Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Enhancing Sign Language Translation with Graph Neural Networks for Hand Pose Classification and Fingerspelling Sequence Translation

Scientific Students' Association Report

Author:

Marcell Dancsó

Advisor:

Dr. Péter Ekler

2024

Contents

Kivonat

Annak ellenére, hogy a természetes nyelvfeldolgozás és a jelnyelvfelismerés terén jelentős előrelépések történtek, a nagyjából 1,5 milliárd fős siket és nagyothalló közösség számára továbbra is hiányoznak a hatékony fordítási megoldások. Korábbi munkám pózbecslési algoritmusok és hagyományos neurális hálózati architektúrák alkalmazását vizsgálták Azonban a kézmozgások összetett dinamikája, ahol az ízületek hatással vannak egymásra, különleges kihívást jelent.

Kezdetben az egyes kézpózokat hagyományos módszerekkel, például MLP alapú megközelítésekkel osztályoztam. Ebben a folytatólagos kutatásban, ugyanezeket a pózokat gráf neurális hálózatokkal, azon belül: GCN és GAT architektúrák alkalmazásával osztályozom, ahol a kéz ízületeit csomópontokként modellezem, az ezek közötti kapcsolatok pedig az ízületek közötti függőségeket tükrözik. Az ujj betűzés fordításához, amely korábban transzformer architektúrát használt egy konvolúció alapú “embedding” réteggel a póz sorozatokhoz, most egy egyedi gráf alapú embedding réteget tervezek. Ez várhatóan hatékonyabban képes megragadni a kézmozdulatok relációs szerkezetét, miközben továbbra is kihasználja a transzformer modellt a szekvenciális adatok feldolgozásához.

A kutatás célja annak a hipotézisnek az igazolása, hogy a GNN-ek bizonyos körülmények között, például bonyolult jelnyelvi pózok esetén, ahol az ízületek erősen függenek egymástól, felülmúlhatják a hagyományos módszereket. Ennek teszteléséhez először több GNN architektúrát valósítok meg, majd átfogó kísérleteket végzek azok teljesítményének mérésére. A fordítás pontosságát és a feldolgozási sebességet a korábbi modellekkel hasonlítom össze.

Emellett megvizsgálom az új megközelítés gyakorlati használhatóságát valós idejű alkalmazásokban, figyelembe véve a számítási hatékonyságot és a különböző jelnyelvi dialektusok közötti skálázhatóságot. A GNN-ek fordítási teljesítményének vizsgálatával, a kutatás célja, hogy betekintést nyújtson a hozzáférhető, magas minőségű jelnyelvfordító rendszerek jövőjébe.

Abstract

Despite advancements in natural language processing and sign language recognition technologies, effective translation solutions for the deaf and hard of hearing community, comprising approximately 1.5 billion individuals globally, remain elusive. Previous work explored pose approximation algorithms and conventional neural network architectures to improve translation accuracy. However, the complex dynamics of hand movements, with joints influencing one another, continue to pose a unique challenge.

Initially, individual hand poses were classified using traditional methods, such as multilayer perceptron-based approaches. In this follow-up research, I propose classifying the same poses using Graph Neural Networks (GNNs), specifically the GCN and GAT architectures, to model the hand as a graph where each joint acts as a node and the connections between them represent joint dependencies. For translating fingerspelling sequences—previously relying on a transformer architecture with a convolution-based embedding layer for pose sequences—I now introduce a custom graph-based embedding layer. This layer is designed to capture the relational structure of hand movements more effectively while still utilizing the transformer model for sequential data processing.

This research aims to validate the hypothesis that GNNs can outperform traditional methods under specific conditions, such as complex sign language poses where joints are obscured or highly dependent on one another. To test this hypothesis, I will design and implement several GNN architectures, followed by comprehensive experiments to measure their performance. Metrics such as translation accuracy and processing speed will be compared against prior models.

Additionally, I will explore the practical usability of this approach in real-time applications, considering challenges like computational efficiency and scalability across diverse sign language dialects. By examining the conditions under which GNNs can enhance translation, this research seeks to provide insights into the future of accessible, high-quality sign language interpretation systems. Ultimately, the goal is to refine and improve current sign language translation technologies, making them more reliable and adaptable to real-world scenarios.

Chapter 1

Introduction

In recent years, we have achieved remarkable breakthroughs in the field of natural language processing (NLP). With today's "voice-to-text" models, we can interact with our devices in a diverse and natural way. Coupled with the revolutionary advancements of large language models, we can now employ virtual assistants, access the world's knowledge through chat interfaces, and even unlock the possibility of seamless communication between any two languages. However, one unfortunate limitation of these systems is that they are available only in traditional spoken languages. While the technology seems ready, no substantial system supporting sign languages has yet emerged. Translating sign language into text would open up communication for approximately 70 million people [29], allowing them to interact with smart devices in their own language, not to mention facilitating communication with those who don't know how to sign. It would also greatly assist in the education of individuals who, despite living with hearing impairments, do not have access to sign language education due to a lack of financial resources or learning tools. In fact, this group is surprisingly larger in many countries than the number of people who can sign, highlighting the complexity of the language and the challenge of this task.

1.1 Fundamentals of Sign Language

Sign language is a visual, gesture-based language used by both deaf and hearing communities for communication. Unlike spoken languages, meaning in sign language is conveyed through hand movements, facial expressions, and body posture.

Contrary to popular belief, sign language is not universal; many distinct versions exist worldwide, each with its own structure. For instance, American Sign Language (ASL) and British Sign Language (BSL) are so different that they are mutually unintelligible, even though both countries speak English. ASL is rooted in French Sign Language [28], while BSL evolved independently [27]. Furthermore, like spoken languages, sign languages have dialects and regional variations, which adds further complexity to the development of intelligent solutions.

1.2 American Sign Language (ASL)

American Sign Language (ASL) is used by the deaf communities in the United States and Canada. Its history dates back to the 19th century when Thomas Gallaudet and Laurent Clerc founded the first school for the deaf in the U.S. [28]. The language used in this school combined local American sign language with French sign language, forming the foundation of ASL.

This study will focus solely on ASL, as it is the most accessible in terms of well-established datasets and knowledge about the language. It's important to note, however, that the models and algorithms developed for sign language datasets are often universal, regardless of which sign language they are applied to.

1.3 Fingerspelling

A subset of most sign languages is fingerspelling (FS), where handshapes represent letters. In some sign languages, finger-spelling is commonly used to convey names, foreign words, or specialized terms that don't have their signs. In other languages, finger-spelling is used less frequently, with signers preferring complete phrases and sentences. Additionally, there are smaller sign languages around the world, which have developed in unique communities and are not influenced by the larger, more widely used sign languages. An example is Kolok Kata [22], or Balinese Sign Language, a village sign language native to two neighboring villages in northern Bali, Indonesia. Interestingly, Kata Kolok has no official FS system, demonstrating that FS is not essential for sign languages to function. These smaller languages often reflect the culture and history of the community in which they developed.

