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

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 [37] 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 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 diversely and naturally. 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 [34], allowing them to interact with smart devices in their 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. 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 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 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. This is because, ASL is rooted in French Sign Language [33], while BSL evolved independently [32]. Furthermore, like spoken languages, sign languages also have dialects and regional variations, which adds further complexity to the development of intelligent solutions.

1.2 American Sign Language

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 US [33]. 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 hand shapes represent letters. In some sign languages, FS is commonly used to convey names, foreign words, or specialized terms that don't have their own signs. In other languages, fingerspelling 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 [26], 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.

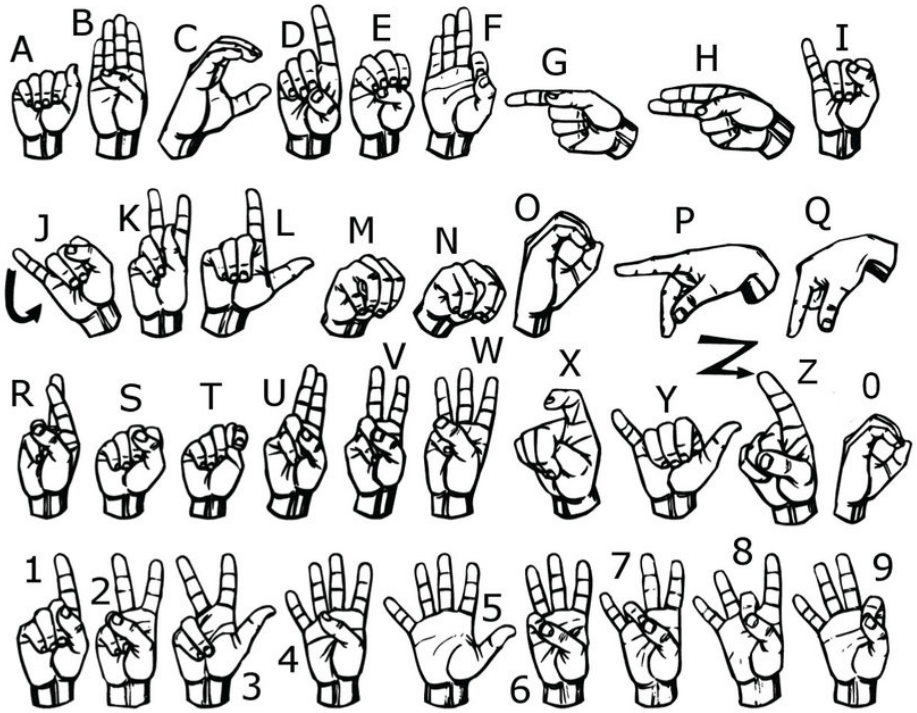

Figure 1.1: Extended ASL Alphabet.[1]

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 phones. 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 36 words per minute in the US [5]. In ASL, the alphabet and numbers are communicated

[1]Source: `www.lifeprint.com/asl101/fingerspelling`

using just one hand, unlike BSL, which requires both hands. While other gestures, like head movements or leaning forward, can add emphasis during fingerspelling, the letters themselves can be recognized solely by observing the dominant hand. FS thus serves as a useful foundation for translating more complex gestures that rely 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, capturing temporal information to construct complete sentences.

A third dimension in the taxonomy is grammar [31], 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 hand shapes 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.

Table 1.1: Applying Taxonomy on ASL Subtasks

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

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.

1.5 About This Work

This study focuses on translating fingerspelled characters and sequences, building on my previous work presented in an earlier Scientific Students' Association Report, where I developed and trained models for the task of Static and Sequential Fingerspelling [11]. This time, I have delved deeper into the representation of gestures as dense vectors using Graph Neural Networks, achieving significant performance improvements compared to the earlier Multi-Layered Perceptron and Convolution-based embedding models. For further details on the former approach, refer to Section 2.3.

The rest of the study is organized as follows:

1. Chapter 2 reviews related work, focusing on foundational approaches in signal processing and translation algorithms essential for translating environmental inputs into model-ready data.

2. Chapter 3 builds the mathematical foundation for Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs), providing a detailed technical overview of these models, which serve as a basis for later innovations.

3. Chapter 4 presents the proposed methodology, introducing a novel GNN-based embedding model specifically designed to overcome limitations observed with the convolutional embedding layer from the previous study.

4. Chapter 5 examines static fingerspelling from a graph-based perspective, validating the concept through isolated gesture recognition.

5. Chapter 6 extends this to sequential fingerspelling, evaluating the GNN embedding layer's impact within transformer architectures for continuous translation tasks.

6. Finally, Chapter 7 concludes the study, summarizing key findings, discussing the superiority of GNN-based models over earlier techniques, and suggesting future research directions in graph-based gesture recognition and sign language 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 deployed in real-world settings.

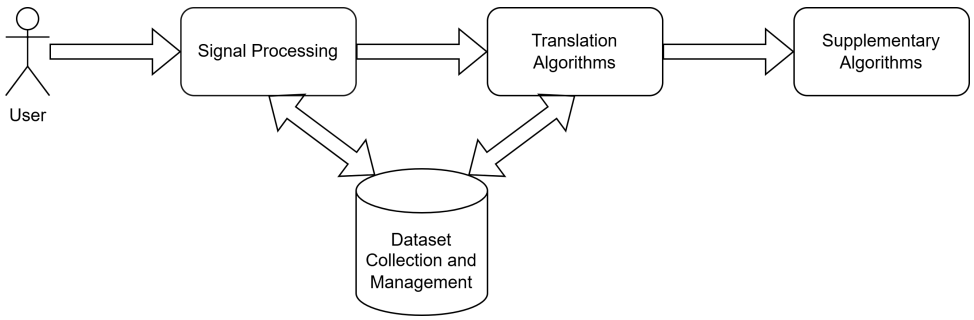

Figure 2.1: General Architecture

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 whole process. 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 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 [12], optical flow calculation [14], or artificially generated depth maps [13], 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 [24]. More traditional methods include tracking the exact position of fingers with gloves [1][38][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. Uses 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 [9] and Mediapipe Holistic [18]. The latter is a composition of 3 separate models: BlazePose [8], MediaPipe Hands [41] and BlazeFace [7]. 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 [41] by the authors. Intuitively this becomes crucial when gestures involve intertwined fingers, favoring Mediapipe Holistic, but a comparative study [29] with two separate teams in agreement, concluded that: "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. Additionally, efforts are underway to develop models specifically for sign language recognition. For instance, research conducted at the University of Surrey [19] has shown promising results, outperforming MediaPipe across several datasets. This model also enhances 3D performance through a specialized layer that predicts joint angles, enabling the calculation of depth with the propagation of position and rotation from a chosen 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 [27][15][43], and there are examples where combining different type of processing methods like optical flow, depth maps and "pose skeleton" as a spatial-temporal graph [20] yields the best results.

It must be noted, however, that 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 [29] 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 [31], 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 also relevant to this research:

1. PoseNet [15] is a Transformer [35] 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 [18], 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 when 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 [28] 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 [40] (ST-GCN). Each video frame has its isolated graph and the corresponding points on consecutive frames are connected, hence the naming. 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. In the original study [40] 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 position of distant body parts. Some signs only differ in the position of the face that is touched. The authors [28] solve this problem by using only a handful of selected key points and constructing their fully connected graph representation.

