. Despite this innovative approach, we observed slightly inferior performance compared to the previous architecture, achieving an accuracy of 60% and comparable metrics for the other evaluation scores.

2.4 Conclusion and Limitations

We reached an accuracy of about 55 % for the CNN and 60 % for the MLP.
This result is quite unsettling, since we expected CNNs to perform better on images due to their translation invariance property.
On the other hand, MLPs are generally easier to train on small datasets, while CNNs require more data to effectively learn the weights of convolutional filters. With limited data, the training of CNNs can be unstable, leading to sub-optimal performance compared to MLPs.
It's also worth noting that the findings were somewhat noisy, especially in the test set. This is most likely the result of our extraction process that ignored the temporal dimension. Finally, an unstable loss and a clearly overfitting model can only worsen when increasing the number of classes.
We will address this issue in the following sections by employing different models.

---

3 Landmarks CNN

In this section, we explore the use of Google MediaPipe to extract landmarks from video frames in the dataset and subsequently feed them into a 2D CNN for translation tasks.

3.1 Data Preprocessing

The *WLASL_v0.3.json*, introduced at the beginning, consists of two video directories:

- **wlasl-processed:** contains higher-quality videos, though not all videos were available;
- **wlasl2000-resized**: serves as a backup for missing videos, albeit with somewhat lower quality.

Despite the preference for higher quality videos for our project, we decided to include videos from both directories to ensure a comprehensive dataset. More specifically, if we discarded the lower quality videos from the analysis, it would have resulted in the loss of more than half of the total available videos.

The necessary information for our task were:

- **gloss**
- **frame_start**
- **frame_end**
- **split**

Once the data was organized, we proceeded with landmark extraction.

3.2 Landmarks Extraction

In the context of computer vision and machine learning, a *landmark* refers to specific, predefined points on an object that are used to understand its structure and spatial configuration. Landmarks

are typically chosen because they are stable, easily identifiable, and relevant.
The approach we followed to extract landmarks from videos relies on **Google MediaPipe**, an open-source framework developed by Google, providing a suite of tools for real-time perception in videos and images. It is particularly suitable for tasks that require immediate feedback and accuracy, such as gesture recognition, object tracking, and pose estimation.

Figure 5: *Hand landmarks extracted from a low-quality frame*

For our ASL translation task, we initially considered various choices of landmarks. In related work on WLASL translation, it is common to use landmarks from the entire body, including hands, face, and body, as all these regions can be relevant for communication. However, due to the extensive demand of computational resources required by landmark extraction, and the quality of the available video data, we determined that prioritizing hand landmarks would be the most effective approach.
In particular, we noticed that a high intrinsic noise in facial pixels from low quality frames jeopardised the analysis, as it was likely to lead to poor and unreliable landmark extraction.
Therefore, we decided to balance the need for accuracy and computational efficiency by extracting 21 landmarks for each hand. To do so, we defined a process combining frame and video landmarks extraction which enabled precise hand tracking and analysis.

3.3 Main Architecture

After extracting landmarks, we give them to a CNN. This has to be able to handle sequences of different lengths. Indeed, for every video, we have 2 constant dimensions, the landmarks and the spatial coordinates ((42,3)), but a variable temporal dimension. Neural networks, particularly those using mini-batch processing, require that all samples in a batch have the same dimensions in order to perform parallel processing. We thus decided to pad shorter sequences with zeros up to the longest sequence in the batch. It is important to note though that the sequence length has to be consistent only within batches, so that matrix operations can be performed safely, but the number of parameters for the CNN is independent from it.

The architecture has these key components:

Feature extraction

- **2 convolutional layers**: It takes inputs with 3 channels, 1 temporal dimension, and 1 spatial dimension. The convolution is applied only on the temporal one.
- **Batch Normalization**: Normalizes the output of convolutional layers. There is one after every convolution.
- **Pooling**: We perform 2 max pooling operations after every normalization. Pooling is done only in the temporal dimension.
- **ReLu** : activation function after every pooling layer.
- **Dropout**: there is one after every ReLu function.

Classification

Before passing the feature maps to the dense layer, we perform an average pooling on the temporal dimension. This approach allows us to handle sequences of different lenghts without the need of recurrent networks or more complicated structures. In this way, the input shape of the dense layer is always the same.

3.4 Model Performance

In this section, we proceed with the model evaluation. For two reduced datasets, containing 20 labels and 50 labels respectively, we run 100 epochs with a batch size of 32 and track the validation and training loss, along with the accuracy.
First, we use the the reduced dataset with only 20 labels to inspect the quality of our model's performance over time. Indeed, a notable behavior can be inferred from the plot in Figure 6: the model does not seem to be overfitting, and the curves displaying the loss and the accuracy for the training/validation data appear sufficiently smooth and not significantly spiky. Additionally, the trends of the curves indicate that

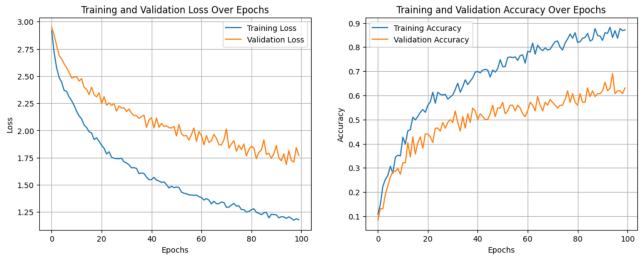


Figure 6: *Loss and accuracy for training and validation data*

they have not yet stabilized, suggesting that extending the training for more epochs could potentially lead to better results. The highest validation accuracy is attained at a peak value of 70%.