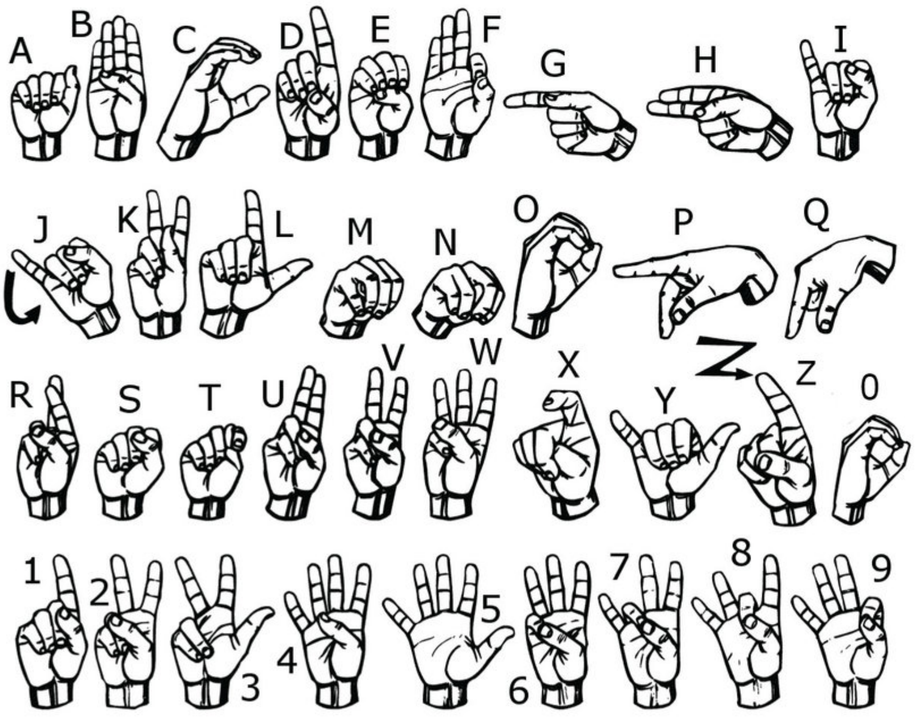

Figure 1.1: Extended ASL Alphabet. (Arrows showing the two signs that require motion.)

Like many other sign languages, ASL includes FS, which is frequently used for names, addresses, phone numbers, and other information commonly typed on a mobile phone. Many deaf smartphone users find FS faster than typing out words, hence being significantly faster than typical virtual keyboard typing, averaging 57 words per minute compared to the U.S. typing average of 36 words per minute [15]. Unlike British Sign Language, which requires both hands, ASL uses only one hand to communicate the letters of the alphabet and numbers. Although other gestures can have meaning in this context, the letters can be identified solely by observing the dominant hand. For example, head movements or leaning forward during finger-spelling can emphasize certain letters.

FS is a great stepping stone for translating more complex gestures relying heavily on facial expressions, head movements, and additional gestures to convey meaning.

1.4 Taxonomy of Sign Language Translation

In sign language translation, three primary approaches are distinguished based on input and output types. The first approach, known as static or isolated, processes discrete units and relies on single snapshots, such as a static image as input or an individual predicted sign as output. The second approach, sequential or sometimes called dynamic, enables models to work with time series data, like video, capturing temporal information to construct complete sentences.

A third dimension in this taxonomy is grammar [26], as visual languages possess rich grammatical diversity, much like spoken languages. Sign language grammar includes phonological parameters divided into manual and non-manual markers, with manual markers covering hand shape, movement, location, palm orientation, and body shift. Non-manual markers like body language, eye gaze, and facial expressions add emotional and grammatical context, while classifiers are specialized handshapes that represent categories or classes of objects, people, or concepts, allowing signers to convey more detailed descriptions beyond individual words. According to this definition, FS is also a special set of manual markers conveying the meaning of concepts without established signs.

As for ASL, static input is only viable for FS and certain grammar markers, as most of the general gestures involve movement, but for sequential input predicting both a single concept or a sentence is possible. This study focuses on translating fingerspelled characters and sequences.

Table 1.1: Applying Taxonomy on ASL Subtasks

Input/Output	**Static**	**Sequential**
Static	Static Fingerspelling	-
Sequential	General Sign Prediction	Sequential Translation

Chapter 2

Related Work

Preliminary research highlights that solutions in this field generally include two main components: signal processing and translation algorithms. Signal processing is essential because models cannot directly interpret input from the environment; rather, they require optimal representations of reality from various perspectives. Examples of such representations include segmenting hands in camera images or detecting finger bending in glove-based approaches. Additionally, this step is closely linked to the creation and management of datasets used for model training.

A third component, often overlooked, is the use of supplementary algorithms in practical applications. These algorithms enhance the reliability and usability of model outputs, especially when deploying models in real-world settings.

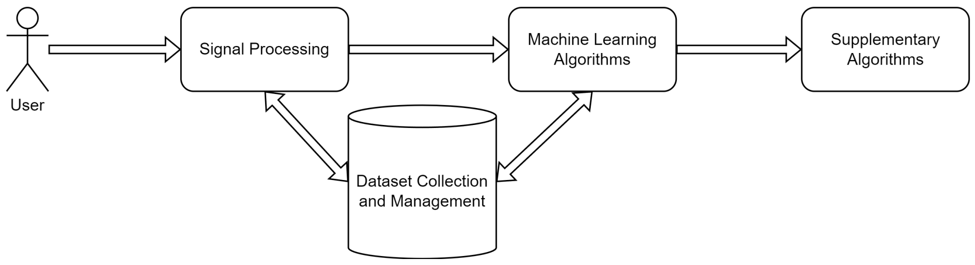
User
Signal Processing
Machine Learning Algorithms
Supplementary Algorithms
Dataset Collection and Management

Figure 2.1: General Architecture

2.1 Signal Processing

It is evident from exploratory research that processing the language signal conveyed through the physical medium of hands, is one of the most crucial parts of the task. If the representation of the signal is not appropriate, it isn't easy to build well-performing models on it. Signal processing techniques can be broadly divided into three primary categories: traditional image processing methods, hardware-enabled translation systems, and pose estimation-based solutions. Each of these approaches offers distinct advantages and faces unique challenges in accurately capturing and interpreting the complex movements and gestures involved in sign language.

2.1.1 Traditional Image Processing Based Translation

It is true for such solutions that they transform the images into a higher form of representation like hand segmentation [9], optical flow calculation [11], or artificially generated depth maps [10], however, they don't model the physiology of the hands. This can be vital in situations where not all fingers are visible from a given camera angle, and perhaps the sign cannot be deduced accurately solely from the image information.

2.1.2 Hardware Enabled Translation

In these studies, a suitable hardware device is used to collect the necessary input for the models instead of a camera feed. This in many cases guarantees more sufficient spatial knowledge of the hands. For example, in a recent study, the movement of hands was measured using a ring with IMU sensors [20]. More traditional methods include tracking the exact position of fingers with gloves [1][32][3]. Drawbacks include Hardware cost and inconvenience of use.

2.1.3 Pose Approximation

This third category of processing methods is an innovative image-processing technique aimed at identifying the positions of both visible and concealed hand joints, also referred to as landmarks. This is accomplished through the utilization of deep learning. By exposing the models to numerous examples of manually labeled or hardware-recorded instances, they understand the anatomical intricacies of these joints, enabling them to detect potential joints that may not be directly observable.

This approach boasts 4 important qualities over other methods:

1. Using the physiological model of the hand without any auxiliary tools.
2. Possibility to extend tracking to posture and facial features.
3. Able to utilize existing video/image format databases.
4. All of the above within real-time constraints.