3. Another approach to this problem is hierarchical spatial-temporal graphs [22]. In this method, graphs are constructed across multiple levels: fine-level graphs represent individual body parts, such as hands and face, while two high-level graphs are constructed with nodes whose features are derived from the bounding boxes around these body parts using optical flow and feature extraction from RGB frames. To process these graphs, specialized GNN layers are applied iteratively. First, features from the fine-level graphs are pooled to construct a new high-level graph, which then undergoes further processing. After several iterations, the graphs are mean-pooled and the latent representation at each time step is concatenated yielding a sequence of dense vectors. The final stage involves a two-step decoding process with LSTM layers: the first LSTM predicts signs at each time step, while a second also LSTM-based encoder-decoder refines these predictions, generating the final output sequence.

4. The following study [27] showcases that graph neural networks can also be applied to images as well. The research presents a dual-stream approach utilizing ResNet-style and graph convolutional layer-based feature extraction. For the latter, the SLIC superpixel [2] algorithm is applied, which 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 the dense vector representation, 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 the task of static fingerspelling, the focus was on handling pose information and building an efficient model. Pose data was captured using the MediaPipe Hands-specific model [41]. Preprocessing involved scaling, centering, and normalizing the x and y coordinates to enhance consistency across samples. Furthermore, I applied data augmentation techniques, including affine transformations, to improve model robustness. A Multi-Layered Perceptron (MLP), optimized with the Hyperband algorithm [25], 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 [35], with modifications to the embedding layer for handling pose data. (More on that in Section 4.2.) The best architecture includes two encoder layers and four decoder layers, with input sequences padded to a fixed length. The embedding layer incorporates positional encoding based on the original Transformer paper's approach [35].

2. The second model is based on an encoder-decoder architecture with Recurrent Neural Network (RNN) units (SimpleRNN, LSTM, or GRU) [10]. 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.

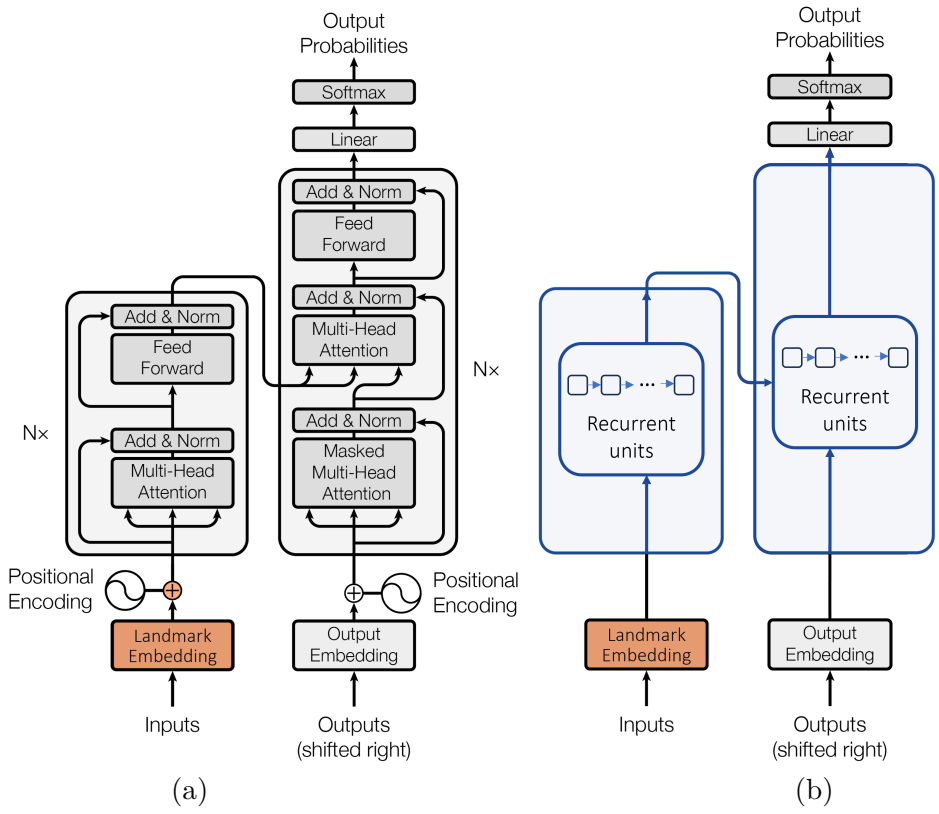

Figure 2.2: (a) Transformer Model (b) Encoder-Decoder with Recurrent Units

For training the Kaggle ASL Fingerspelling Competition dataset [5] 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 [31], 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 deep learning models 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.

The concept of representing pose information as a graph, where nodes denote key points and edges represent anatomical connections, has intrigued me for a long time. Because GNNs excel at capturing relationships in structured data, they offer a promising approach for modeling complex spatial relationships. 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 later sections.

3.1 Intuition

GNN is a general umbrella term for architectures specifically designed to exploit the relational nature of datasets. One of the most common architectures is Graph Convolutional Networks (GCNs), which expand the idea of regular CNN layers for graphs. In CNNs, a filter of arbitrary size slides across an image, calculating the sum of element-wise 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.

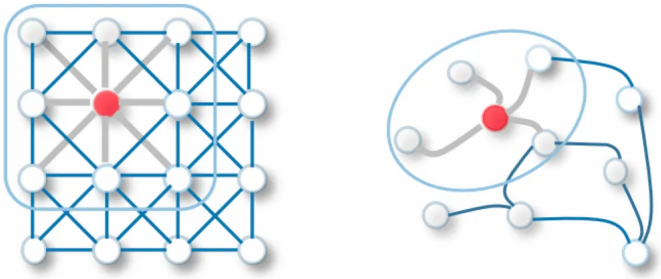

Figure 3.1: Illustration of 2D Convolution and Graph Convolution[39]

Properties of the ideal 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 the most 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 a meaningful representation for each node. From here the most common operation is to predict some property from these "latent" feature vectors on the node level. Still, for 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.

In a graph with N nodes, the features are represented by a matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, where F denotes the number of features per node. Each row vector, denoted as $\vec{x_i}$, corresponds to the features of the i-th node in $\mathbf{X}$. It is important to note that all vectors in this study are treated as row vectors, and edge-level attributes are excluded from consideration.

The adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where each entry $\mathbf{A}_{ij}$ indicates the presence (or weight) of an edge between nodes i and j. For our purposes (undirected, unweighted graph):

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } i \leftrightarrow j \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

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:

$$\mathbf{H}^{(i+1)} = f^{(i)}\left(\mathbf{H}^{(i)}, \mathbf{A}\right) \tag{3.2}$$

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 the progressive extension of each node's "receptive field", which is 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 patterns, depending on the number of layers.

Note that 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.