Then, to assess the general capabilities of our model, we also evaluated it on the second, more complex

version of the dataset, comprising 50 labels. Naturally, given that the model's architecture is kept unchanged, we anticipated an overall decrease in performance as the dataset complexity increased.

Nonetheless, our results appear promising: the overall

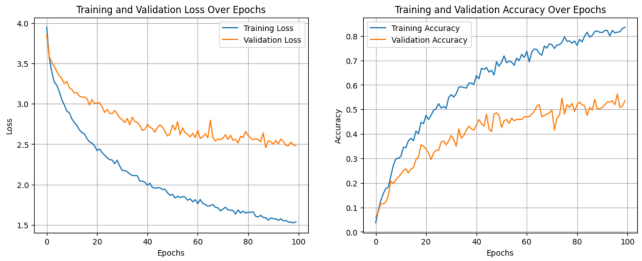


Figure 7: *Loss and accuracy for training and validation data*

trend shown in Figure 7 remained consistent, although there was a slightly increase in the validation loss. The highest accuracy achieved on the validation dataset was 57%.

4 Comparative analysis of the models

Overall, our best performing model was the 2D-CNN using as a baseland the extracted landmarks from videos using MediaPipe.

Transfer Learning proved to be insufficient due to the specifity of our dataset, made of videos of sign language. Pre-trained models used more well defined and distinguishable videos as training set, thus probably learning features that were not relevant for our purpose. We could have trained our own model to learn how to detect hands and more specifically hand gestures, but our dataset was not sufficiently large to perform such task.

The convolution applied to stacked frames proved to perform better, at least looking only at the maximum validation scores obtained, yet the loss and accuracy curves were extremely noisy and unstable. This could mine the generalization to more labels.

Also the MLP, despite having a high accuracy, showed an evident overfitting in the loss, with a strong difference between train and validation. This led us to our final decision of using the 2D-CNN on landmarks to perform a try on 1000 labels.

5 Conclusive Analysis

To ensure the robustness of our approach, we evaluated our best-performing model, the 2D-CNN with landmarks, on a significantly larger set of 1000 labels. We selected words to maintain a balanced dataset, choosing only those represented by 10 to 20 videos each.

Although this complex task might typically require deeper networks, our project's goal is to explore the performance differences among various models for sign language translation rather than to find the optimal performing model. This test is thus performed only for the sake of completeness. As we observed when increasing from 20 to 50 labels, the validation loss initially decreased but then stabilized to a fixed value. Importantly, the stabilization happened after fewer than 40 epochs, indicating an inherent limit of the model rather than a tendency to overfit. The same reasoning extends to the accuracies. Our results are still promising, since we achieved a maximum accuracy of 27%, which is commendable given the complexity and size of the label set, as well as the simplicity of our architecture. The stabilization of validation loss and accuracy indicates that the model reaches its capacity without overfitting, showcasing its potential for handling large-scale classification tasks within the constraints of a simplified architecture.

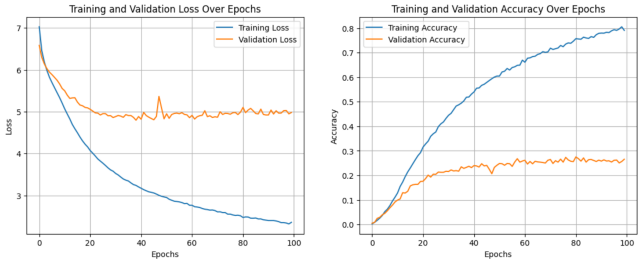


Figure 8: *Loss and accuracy for 1000 labels*

These outcomes suggest that our simplified model effectively captures the essential features necessary for sign language translation. The results imply that extracting landmarks as an intermediate step is crucial for the model's efficiency and performance. This step ensures that the model focuses on the most relevant aspects of the input data, namely, the hand movements, which are critical for accurate ASL interpretation. In particular, a specialized model that extracts hand landmarks consistently outperforms a more general model that processes the entire image. In this way, we reduce the noise and irrelevant information present in the broader context, thus enhancing the model's ability to learn and generalize from the data, remaining robust and effective across varying datasets.

References

WLASL (World Level American Sign Language). `https://www.kaggle.com/datasets/risangbaskoro/wlasl-processed?select=nslt_2000.json`