Two famous baseline algorithms in the field of pose approximation: OpenPose [7] and Mediapipe Holistic [14]. The latter is a composition of 3 separate models: BlazePose [6], MediaPipe Hands [34] and BlazeFace [5]. They both support real-time whole-body tracking from videos and still images as well, but while OpenPose only provides 2D landmarks, Mediapipe also predicts relative depth, referred to as 2.5D [34] by the authors. Intuitively this becomes crucial when gestures involve intertwined fingers, favoring Mediapipe Holistic, but a comparative study [25] with two separate teams concluded that "On the test set, despite making different predictions, both OpenPose and Holistic performed equally well", and training the model on the combined set of poses yielded the best result. It's also worth mentioning that there are efforts to make sign language recognition-specific models, like the one conducted at the University of Surrey [16], showing promising results, outperforming MediaPipe on several datasets, while also employing a special layer predicting joint angles and propagating position and rotation from a selected pivot point using forward kinematics.

2.1.4 Conclusion

There is no clear winner in terms of signal processing methodologies for sign language recognition, as several recent studies, with different algorithms, showed a high degree of accuracy on various datasets [23][12][35], and there are examples where combining different type of processing methods like optical flow, depth maps, pose skeleton as spatio-temporal graph is also experimented with [17].

It must be noted however, in recent years pose approximation-based translation has seen a significant bump in popularity and is currently the most popular method to handle gesture translation. Current trends suggest that it will become superior to other methods. A study conducted on the CVPR21 ChaLearn challenge dataset [25] concluded that despite challenges with recognition of overlapping and interacting hands, "pose estimation features do indeed generalize better to the nature of the challenge, including unseen signers and backgrounds".

2.2 Translation Algorithms

Backed by literature [26], the most common translation algorithms in the field of sign language translation, are encoder-decoder models, out of which Transformers dominate the leaderboards. In this section, I present studies that propose thought-provoking ideas and are relevant to this research:

1. PoseNet [12] is a Transformer [30] model modified to translate FS sequences into continuous text. Similarly to my previous work (2.3) the full body pose skeleton is provided by MediaPipe Holistic [14], however only the x and y coordinates are utilized during training. For embedding the features, a fully connected layer is applied, scaling them up to the desired dimensions, before using positional embedding. The most intriguing part of the model is a new way of handling the length of the generated sequences: instead of solely relying on the decoder to predict the end of sequence character, the model is capable of computing this information earlier in the encoder layer. Unlike in the general case, there is a 1-1 mapping between fingerspelled characters and the corresponding symbols in the output sequence. Computing this information earlier, where the latent space representation is not yet mixed with the decoder input is beneficial according to the author.

2. The idea of imagining the skeletal pose data as a graph is fascinating, as we can apply algorithms designed for processing graphs. The following study [24] works on classifying general sign language poses using this very technique. From the arbitrary long sequences, a spatial-temporal graph is constructed and processed with a modified spatial-temporal graph convolution network [33] (ST-GCN). Each video frame has its graph and the corresponding points on consecutive frames are also connected, hence the name. After unrolling the network a few times, each node in the graph will have a receptive field not only in the spatial but also in the time domain making this method ideal for dynamic movement recognition. Originally the graph for each frame is connected following the human anatomy, therefor the nodes representing the face and hands are only distantly related in the graph. For signing, however, there's much emphasis on the relative relative position. Some signs only differ in the position of the face that is touched. The authors solve this problem by using only a handful of key points and constructing their fully connected graph representation.

3. Another approach to the previous problem leverages hierarchical spatial-temporal graphs [18]. In this method, spatial-temporal graphs are constructed across multiple levels: fine-level graphs represent each "modality" (hands and face) independently, while two additional high-level graphs are constructed with nodes whose features derive from the bounding box positions of each modality. Once the spatial-temporal graphs are constructed, all the fine-level graphs are processed with ST-GCN models [33]. For each frame in the sequence, these fine-level graphs are average-pooled by calculating the mean features across nodes in a single frame to build a new high-level spatial-temporal graph from the three sequences of vectors. After several iterations, all graphs are average-pooled in the same way, and the resulting vectors are concatenated, forming a latent space representation for each time step within the encoder. Finally, the sequential translation task concludes with a two-stage decoding process using LSTM layers: in the first stage, a sign is predicted at each time step, and then a separate LSTM-based encoder-decoder refines this into the final output sequence.

4. The following study [23] showcases that graph neural networks can also be applied to images as well. The research presents a dual-stream approach utilizing features extracted with both regular CNN and graph convolutional layers. It wouldn't be efficient to treat all pixels as separate nodes, so a SLIC superpixel [2] algorithm is applied that clusters nearby pixels into small, coherent regions (superpixels) that group together pixels with similar characteristics, such as color, intensity, or texture. Unlike individual pixels, superpixels capture larger and more meaningful structures within an image, making them useful for simplifying and speeding up higher-level computer vision tasks, like object detection and segmentation. Each superpixel is treated as a node with edges to the neighboring regions. With Graph Convolution and the previously mentioned "regular" feature extraction, the model performs sign classification.

2.3 Previous Work

In my previous paper, I explored pose approximation techniques to translate both individual signs and gesture sequences, building and training several models on diverse datasets while applying the best-performing models in real-world applications. This work serves as the foundation for the current study, briefly summarized in the following subsections.

2.3.1 Static Fingerspelling

For static fingerspelling, the focus was on handling pose information and building an efficient model. Pose data was captured using the MediaPipe Hands-specific model, which provided 3D coordinates for key points. Preprocessing involved scaling, centering, and normalizing the 2D coordinates to enhance consistency across samples. I further applied data augmentation techniques, including affine transformations, to improve model robustness. A Dense neural network, optimized with the Hyperband algorithm [21], was trained on the augmented dataset to achieve effective pose classification, with testing conducted on a separate manually recorded dataset.

2.3.2 Sequential Translation

The future of sign language translation lies in sequence-to-sequence algorithms, which have shown great success in traditional language translation by capturing complex relationships within sequences. Unlike general sign prediction, where models interpret isolated frames, sequence-based translation enables a more nuanced contextual understanding.

Implemented models can be split into two families:

1. The first one is an adaptation of the original Transformer architecture [30], with modifications to the embedding layer for handling pose data. (More on that in a Section 4.2.) The architecture includes two encoder layers and four decoder layers, with input sequences padded to a fixed length. In one version, the embedding layer incorporates positional encoding based on the original Transformer paper's approach [30].

2. The second model is based on an encoder-decoder architecture with Recurrent Neural Network (RNN) units (SimpleRNN, LSTM, or GRU) [8]. Here, the encoder's hidden state vector serves as the latent representation passed to the decoder, which is structured similarly. The same embedding layer as in the Transformer model is applied, though without positional encoding, as the RNN units can naturally encode sequential information.