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 $\vec{x_j}$, the feature vector of node i, and, in certain specialized scenarios, edge features. The formula for message generation can be expressed as:

$$\vec{m_{ij}} = \phi(\vec{x_i}, \vec{x_j}) \tag{3.3}$$

   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.

$$\vec{m}_i = \mathrm{AGG}_{j \in \mathcal{N}(i)}(\vec{m}_{ij}) \tag{3.4}$$

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:

$$\vec{h_i} = g(\vec{m_i}, \vec{x_i}) \tag{3.5}$$

where g is another learnable function, which could vary in form, ranging from a simple linear transformation to a more intricate neural network structure and $\vec{h_i}$ is the updated feature vector, or i-th row of $\mathbf{H}^{(i)}$ from Equation 3.2.

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:

$$\vec{h_i} = \sigma\left(\mathrm{AGG}_{j\in\mathcal{N}(i)}\left(\vec{x_j}\mathbf{W}\right) + \vec{x_i}\mathbf{W}'\right) \tag{3.6}$$

where:

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

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

$$\vec{h_i} = \sigma\left(\frac{1}{|\hat{\mathcal{N}}(i)|}\sum_{j\in\hat{\mathcal{N}}(i)}\vec{x_j}\mathbf{W}\right) \tag{3.7}$$

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 3.2 as:

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

where:

- $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops, with $\mathbf{I}$ being 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.
- Note, that the Transposition of $\mathbf{W}^{(i)}$ is omitted, because it is learned

An interesting additional thing that Thomas N. Kipf, Max Welling did in the original paper [23] is to use "symmetric normalization". Instead of calculating the mean they normalized the adjacency matrix by the inverse square root of the degree matrix:

$$\mathbf{H}^{(i+1)} = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(i)}\mathbf{W}^{(i)}\right) \tag{3.9}$$

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\hat{\mathcal{N}}(i)} \frac{1}{\sqrt{|\hat{\mathcal{N}}(i)||\hat{\mathcal{N}}(j)|}}\vec{x_j}\mathbf{W}\right) \tag{3.10}$$

The effect of this approach is further illustrated in Equation 3.10. Intuitively this technique ensures that the message from each neighbor is scaled by the number of connections it has so that nodes with a high connectivity are not overrepresented. This GCNConv layer is one of the most popular approaches when it comes to GNNs.

3.5 Graph Attention Networks

Graph Attention Networks [36] 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 weighed 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 3.7 to:

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\hat{\mathcal{N}}(i)} \alpha_{ij}\vec{x_j}\mathbf{W}\right) \tag{3.11}$$

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{a}^T\left[\vec{h_i}\mathbf{W} \parallel \vec{h_j}\mathbf{W}\right]\right)\right)}{\sum_{k\in\hat{\mathcal{N}}(i)}\exp\left(\text{LeakyReLU}\left(\vec{a}^T\left[\vec{h_i}\mathbf{W} \parallel \vec{h_k}\mathbf{W}\right]\right)\right)} \tag{3.12}$$

where:

- $\parallel$ denotes concatenation,
- $\vec{a} \in \mathbb{R}^{2O}$ represents the learnable parameters for a shared attention mechanism. (O is the output feature dimension of the layer.)

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

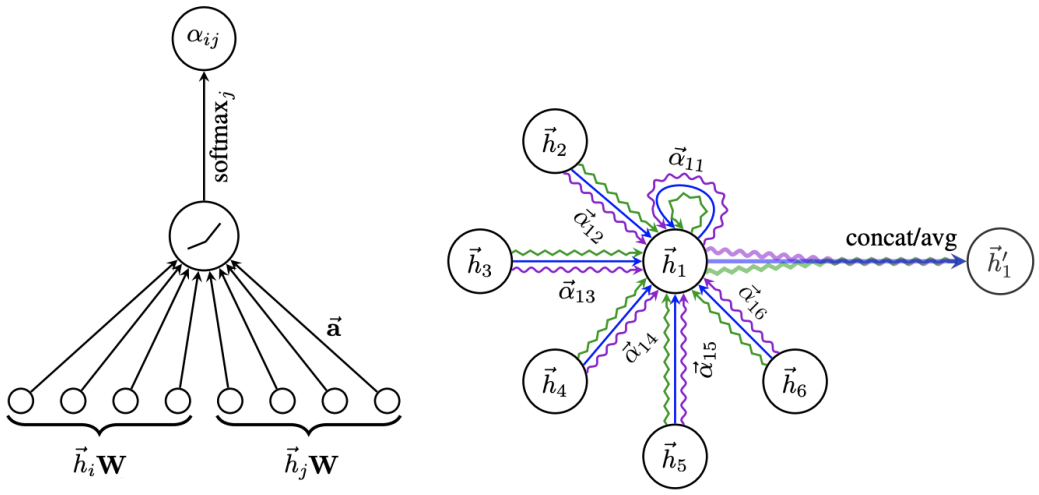

Figure 3.2: Illustration of GAT Conv Layer from The Original Study [36]: **Left:** The attention mechanism $a(\vec{h_i}\mathbf{W}, \vec{h_j}\mathbf{W})$, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2O}$, applying a LeakyReLU activation and finally a special softmax to preserve structure. **Right:** An illustration of multi-head attention (with 3 heads) on node 1 and its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Chapter 4

Proposed Methodology

Building upon the promising results of my previous study [11], this chapter examines potential enhancements in its performance. Recent literature [31], 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 simple vectors. 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 the new GNN-based approach.

4.1 Embedding Layers for Sign Language Translation

Creating dense vector representations of input is a proven method in deep learning. By mapping tokens to a compact, lower-dimensional space, embeddings enable models to capture underlying patterns, relationships, and similarities between data points. This efficient representation allows for faster learning and better generalization across diverse contexts.

In sequential neural machine translation (NMT), 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 $f_{trad} : \mathbb{Z}^t \to \mathbb{R}^{t \times d}$, where T is the sequence length and d is the embedding dimension. However, for pose estimation data, the embedding function must map a sequence of feature vectors, rather than discrete token indices. Here, $f : \mathbb{R}^{t \times 3n} \to \mathbb{R}^{t \times d}$, 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. In transformer-based Neural Machine Translation (NMT) models, embedding layers also include positional encoding [35], allowing the model to recognize the order of words, which is critical for capturing sentence structure.

4.2 Convolutional Embedding For Sequential Pose Data

In my previous study, I applied a 1D Convolution-based embedding layer. This type of architecture is suitable for encoding temporal information in the embedding vectors. The input ($X \in \mathbb{R}^{t \times 3n}$) was treated as having $3n$ features at each time step t. 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 sequence was padded evenly to the left and right to preserve the temporal dimension, with $k-1$ padding tokens. The output $Y \in \mathbb{R}^{t \times h}$ will thus have the same temporal dimension t but h channels.

$$Y(x, c) = \sum_{i=0}^{k-1} \sum_{j=1}^{3n} W(i, j, c) \cdot \hat{X}(x + i - p_{left}, j) \tag{4.1}$$

where:

1. $x \in [0, t-1]$ indexes the time step, and $c \in [1, h]$ indexes the output channels,
2. $\hat{X}$ refers to the padded input,
3. $p_{left} = \lfloor \frac{k-1}{2} \rfloor, \quad p_{right} = \lceil \frac{k-1}{2} \rceil$ representing the left and right padding.