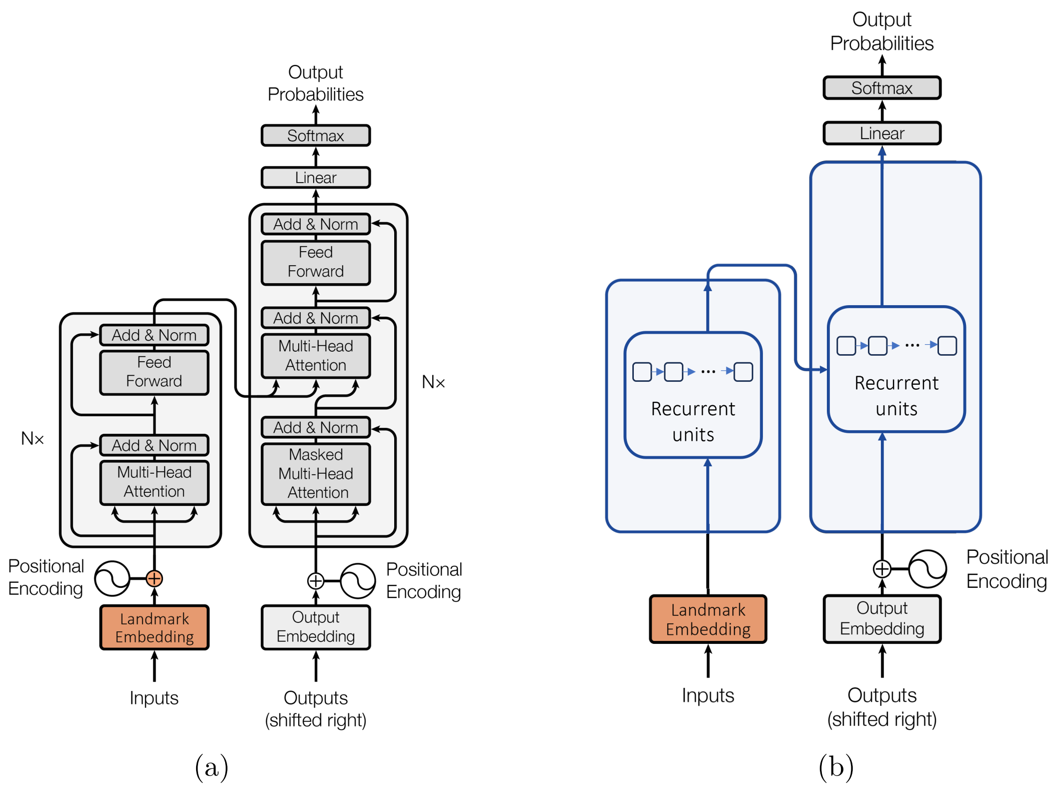

Figure 2.2: (a) Transformer Model (b) Encoder-Decoder with Recurrent Units

For training the Kaggle ASL Fingerspelling Competition dataset [4] was used. Since all signers used only one hand, the dominant one was identified with a few other body and facial key points. After normalization, the landmarks were concatenated to form the final input. The results were consistent with the findings reported in the literature [26], namely the Transformer models came out superior.

Despite performing well, in this follow-up research, I explore a more advanced approach using graph neural networks (GNNs) to generate pose embeddings. This newer approach has demonstrated superior performance in both hand pose classification and continuous fingerspelling recognition, surpassing the earlier models.

Chapter 3

Graph Neural Networks

Graph Neural Networks (GNNs) are neural networks designed to work with graph-structured data, where relationships between entities are as essential as the entities themselves. In traditional machine learning, data is often represented in grids or sequences, like images or text. However, graphs provide a way to model complex relationships, with nodes (vertices) representing entities and edges representing connections between them. GNNs extend the power of neural networks to this relational data by learning from both the features of nodes and the graph structure.

A natural application of GNNs is topic classification for academic papers, leveraging not only the text content but also the attributes of cited papers, which often share similar topics. The concept of representing pose information as a graph—where nodes denote key points and edges represent anatomical connections—has similarly intrigued me for a long time. Because GNNs excel at capturing relationships in structured data, they offer a promising approach not only for document classification but also for modeling complex spatial relationships in fields such as pose estimation and sign language translation. This fascination led me to develop a novel GNN-based embedding layer aimed at enhancing sign language translation accuracy. In this chapter, I provide a detailed technical overview of this approach, forging a strong foundation for the insights and analyses that follow.

3.1 Intuition

An intuitive application of GNNs would be to classify the topic of papers using not only the content but also the cited papers' properties as their topics are likely to be similar. GNN is a general umbrella term for architectures specifically designed to exploit the relational nature of graph datasets. One of the most common architectures is Graph Convolutional Networks (GCNs), which try to mimic regular CNN layers for graphs. The conceptual foundation of GNNs can be understood through analogy with convolutional neural networks (CNNs), a standard choice in image processing. In CNNs, a filter of arbitrary size slides across an image, calculating the sum of elementwise products between filter values and corresponding pixel values. With automatic differentiation and backpropagation, CNNs learn to extract meaningful features from images based on a specified objective function. Extending this principle to graphs, an image can be conceptualized as a graph structure, with each pixel representing a node and edges linking adjacent pixels. Nodes have different values (features) associated with them, like in the case of images the RGB values. GNNs enable localized feature extraction from graph-structured data by generalizing this approach, facilitating efficient learning of complex relationships within arbitrary graphs.

Ideal properties of such graph convolution layer include:

1. **Learnable parameters**: Parameters that can be optimized during training to capture meaningful features.
2. **Computational and storage efficiency**: The layer should be efficient enough to support iterative applications within deep networks.
3. **Localization**: Focused processing of nearby nodes, as these typically contain more relevant information.
4. **Flexible node weighting**: Different nodes should be weighted based on their significance in the context of the graph.

Applying such a layer will result in a new graph with a different set of node features and with deep learning the models can learn to create meaningful representations of each node. From here the most common operation is to predict some property from these "latent" feature vectors on the node level. Still, for the purpose of this study, it is much more useful to aggregate the node-level data to gather information on the whole graph, from which for example the classification of the gestures can happen.

3.2 Formalizing The Problem

A graph can be formally represented as $G = (V, E)$, where V is the set of nodes (or vertices) and $E \subseteq V \times V$ is the set of edges.

For a graph with N nodes, we denote the adjacency matrix by $\mathbf{A} \in \mathbb{R}^{N\times N}$, where each entry $\mathbf{A}_{ij}$ indicates the presence (or weight) of an edge between nodes i and j. Node features are represented by a matrix $\mathbf{X} \in \mathbb{R}^{N\times F}$, where F is the number of features per node. Edge-level features are ignored for this study.

The adjacency matrix $\mathbf{A}$ for our purposes (undirected, unweighted graph) is defined as:

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } i \leftrightarrow j \\ 0, & \text{otherwise} \end{cases}$$

```
Note:
A simple neural network applied directly to the adjacency and feature matrices is not suitable for
    graph data due to two key limitations:

   1. **Varying Graph Sizes**: Different graphs have different numbers of nodes, resulting in
    adjacency and feature matrices of varying shapes. Standard neural networks require fixed-size
    inputs, making them ill-suited for graphs of arbitrary size.

   2. **Lack of Node Permutation Invariance**: A neural network’s output depends on the order of
    nodes in the input matrices, but the graph structure should remain unchanged regardless of node
    ordering. Simple neural networks cannot handle this permutation invariance, resulting in
    different outputs for the same graph with varied node arrangements.
```

Given a graph G with adjacency matrix $\mathbf{A}$ and node features $\mathbf{X}$, the objective of GNNs is to learn a function $f : \mathbb{R}^{N\times F} \to \mathbb{R}^{N\times O}$, where O is the dimension of the output feature space. The iterative application of f across layers can be expressed as:

$$\mathbf{H}^{(i+1)} = f^{(i)}\left(\mathbf{H}^{(i)}, \mathbf{A}\right) \tag{3.1}$$

The initial node features are represented by $\mathbf{H}^{(0)} = \mathbf{X}$, with $f^{(i)}$ as the update function applied at layer i. This iterative formula allows each layer $f^{(i)}$ to progressively extend each node's "receptive field"—the set of nodes from which it gathers information. With each new layer, nodes gain a broader view, integrating insights from increasingly distant neighbors. This gradual expansion enables the model to learn representations that capture both local and global graph structures—depending on the number of layers—enhancing its ability to recognize complex patterns across the graph.

3.3 Message Passing Networks

To exploit localization, which is one of the most important criteria in the requirements presented in Section 3.1, we focus on how a node i communicates with its immediate neighbors. During the so-called "message passing" process, each node i gathers information from its neighbors $j \in \mathcal{N}(i)$ through a series of steps:

1. **Message Generation**: Each neighboring node j computes a message m_{ij} intended for node i. This message can be derived from its feature vector $\mathbf{x}_j$, the feature vector of node i, and, in certain specialized scenarios, edge features. The formula for message generation can be expressed as:

$$m_{ij} = \phi(\mathbf{x}_j, \mathbf{x}_i) \tag{3.2}$$

   where ϕ represents a learnable function, which could be a linear transformation, a multi-layer perceptron (MLP), or even a more complex operation depending on the specific architecture.

2. **Aggregation**: After generating messages from all neighboring nodes, node i aggregates these incoming messages. The aggregation function AGG combines the messages in a way that maintains permutation invariance:

$$m_i = \mathrm{AGG}_{j \in \mathcal{N}(i)}(m_{ij}) \tag{3.3}$$

3. **Update**: Finally, node i updates its feature vector by incorporating the aggregated message. This step typically involves a learnable function that merges the aggregated message with the node's current feature vector, represented as:

$$\mathbf{h}_i = g(m_i, \mathbf{x}_i) \tag{3.4}$$

   where g is another learnable function, which could vary in form—ranging from a simple linear transformation to a more intricate neural network structure.

Through this localized message-passing process, each node effectively captures and integrates the features of its local neighborhood. This mechanism facilitates the learning of representations that are sensitive to the graph's structure, enabling the model to generalize effectively across various graph topologies.

However, in its general form, this approach is often limited to smaller graphs and complex problems involving edge features, as certain aspects can frequently be simplified to enhance computational efficiency while still achieving comparable performance. By strategically reducing complexity, we can leverage the strengths of message passing in larger and more intricate graph structures.

3.4 Graph Convolution Networks

A common approach for transforming the general message-passing process presented in Section 3.3 is to compute the messages by applying linear transformations to both the sender node's feature vectors and also to the receiving node's as well. It is also common practice to introduce non-linearity to the system.

$$\mathbf{h}_i = \sigma\left(\mathrm{AGG}_{j\in\mathcal{N}(i)}\left(\mathbf{W}\mathbf{x}_j\right) + \mathbf{W}'\mathbf{x}_i\right) \tag{3.5}$$

where:

- $\mathbf{W}$ and $\mathbf{W}'$ are learned weight matrices, which are often equal,
- σ is an activation function that introduces non-linearity to the model.

If the same linear transformation is applied to both x_i and $x_j forjin\mathcal{N}(i)$ ($\mathbf{W} = \mathbf{W}'$), while taking the average of transformed node features we arrive at the concept of graph convolution layers:

$$\mathbf{h}_i = \sigma\left(\frac{1}{|\hat{\mathcal{N}}(i)|}\sum_{j\in\hat{\mathcal{N}}(i)} \mathbf{W}\mathbf{x}_j\right) \tag{3.6}$$

where $\hat{\mathcal{N}}(i)$ represents the neighborhood of the ith node, with an added self loop.

Expressing this in matrix form allows us to rewrite the graph update rule from Equation 3.1 as:

$$\mathbf{H}^{(i+1)} = \sigma\left(\hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{H}^{(i)}\mathbf{W}^{(i)}\right) \tag{3.7}$$

where:

- $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops added, where $\mathbf{I}$ is the identity matrix.
- $\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$, defined as: $\hat{\mathbf{D}}_{ii} = \sum_j \hat{A}_{ij}$ which sums the elements of the i-th row of $\hat{\mathbf{A}}$ to determine the degree of node i.

An interesting additional thing that Thomas N. Kipf, Max Welling did in the original paper [19] is to use "symmetric normalization". Instead of calculating the mean they normalized the adjacency matrix by the inverse square root of the degree matrix as in Equation 3.8.

$$\mathbf{H}^{(i+1)} = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(i)}\mathbf{W}^{(i)}\right) \tag{3.8}$$

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\hat{\mathcal{N}}(i)} \frac{1}{\sqrt{|\hat{\mathcal{N}}(i)||\hat{\mathcal{N}}(j)|}}\mathbf{W}\mathbf{x}_j\right) \tag{3.9}$$

The effect of this approach is further illustrated in Equation 3.9. Intuitively, this technique ensures that the message from each neighbor is scaled by the number of connections it has, weighing each neighbor's influence also according to its connectivity. This GCNConv layer is one of the most popular approaches when it comes to GNNs.

3.5 Graph Attention Networks

Graph Attention Networks [31] or GAT for short, boast the ability to learn which nodes are important when performing message passing. This is especially appealing in terms of treating the hands as graphs, from which the model can learn anatomical details based on connected joints.

The main difference between GCNConv and Graph Attention Network (GAT) lies in how they aggregate information from neighboring nodes in a graph. GAT introduces an attention mechanism that allows the model to assign different importance, or "attention scores," to different neighbors during the aggregation. Each neighbor's contribution is weighted dynamically based on learned attention coefficients, enabling GAT to focus more on influential nodes and less on irrelevant ones. While GCNConv is simpler and computationally efficient, GAT provides more flexibility by learning to emphasize specific neighbors, which can improve performance in cases where certain connections are more informative than others. So compared to GCNConv it essentially changes how the latent node features are calculated in Equation 3.6 to:

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\hat{\mathcal{N}}(i)} \alpha_{ij}\mathbf{W}\mathbf{x}_j\right) \tag{3.10}$$

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_i \parallel \mathbf{W}h_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_i \parallel \mathbf{W}h_k\right]\right)\right)} \tag{3.11}$$

where:

- $\parallel$ denotes concatenation,
- $\mathbf{a}$ represents the learnable parameters for a shared attention mechanism.

By having access to the transformed features of the two nodes, it can learn their relevance based on their "content". Note that without applying the special $softmax$ at the end each node could "attend" to any other, essentially throwing away the structure of the graph.

Chapter 4

Proposed Methodology

Building upon the promising results of the model from my previous study, this chapter examines potential enhancements in its performance. Recent literature [26] indicates that transformer-based architectures with pose approximation achieve notable success in the task of sign language translation, motivating me to retain this approach. However, since completing the initial study, I have been intrigued by the potential of representing hand movement data as a graph rather than as simple vectors. Leveraging the natural anatomical structure, where joints are interconnected through bones and ligaments, offers an intuitive way to capture dynamic interactions. This chapter will delve into the limitations observed with the convolution-based embedding layer used previously and present a new Graph Neural Network (GNN) architecture aimed at addressing these issues. Additionally, I will discuss the technological challenges encountered in developing and optimizing this GNN-based approach.