There are a few interesting problems with this setup:

1. **Semantic mismatch with traditional positional encoding**: In transformer-based NMT, positional encoding provides each token with a unique representation based on its position in the sequence, helping the model interpret the order of information. With convolution, however, a window of k input time steps influences the embedding at each time step. This means that applying positional information isn't necessarily correct semantically.
2. **Handling padding**: In cases where sequences differ in length, padding shorter ones adds more than just a single padding token per time step, it introduces a vector of size $3n$ for each padded element. This can obscure the true end of the sequence, making it harder for the model to distinguish meaningful content due to the influence of the padding within the kernel length. At sequence ends, these padding vectors impact the weighted sum, creating a "smoothing" effect that may dilute significant features. Moreover, applying masking in the encoder isn't straightforward, as there isn't a direct one-to-one correspondence between feature values and the embedding vectors, complicating the masking process.
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 removed nodes as a potential replacement for missing joint coordinates. However, this also comes with a trade-off, because several optimization strategies could rely on having uniform graphs, which 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 with graphs leads to better performance, is worth 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 [22] [28] 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 fingerspelling 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 connections of the neighboring knuckles as they can't move independently from each other. Thanks to the flexibility of PyTorch Geometric [16], the structure of the graph could later be modified for experimenting.

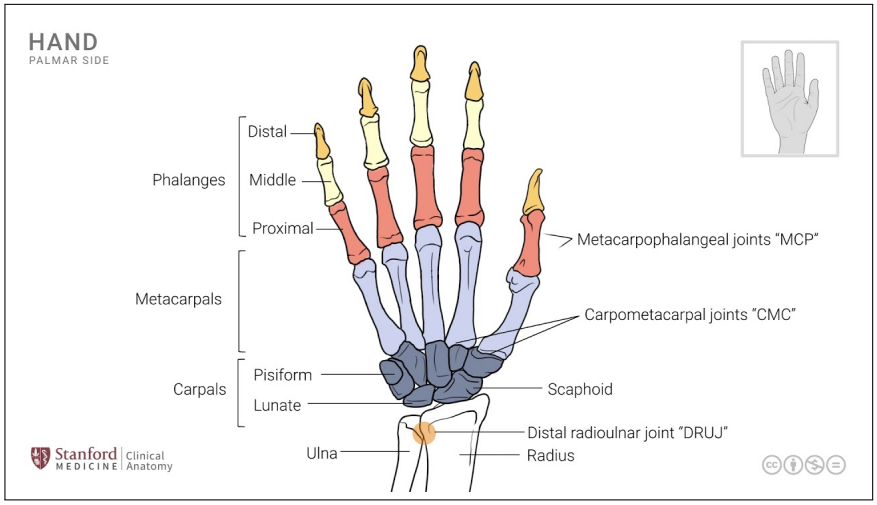

Figure 4.1: Anatomy of Hand Joints [17].

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 these parameters. There are two options for handling the additional data, that comes with the extra heads: 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 [35]. 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

The input for such a task is not treated as a spatial-temporal graph as in [40] 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 the embedding and captured pose frame. The key to keeping this property is to process each time step separately with the architecture presented in Section 4.3.1, but handling a sequence of graphs is a much more challenging task than I first thought.

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 [16], a PyTorch extension for working with graphs and Graph Neural Networks (GNNs), addresses this issue with a specialized mini-batching technique, which involves 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 4.2) 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 4.2) for each graph are combined along the diagonal to form a single, large sparse matrix, effectively creating a merged graph with isolated subgraphs. Using a sparse matrix reduces storage overhead.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_n \end{bmatrix} \tag{4.2}$$

With this approach, the Graph Neural Network (GNN) 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 graph algorithms necessarily support this type of model by nature. So in this study, PyTorch-style mini-batching was used which might sacrifice a little processing time 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, training times were manageable.

Chapter 5

Static Fingerspelling

Translating American Sign Language fingerspelling (ASL FS) from static images may initially seem unconventional, as signs often require motion to be fully understood. This limitation means the model will likely struggle with specific gestures that depend on movement, but using static images serves as a foundational step toward more complex translation tasks. Beyond sign language translation, recognizing isolated gestures has applications in gesture-based human-computer interaction, where detecting certain signs can indicate intent and changes in joint positions can drive spatial actions like dragging. Nonetheless, I used ASL FS as a way to validate the idea of interpreting pose approximation data as a graph for neural processing and as a stepping stone for sequential FS.

5.1 Dataset

In this study, I used the same dataset as in my previous research: the Sign Language MNIST [21], the University of Exeter ASL [30], and a self-recorded set of FS gestures. Due to low image quality, in the case of the first two, I applied enhancement techniques to improve conversion rates with MediaPipe Hands [41]. Super-resolution algorithms had limited impact, but the ECCV16 colorization model [42] was essential for the MNIST dataset since the pose approximation algorithm required RGB images. This approach enabled successful pose extraction for approximately 31% of the Sign Language MNIST and 39% of the University of Exeter ASL datasets.

For preprocessing, I explored a new approach aimed at addressing the issue where variations in camera focal lengths and signer distances cause hand skeletons to appear significantly different. This approach (Figure 5.1) minimizes the effect of these factors. The steps are as follows:

- Similarly to the previous study only the x,y components are processed as z only indicates relative depth. At the end, the 3 axes are stacked together again.
- First, the points are centered by shifting the coordinates so that their mean aligns with the origin, allowing for symmetric scaling around the center.
- Then, the maximum distance of any point from the origin is determined separately for both axes. This enables the calculation of independent scaling factors for each axis, effectively resizing the points to fit within a standard range along both directions.

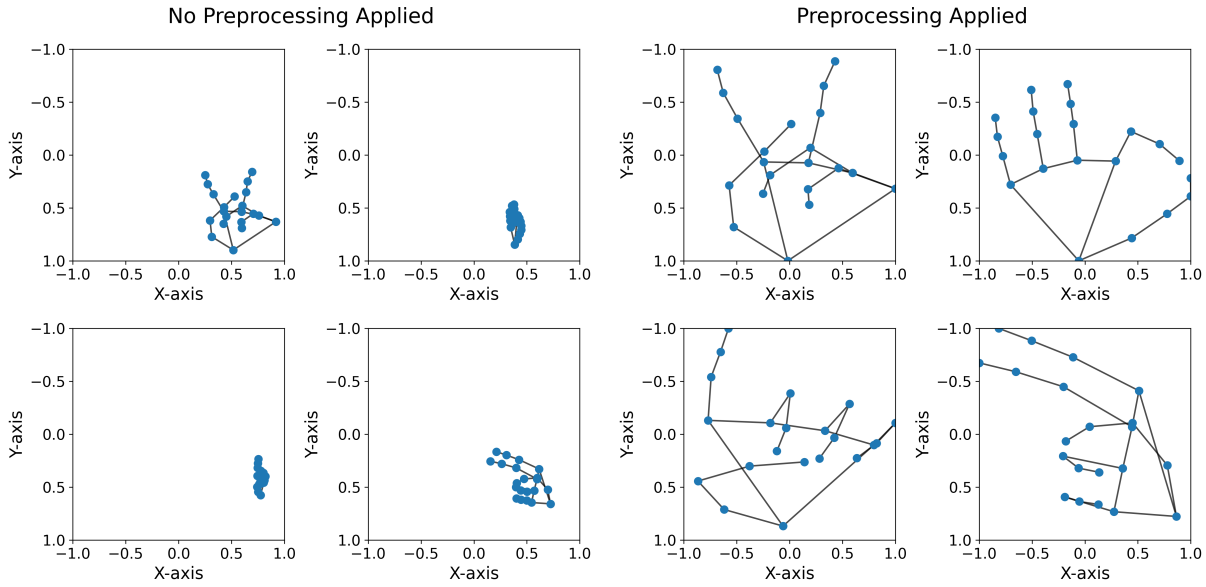

Figure 5.1: Visualization of Preprocessing Steps

5.2 Experimental Setup

To compare the performance and suitability of various GNN architectures for pose classification, I conducted several experiments with a standard 80/10/10 train-validation-test split. Initially, each GNN model was trained for 30 epochs, with a batch size of 256.