4.1 Embedding Layers for Sign Language Translation

The problem of creating a dense vector representation of pose approximation data is an important part of all ASL translation subtasks. In the case of static FS, a single dense vector representation of the pose skeleton is enough.

It is a bit more complicated for dynamic fingerspelling as in sequential neural machine translation (NMT), token embeddings play a crucial role in transforming input text into a format suitable for processing by neural networks. Each token (usually a few letters or a word) in the input sentence is mapped to a continuous vector representation, known as an embedding, which captures semantic and syntactic information about the token. This embedding process is essential because it enables the model to interpret tokens in terms of their meaning and relationships, rather than as isolated symbols. By representing tokens in a dense, lower-dimensional space, embeddings allow the model to generalize across similar words and phrases, improving its ability to handle unseen vocabulary and context variations. In transformer-based NMT models, embeddings also include positional encoding, allowing the model to recognize the order of words, which is critical for capturing sentence structure. Through these embeddings, the NMT system can more effectively learn patterns in language, leading to translations that are more accurate and fluent.

In traditional NLP and neural machine translation (NMT), as described in the original paper [30], tokens are typically associated with a unique index, allowing the embedding layer to map a sequence of indices to a sequence of dense vectors. Formally, this involves defining a function $f : \mathbb{Z}^m \to \mathbb{R}^{m \times d}$, where m is the sequence length and d is the embedding dimension, transforming each token i into an embedding vector $f(i) \in \mathbb{R}^d$.

However, for sequential pose estimation data, the embedding function f must map a sequence of feature vectors to embedding vectors, rather than discrete token indices. Here, $f : \mathbb{R}^{m \times 3n} \to \mathbb{R}^{m \times d}$, where each element in the sequence is a feature vector containing x, y, z coordinates of n selected key points. This approach adds complexity as f must not only be computationally efficient but also preserve the semantic structure of the data. Maintaining this structure is crucial, as these feature vectors capture the physical arrangement and relationships of key points in space. Balancing computational efficiency with the need to encode the spatial and temporal dependencies of pose data presents unique challenges not found in traditional NMT.

4.2 Convolutional Embedding For Sequential Pose Data

As discussed in the previous (4.1) section, the input data has a shape of $X \in \mathbb{R}^{T \times 3n}$, where T represents the number of time steps, and $3n$ represents the x, y, z coordinates for n key points per time step, arranged as a flattened vector for each time frame. In my previous study, I applied a 1D convolution along the time dimension with "same padding" to maintain the temporal dimension T in the output. Let $W \in \mathbb{R}^{k \times 3n \times h}$ represent a convolutional kernel with width k, which applies across the time dimension, and h is the number of filters or output channels. The output $Y \in \mathbb{R}^{T \times h}$ will thus have the same temporal dimension T but h channels.

With same padding, padding is applied to the input along the time axis so that the output has the same length as the input. For a kernel of width k, the padding is calculated: $p = \frac{k-1}{2}$

Assuming k is odd, the convolution with same padding for each time step t in the output Y can be represented as:

$$Y(t, c) = \sum_{i=0}^{k-1} \sum_{j=1}^{3n} W(i, j, c) \cdot X(t + i - p, j)$$

where:

1. $t \in [0, T - 1]$ indexes the time step,
2. $c \in [1, h]$ indexes the output channels,
3. $W(i, j, c)$ is the weight of the i-th position in the kernel for input dimension j and output channel c,
4. $X(t + i - p, j)$ refers to the padded input at time step $t + i - p$ and dimension j.

This convolutional setup ensures that the temporal dimension of Y matches T, while transforming each input time step into an h-dimensional output, capturing temporal features across the pose data's spatial dimensions.

There are a few interesting problems with this setup:

1. **Semantic mismatch with traditional positional encoding**: In standard NLP and NMT, positional encoding directly assigns a unique representation to each to-

ken's position in the sequence, ensuring the model understands the order of information. However, with convolution, the resulting embedding at each time step is a function of a window of (k) input time steps. This means that the positional information encoded in the final embedding is not a direct mapping of the input sequence's order, but rather a weighted combination of (k) consecutive time steps.

2. **Handling padding**: Not all sequences are the same length, padding the shorter ones introduces not only a single padding character per time step but a vector of them. This makes it harder for the model to learn where the end of the sequence is because due to the length of the kernel at the end of the sequence padding characters will affect the weighted sum, resulting in a "smoothing" effect. Applying masking in the encoder is also not a trivial task, as it's not a 1-1 mapping between the feature and embedding vector.

3. **Missing Points**: When the hands are cut off by the video frame, or in case of detection failure there are potentially missing values for certain joints. Handling this with rigid input sizes, required by convolutional layers, leaves only to "pad" those missing values. Note that this type of intra-frame padding is different from padding sequences to a fixed length in the time domain discussed in the previous point. Graphs on the other hand allow for a less rigid structure with potentially missing nodes (joint coordinates), but this also comes with a trade-off, because several optimization strategies could rely on having uniform graphs. This effect is especially prominent in the case of batching graph data for training discussed in Section 4.3.2. Nevertheless, the hypothesis that handling damaged data points with graphs leads to better performance, is definitely with exploring in the future.

4.3 Graph Neural Network-Based Embedding

With this new component, I aimed to fix the semantic problems with the convolution-based embedding layer, while also utilizing the concealed information in the structure of the data. I set off with a bottom-up design starting with data from a single time frame. My ultimate goal was to enhance sequential translation, but this first step provided an opportunity to validate my idea before over-investing myself. Preliminary literature [18] [24] highlighted several ways of dealing with distant nodes in the whole skeleton graph, but wanting to isolate the changes in the embedding layers, I decided to only use the dominant hand of the signers. Luckily this is not restricting in the case of ASL FS as conveyed meaning is deducable by solely looking at the hands.

4.3.1 Case of Isolated Data Point

First I constructed a graph using the 21 MediaPipe key points of the dominant hand with 3 features (x,y,z) for each node and connected them anatomically with the additional connection of the neighboring knuckles as they can't move independently from each other. Thanks to the flexibility of PyTorch Geometric [13], the structure of the graph could later be modified for experimenting.

I used the GNN layers described in Chapter 3. These layers essentially update the "state" of the graph, by iteratively updating the feature vectors of each node. Multiple layers were stacked to experiment with different receptive field sizes. As for the GATConv layers, there is an option to increase the number of attention heads, which has the effect of running the layer with different attention weights. This could be essential due to the shared nature of

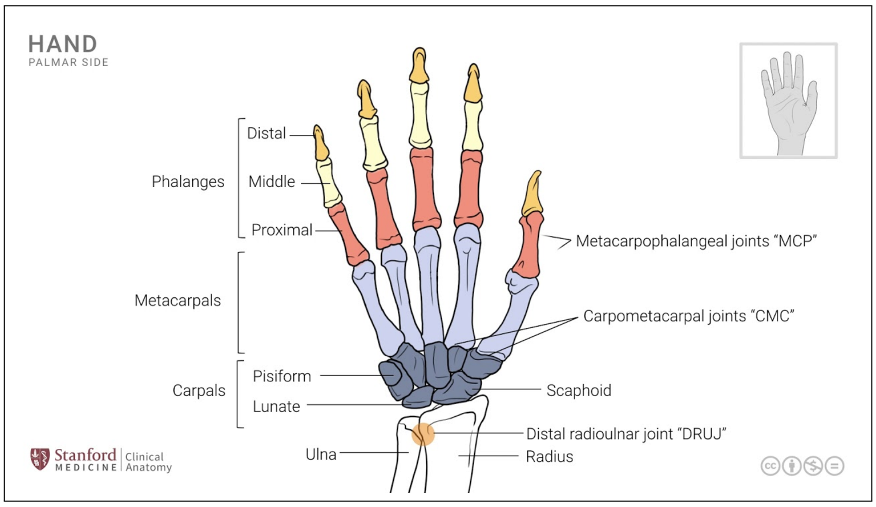
HAND
PALMAR SIDE
Phalanges
Distal
Middle
Proximal
Metacarpals
Carpals
Pisiform
Lunate
Ulna
Metacarpophalangeal joints "MCP"
Carpometacarpal joints "CMC"
Scaphoid
Distal radioulnar joint "DRUJ"
Radius
Stanford MEDICINE Clinical Anatomy

Figure 4.1: Anatomy of Hand Joints.

these weights. There are two options for handling the additional data: either compute their mean or concatenate them and apply an optional downscale transformation to retain the original dimensions as seen in the multi-head attention implementation of the Attention is All You Need paper [30]. As a final step average pooling was used to get the dense vector representation of the whole graph. For classifying the gestures any neural network can be operated on this latent space vector.

4.3.2 Sequential Data Points

Handling a sequence of graphs is a much more challenging task. Note that this is not a spatial-temporal graph as in [33] because one of the main goals with this embedding layer was to treat individual data points separately so that a one-to-one match exists between embedding and captured pose frame.

Implementing graph-level data batching presented a technical challenge, as batching for graphs differs significantly from batching regular vectors. Unlike standard vectors, graphs often vary in size, meaning their adjacency matrices and node feature vectors cannot be batched directly. PyTorch Geometric [13], a PyTorch extension for working with graphs and Graph Neural Networks (GNNs), addresses this issue with a specialized mini-batching technique, which involves the merging the graphs in the following three steps:

1. **Concatenate Node Feature Vectors**: Each graph's node feature vectors (e.g., $\mathbf{X}_i$ in Equation 4.1) are concatenated, forming a unified feature representation for the batch.

2. **Track Graph Membership**: An index array is maintained to indicate which graph each feature vector belongs to, preserving the structure and association of features within each graph.

3. **Diagonal Concatenation of Adjacency Matrices**: The adjacency matrices (e.g., $\mathbf{A}_i$ in Equation 4.1) for each graph are combined along the diagonal to form a single, large sparse matrix, effectively creating a large graph with isolated subgraphs. Using a sparse matrix reduces computational overhead.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_n \end{bmatrix} \tag{4.1}$$

With this approach, the Graph Convolutional Network (GCN) layers can compute updates validly, as node updates in the used layers rely on information solely from each node and its immediate neighbors. This is the case even though the graphs look the same in this particular problem. It could be a separate study to look into re-implementing all graph layers with support for "regular" batching, but not all layers necessarily support this type of model by nature. So in this study, PyTorch-style mini-batching was used which might sacrifice a little processing power for the ability to preserve compatibility with more layer types.

Compounding this complexity, each entry in the sequential dataset consisted of a set of graphs, meaning that sequence length also impacts the "mini-batch" size. For instance, representing each entry with 21 key points as nodes produces an adjacency matrix of size $441T^2b^2$, where T is the sequence length and b is the batch size. This matrix must be constructed for each batch, making caching valuable. Consequently, the Graph Embedding layer must be aware of the batch size, introducing a minor dependency that slightly complicates the architecture. In the end, keeping the batch size at 32 and utilizing a GPU, the training times were manageable.

Chapter 5

Static Fingerspelling

dataset: ...

- Given medium model size does preprocessing have any effect on Dense vs GNNs -> choose what's best for each. (2-3 layers for graphs whichever seems better)

- how many layers? (2-3-4 for each GNN)

- GCN vs GAT vs dense for x epochs (compare the effects of the model's hidden size -> small/medium/large)

- calculate the performance of the best models from each 3 categories (CPU and GPU as well)

- using x,y,z vs x,y (on better GNN and dense) - inverse graph (joints that are not connected in the hand are edges in the graph) (on better vs GNN)

Chapter 6

Sequential Fingerspelling

The data originates from a Kaggle competition [4] sponsored by Google. The goal was to detect and translate American Sign Language (ASL) fingerspelling into text. The dataset contains over three million fingerspelled characters, captured by over 100 deaf sign language users using smartphone cameras under various backgrounds and lighting conditions.

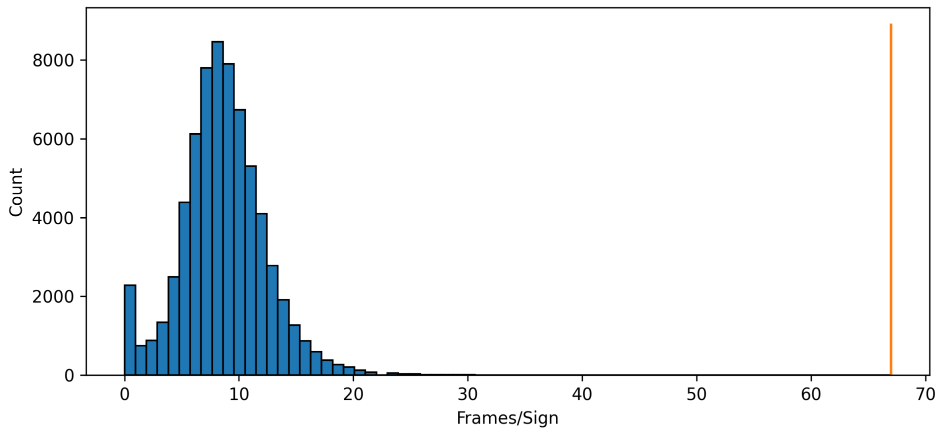

Figure 6.1: Distribution of Frames per Sign (Max: 67)

Preprocessing steps:

1. Remove the extensive number of outlier sequences as revealed by (Figure 6.1) with very few or excessive frames per sign.
2. Exclude sequences, including: ":_?=%@,&~()$'!#[*;" characters that are not part of the extended ASL Alphabet (Figure 4.1). These signs only appear in the minority of the cases and and also lack consensus on their representation.
3. Select the dominant hand, which is the one that appears more on the frames.
4. Filter frames that cut off some joints of the previously selected hand.
5. Remove sequences that don't have either fewer frames/signs than 3 or more than 20.
6. Limit dataset to sequences shorter than (small) 128 and (large) 256 frames, forming 2 separate ones.

Chapter 7

Conclusion

...

7.1 Future Work

Acknowledgements

I would like to express my sincere gratitude to my university professor, Dr. Péter Ekler, for his invaluable guidance and support throughout my research. His insights and encouragement have been instrumental in shaping my work.