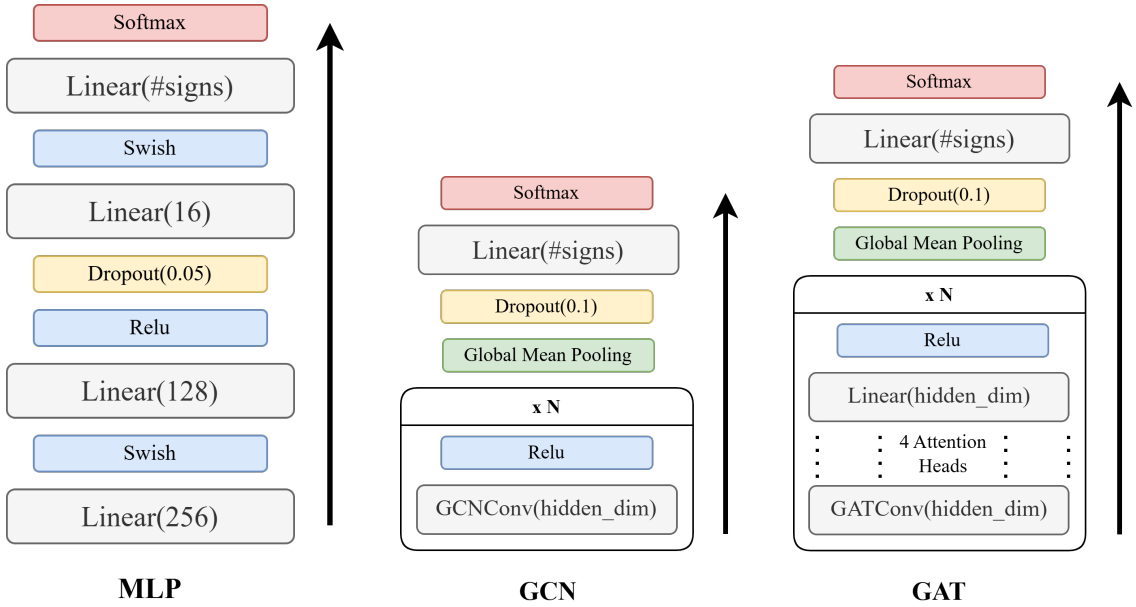

Figure 5.2: Block Diagram of Used Model Architectures

Performance was evaluated across the following configurations:

- **GNN Layer Type**: Two GNN architectures were examined, specifically GCN [23] and GAT [36]. For GAT, four attention heads were employed in each layer to stabilize the mechanism throughout the experiment.

- **Model Layers:** Different layer depths were tested, with models configured at 1, 2, and 3 layers. This setup allowed analysis of the relationship between receptive field expansion and performance.
- **Hidden Size Variations:** The hidden layer output dimensions were varied at three levels, small (32), medium (64), and large (128), to observe the effect of hidden dimensions on model performance.

After finding the best GCN and GAT-based models, their performance is compared with the MLP model from the previous study in a longer training session.

5.3 Results

First, the performance of GCN and GAT models was evaluated based on the accuracy achieved across different configurations. As shown in Table 5.1, both GNN architectures demonstrated improved accuracy with increased model depth and hidden layer size. Notably, GAT models consistently outperformed GCN models across all configurations, achieving the highest test accuracy of 96.58% with three layers and a "large hidden size". This trend suggests that the attention mechanism in GAT provides a significant advantage in capturing complex relationships within the pose classification task.

Table 5.1: Accuracies on Test Dataset for GCN and GAT Models

Model	Layers	Small	Medium	Large
GCN	1	60.00%	66.18%	70.49%
	2	81.07%	84.07%	86.89%
	3	83.90%	88.37%	90.53%
GAT	1	82.21%	89.16%	91.53%
	2	92.07%	94.76%	95.90%
	3	92.55%	95.11%	96.58%

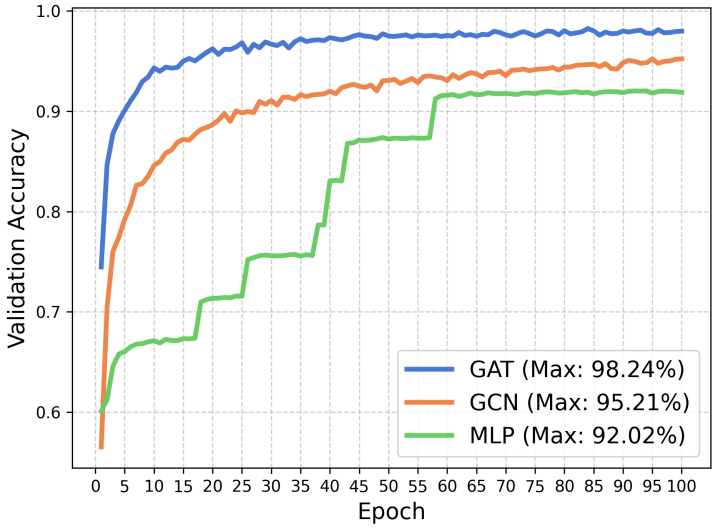

Figure 5.3: Validation Accuracy Comparison During Training

Comparing the best models in a longer training session, with 100 epochs, the 3-layer Large GAT increased the performance by a significant margin compared to Multi-Layer Perceptron from the previous study, while the 3-layer Large GCN performed only slightly better, with the best test set accuracies of 98.05% for the GAT, 95.05% for the Large GCN, and 91.9% for the Multi-Layer Perceptron. Figure 5.3 shows a more detailed representation of the training process.

I have also compared the computational efficiency of the embedding layer configurations. The result is a heatmap (5.4), showing the mean runtime of each layer over 700 iterations with a batch size of 256. The experiments were conducted on an Intel(R) Xeon(R) CPU @ 2.00GHz CPU with 32 GBs of DDR4 RAM, with a Tesla P100-PCIE-16GB GPU for the relevant tests. The analysis highlights an engineering trade-off between performance and runtime when comparing GCN and GAT layer types. Notably, increasing the hidden layer dimensions incurs lower costs than adding additional layers, regardless of the hardware used. The impact of transitioning from a small to medium or large hidden dimension size is negligible on the GPU; this is because it primarily increases the size of the matrices being multiplied, which GPUs excel at parallelizing. When considering real-time performance, only the CPU implementation of the three-layer Large GAT scores below 60 fps, indicating that computational efficiency is not a significant concern.