I would also like to extend my heartfelt appreciation to my girlfriend for her unwavering patience and understanding during the times I sacrificed our moments together for the sake of my research. Her support has made this journey all the more meaningful.

Bibliography

[1] Kalpattu S. Abhishek, Lee Chun Fai Qubeley, and Derek Ho. Glove-based hand gesture recognition sign language translator using capacitive touch sensor. In *2016 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 334–337, 2016. DOI: `10.1109/EDSSC.2016.7785276`.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. 34, 05 2012. DOI: `10.1109/TPAMI.2012.120`.

[3] Mohamed Aktham Ahmed, Bilal Bahaa Zaidan, Aws Alaa Zaidan, Mahmood Maher Salih, and Muhammad Modi Bin Lakulu. A review on systems-based sensory gloves for sign language recognition state of the art between 2007 and 2017. 18(7):2208, 07 2018. ISSN 1424-8220. URL `https://www.mdpi.com/1424-8220/18/7/2208`. DOI: `10.3390/s18072208`.

[4] Manfred Georg Mark Sherwood Phil Culliton Sam Sepah Sohier Dane Thad Starner Ashley Chow, Glenn Cameron. Google - american sign language fingerspelling recognition. `https://kaggle.com/competitions/asl-fingerspelling`, April 2023.

[5] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus. abs/1907.05047, 2019. URL `http://arxiv.org/abs/1907.05047`.

[6] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. Blazepose: On-device real-time body pose tracking. abs/2006.10204, 2020. URL `https://arxiv.org/abs/2006.10204`.

[7] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. abs/1812.08008, 2018. URL `http://arxiv.org/abs/1812.08008`.

[8] Jonte Dancker. A brief introduction to recurrent neural networks. `https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4`, April 2022.

[9] Sunanda Das, Md. Samir Imtiaz, Nieb Hasan Neom, Nazmul Siddique, and Hui Wang. A hybrid approach for bangla sign language recognition using deep transfer learning model with random forest classifier. 213:118914, 2023. DOI: `10.1016/j.eswa.2022.118914`.

[10] Giulia Zanon de Castro, Rúbia Reis Guerra, and Frederico Gadelha Guimarues. Automatic translation of sign language with multi-stream 3d cnn and generation of artificial depth maps. 215:119394, 2023. ISSN 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2022.119394`.

[11] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. volume 2749, pages 363–370, 06 2003. ISBN 978-3-540-40601-3. DOI: `10.1007/3-540-45103-X_50`.

[12] Pooya Fayyazsanavi, Negar Nejatishahidin, and Jana Košecká. Fingerspelling posenet: Enhancing fingerspelling translation with pose-based transformer models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, pages 1120–1130, January 2024.

[13] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019. URL `http://arxiv.org/abs/1903.02428`.

[14] Google. Mediapipe holistic. `https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md`, November 2023.

[15] Google. Google - american sign language fingerspelling recognition, 2023. URL `https://www.kaggle.com/c/asl-fingerspelling`. Accessed: 2023-12-01.

[16] Maksym Ivashechkin, Oscar Mendez, and Richard Bowden. Improving 3d pose estimation for sign language. `https://arxiv.org/abs/2308.09525`, 2023.

[17] Songyao Jiang, Bin Sun, Lichen Wang, Yue Bai, Kunpeng Li, and Yun Fu. Skeleton aware multi-modal sign language recognition. abs/2103.08833, 2021. URL `https://arxiv.org/abs/2103.08833`.

[18] Jichao Kan, Kun Hu, Markus Hagenbuchner, Ah Chung Tsoi, Mohammed Bennamoun, and Zhiyong Wang. Sign language translation with hierarchical spatio-temporalgraph neural network. abs/2111.07258, 2021. URL `https://arxiv.org/abs/2111.07258`.

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

[20] Jiyang Li, Lin Huang, Siddharth Shah, Sean J. Jones, Yincheng Jin, Dingran Wang, Adam Russell, Seokmin Choi, Yang Gao, Junsong Yuan, and Zhanpeng Jin. Signring: Continuous american sign language recognition using imu rings and virtual imu data. 7(107), 9 2023. URL `https://doi.org/10.1145/3610881`. DOI: `10.1145/3610881`.

[21] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. 18(185):1–52, 2018. URL `http://jmlr.org/papers/v18/16-558.html`.

[22] H. Lutzenberger. Manual and nonmanual features of name signs in kata kolok and sign language of the netherlands. 18(4):546–569, 2018. URL `https://www.jstor.org/stable/26637448`. Accessed: 2023-12-01.

[23] Abu Saleh Musa Miah. Hand gesture recognition for multi-culture sign language using graph and general deep learning network. PP:1–12, 02 2024. DOI: `10.1109/OJCS.2024.3370971`.

[24] Abu Saleh Musa Miah, Md. Al Hasan, Satoshi Nishimura, and Jungpil Shin. Sign language recognition using graph and general deep neural network based on large scale dataset. PP:1–1, 01 2024. DOI: `10.1109/ACCESS.2024.3372425`.

[25] Amit Moryossef, Ioannis Tsochantaridis, Joe Dinn, Necati Cihan Camgöz, Richard Bowden, Tao Jiang, Annette Rios, Mathias Müller, and Sarah Ebling. Evaluating the immediate applicability of pose estimation for sign language recognition. abs/2104.10166, 2021. URL https://arxiv.org/abs/2104.10166.

[26] Nada Shahin and Leila Ismail. From rule-based models to deep learning transformers architectures for natural language processing and sign language translation systems: survey, taxonomy and performance evaluation. 57(271), 08 2024. DOI: 10.1007/s10462-024-10895-z.

[27] H. D. W. Stiles. A brief history of bsl, 2012. URL https://blogs.ucl.ac.uk/library-rnid/2012/07/06/a-brief-history-of-bsl/. Accessed: 2023-12-01.

[28] Gallaudet University. American sign language & french sign language. URL https://gallaudet.edu/museum/history/american-sign-language-and-french-sign-language/. Accessed: 2023-12-01.

[29] William Woods University. Sign language around the world, 2016. URL https://asl-blog.williamwoods.edu/2016/01/sign-language-around-the-world/. Accessed: 2023-12-01.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

[32] Ronghui Wu, Sangjin Seo, Liyun Ma, Juyeol Bae, and Taesung Kim. Full-fiber auxetic-interlaced yarn sensor for sign-language translation glove assisted by artificial neural network. 14(1):139, 07 2022. ISSN 2150-5551. URL https://doi.org/10.1007/s40820-022-00887-5. DOI: 10.1007/s40820-022-00887-5.

[33] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. abs/1801.07455, 2018. URL http://arxiv.org/abs/1801.07455.

[34] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. abs/2006.10214, 2020. URL https://arxiv.org/abs/2006.10214.

[35] Lijuan Zhou, Bin Zhao, Jingye Liang, Fangying Lu, Weiping Yang, Jishuai Xu, Jingxuan Zheng, Yong Liu, Run Wang, and Zunfeng Liu. Low hysteresis, water retention, anti-freeze multifunctional hydrogel strain sensor for human–machine interfacing and real-time sign language translation. 11:3856–3866, 2024. URL http://dx.doi.org/10.1039/D4MH00126E. DOI: 10.1039/D4MH00126E.