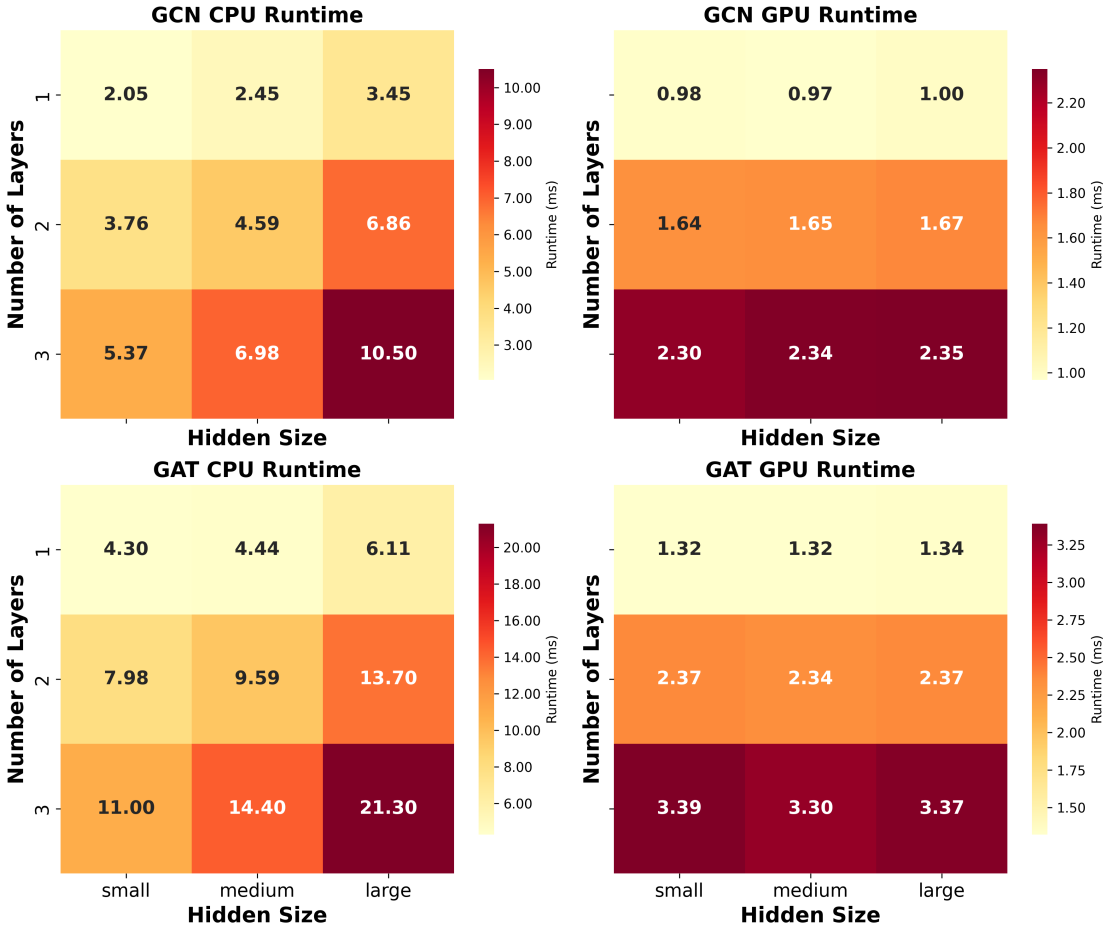

Figure 5.4: Runtime Comparison of GNN Architectures

5.4 Testing Hypotheses

Beyond identifying the best-performing model, several hypotheses were examined to understand the capabilities and limitations of GNN models better for pose classification:

1. **Effect of Preprocessing on Model Type:** tested by assessing the effect of preprocessing on top-performing models. Results partially confirmed the hypothesis: while the GAT model showed little to no change in performance without preprocessing, the GCN model experienced a notable decrease in test accuracy, with drops nearing 30% in some runs. For the MLP model, the validation accuracy displayed an intriguing pattern: it would plateau, then jump suddenly at seemingly random epochs, independent of learning rate or preprocessing (Figure 5.3). Removing preprocessing reduced the likelihood of these jumps, leading to poorer overall performance. This suggests that the GAT architecture is the most reliable for effectively learning to separate gestures in latent space, without the need for preprocessing.

2. **Inverse Graph Configuration:** An "inverse graph" structure (Figure 5.5) was also tested, where nodes representing joints not anatomically connected in the hand were linked. Given that anatomically connected joints tend to move in tandem, thus carrying related information in the features, connecting "independent" joints could theoretically increase information flow and improve model performance. This was categorically debunked, as both GNN architectures performed better with the anatomical connections, but GCN consistently felt a bigger drop in end-of-training test accuracy scores.

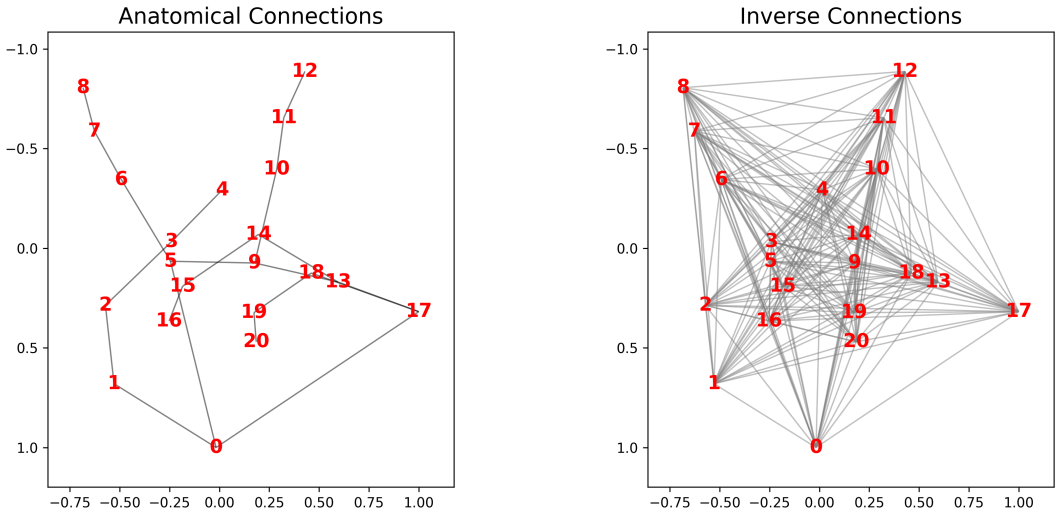

Figure 5.5: Two Methods for Connecting Hand Joints in Graph Representation

3. **Using z Axis Information from Mediapipe Hands:** Although depth information is theoretically beneficial, particularly for recognizing complex finger interactions, as noted in the Kaggle ASL FS Competition data description [5], "the MediaPipe model is not fully trained to predict depth so you may wish to ignore the z values". Doing so results in a roughly 1% decrease in accuracy, meaning that at least for this study the component is irrelevant.

Chapter 6

Sequential Fingerspelling

Earlier experiments (Chapter 5), on the Static Fingerspelling subtask showed that the new GNN-based embedding layer improved the separation of gestures in latent space representation. However, it remains to be seen whether this embedding technique can enhance the performance of the transformer architecture [35]. Addressing this question is critical, as sign language translation primarily involves sequential tasks.

6.1 Dataset

The dataset, created through a Kaggle competition sponsored by Google [5], aims to translate ASL fingerspelling into text. It includes over three million fingerspelled characters recorded by more than 100 Deaf ASL users from across the United States. Each participant, using a smartphone with a data collection app, was shown English phrases to fingerspell. They controlled video recording by pressing an on-screen button, and though video boundaries were adjusted, some inaccuracies remain. The dataset captures diverse signers across various lighting conditions, backgrounds, and hand preferences, with some even switching hands between clips. Additionally, co-articulation, a natural blending of letter handshapes, and unique approaches to capitalization are reflected, though target phrases are provided in lowercase.

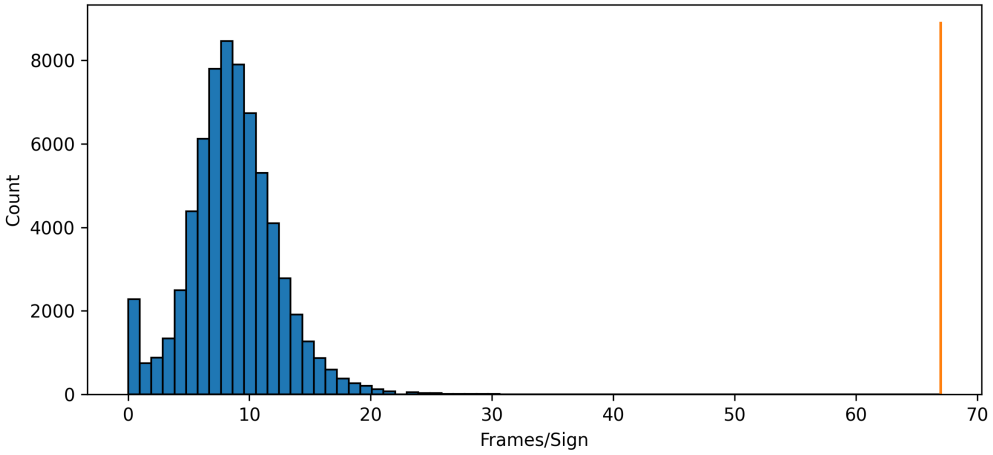

Figure 6.1: Distribution of Captured Frames/Sign (Max: 67)

Preprocessing steps:

1. 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 also lack consensus on their representation.
2. Select the dominant hand, which is the one that appears on more frames.
3. Filter frames that cut off some joints of the previously selected hand.
4. Remove the extensive number of outlier sequences as revealed by the long-tailed distribution (Figure 6.1). Sequences are removed that don't have at least 3 frames/sign or more than 20. (Note that the number of signs is precisely known from the phrase that the signers tried to recreate with gestures, as there is a one-to-one mapping between members of the alphabet and gestures. It is also true in the special case of repeating letters, where signers have to use distinct manual markers [31].)
5. Limit dataset to sequences shorter than 128.
6. The same normalization steps were applied as for the Static Fingerspelling Dataset described in Section 5.1.

6.2 Training and Evaluation

Using the premises of Chapter 5 (Static Fingerspelling), the graph representation of hands will be constructed with anatomically connected joints, where node features include only x and y dimensions returned by the MediaPipe Hands pose approximation model. The embedding layers to compare include the two best-performing GNN models from Chapter 5 and the Convolution-based solution from my previous work discussed in Section 4.2.

Table 6.1: Global Settings for Transformer Model Training

Transformer Parameters	**Value**
hidden dim	128
features/node	2 (x, y)
attention heads	2
# encoder layers	2
# decoder layers	4

The models are trained for 100 epochs each with the batch size limited to 32 for reasons discussed in Section 4.3.2. The model parameters were set according to Table 6.1. For evaluation, two key metrics were used:

1. **Accuracy** was calculated using teacher forcing.
2. **Masked Levenstein Distance**, which cuts off the sequences at the first "End of Sequence" token (EOS) and only compares the "valid" parts of both generated and expected output sequences.

At first, no learning rate scheduler was applied, only the Adam optimizer with the default learning rate of 0.001. The model equipped with the convolutional embedding layer, however, didn't converge using these settings. After applying a custom learning rate scheduler suggested by the original authors of the transformer architecture [35], the model finally started learning.

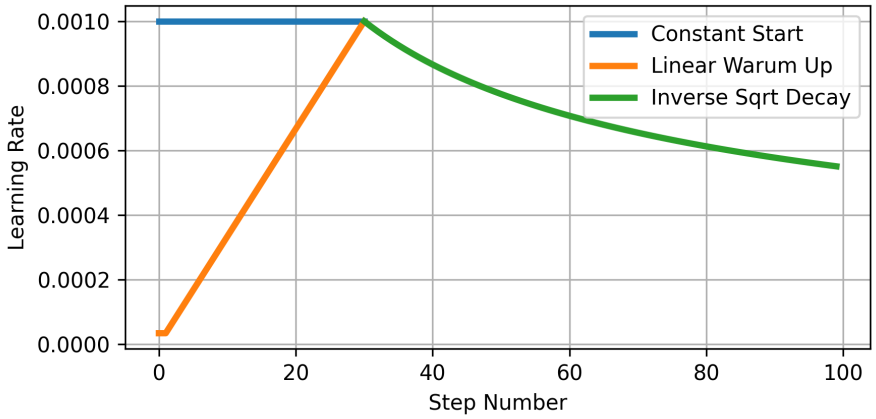

Figure 6.2: Applied Learning Rate Scheduling: the model with convolutional embedding benefited more from a **linear warmup** strategy, but the GNN architectures performed best, with a **constant learning rate** at the start. **Inverse Squareroot Decay** helped all models learn at later epochs.

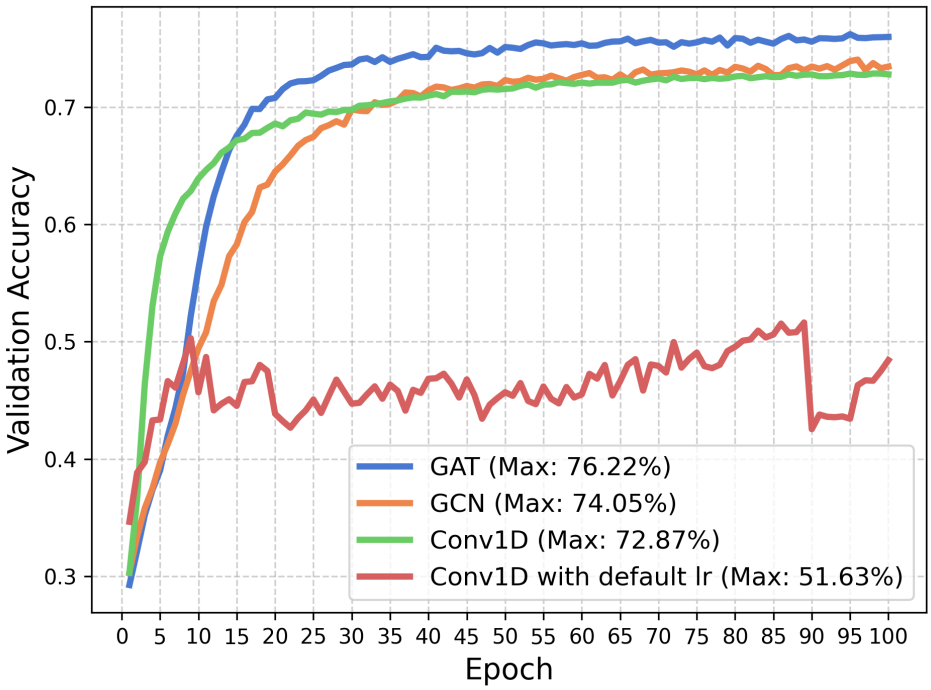

Figure 6.3: Validation Accuracies During Transformer Training

Results are similar to Static Fingerspelling, with GAT outperforming both other solutions, with a 76.19% test set accuracy compared to the 73.85% by the GCN equipped transformer and 72.44% by the model from the previous study. The application of GGNs once again proved to be more stable when it comes to different hyperparameters.

The only thing left to compare is the inference speeds. This is particularly important in the case of sequential translation as the model cumulatively generates the output, and a single translation takes several model runs to complete. Inference time was averaged over 100 iterations for each transformer equipped with different embedding algorithms.

Table 6.2: Performance Comparison of Different Transformer Approaches.

Embedding	**Accuracy**	**Levenshtein Distance**	**CPU (ms)**	**GPU (ms)**
GAT	76.19%	4.47	1110	33
GCN	73.85%	4.71	762	24
CONV	72.44%	4.6	20	12

From the data seen in Table 6.2, running the GNN-based models on the CPU is simply not feasible in its current state, as the GNN embeddings require the batch size during model creation, and it can't be modified during inference, causing a substantial difference in run times. Potential upgrades include fixing this problem, but another possible approach is to optimize the Graph Convolution algorithms to support proper batching, using the fact that graphs in this problem setting always have the same structure. Nevertheless utilizing a GPU makes all model configurations feasible for real-time translation.

Chapter 7

Conclusion

In summary, my research successfully achieved its primary objectives, validating the hypotheses through both empirical measurements and the development of a concrete solution. This work demonstrated the feasibility and effectiveness of using Graph Neural Networks (GNNs) for pose data in American Sign Language (ASL) fingerspelling translation, exceeding the performance of prior models. The GNN-based models surpassed the previously used Multilayer Perceptron (MLP) for static fingerspelling and convolutional embeddings for sequential data, demonstrating enhanced accuracy and adaptability. Specifically, GNNs addressed the limitations of convolutional approaches, such as sequence padding issues and the preservation of semantic structure, providing a robust alternative.

Through extensive experimentation with different model configurations, including variations in depth and hidden layer sizes, I found that the Graph Attention Network (GAT) architecture yielded the best results. This model leveraged a custom weighting mechanism for neighboring nodes during message passing, achieving a peak test accuracy of 98.05% for static fingerspelling, substantially higher than the 91.9% accuracy of the MLP model from my previous study. In the context of sequential fingerspelling, the new GAT-based embedding achieved a test accuracy of 76.19%, outperforming the previous best model, which reached only 72.44%. Sequential graph batching techniques, enabled by PyTorch Geometric, were also pivotal, allowing the GNN model to efficiently scale to larger datasets and preserve spatial information in real time.

In conclusion, this research not only established that pose information can effectively be represented as a graph, with nodes representing key points and edges denoting anatomical connections, but also that GNNs offer a powerful framework for advancing ASL translation tasks. The insights and results of this study provide a foundation for future exploration in graph-based approaches to sign language recognition and other gesture-related applications.

Future Work

1. Improving pose approximation algorithms is the single most critical factor in advancing sign language translation with Graph Neural Networks. Enhanced models that provide more accurate 3D representations and reliably handle interactions between hands and other body parts are essential. A promising study has shown progress in this area [19].

2. In the current version of GNN embeddings, graph convolutional layers must be applied to a large number of graphs during each inference. Optimizing the underlying neural network operations, given that these graphs share the same structure, could significantly improve computation time.

3. To further demonstrate the superiority of GNN-based embedding layers, it would be beneficial to explore how they can handle missing points in pose approximation data. By leveraging graphs to represent damaged input, we can avoid the use of padding tokens, allowing for a more efficient and accurate representation of incomplete data. This approach not only enhances the model's robustness but also highlights the flexibility of GNNs in managing real-world scenarios where data may be sparse or incomplete.

4. Since the connections in the pose graph impact model performance, it may be fruitful to experiment with connection strategies beyond the anatomical and inverse anatomical setups. GNN-based link prediction could be applied to identify which joints move together across different sign languages, revealing connections that are influential for translating gestures.

5. There are promising results for the reverse task of this study, where animations of sign language gestures are generated from a piece of text [6][4]. By creating diverse gesture animations from textual input, we can enhance the training datasets available for models, ultimately improving their performance and robustness in understanding and interpreting sign language.

Acknowledgements

Supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

I would also 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.

Additionally, my heartfelt appreciation goes to my girlfriend for her unwavering patience and understanding during the times I sacrificed our moments together for the sake of this 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] Rotem Shalev Arkushin, Amit Moryossef, and Ohad Fried. Ham2pose: Animating sign language notation into pose sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21046–21056, 6 2023.

[5] 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.

[6] Vasileios Baltatzis, Rolandos Alexandros Potamias, Evangelos Ververas, Guanxiong Sun, Jiankang Deng, and Stefanos Zafeiriou. Neural sign actors: A diffusion model for 3d sign language production from text, 2024. URL https://arxiv.org/abs/2312.02702.

[7] 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.

[8] 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.

[9] 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.

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

[11] Marcell Dancsó and Péter Ekler. Sign language translation to assist the hearing-impaired using pose approximation methods, artificial intelligence algorithms, and result correction with large language models. In *Proceedings of the Automation and Applied Computer Science Workshop 2024 (AACS'24)*, June 2024. ISBN 9789634219606.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Stanford Center for Health Education. Anatomy of the upper limb: Osteology of hand and wrist joint. YouTube, 2019. URL https://www.youtube.com/watch?v=gVEtcwrviVY&ab_channel=StanfordCenterforHealthEducation. Accessed: 2024-11-02.

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

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

[20] 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.

[21] Kaggle user: tecperson. Sign language mnist. https://www.kaggle.com/datasets/datamunge/sign-language-mnist. Accessed: 2024-10-28.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] Nicolas Pugeault. Asl finger spelling dataset. https://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset. Accessed: 2024-10-28.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] World Health Organization. Deafness and hearing loss. https://www.who.int/health-topics/hearing-loss#tab=tab_2, 2023. Accessed: 2024-11-03.

[38] 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.

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. abs/1901.00596, 2019. URL http://arxiv.org/abs/1901.00596.

[40] 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.

[41] 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.

[42] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. abs/1603.08511, 2016. URL http://arxiv.org/abs/1603.08511.

[43] 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.