PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Continuous Sign Language Recognition Using Deep Learning Models

RAYHAN RAHMAN
SPRING 2023

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Engineering
with honors in Computer Engineering

Reviewed and approved* by the following:

Mahanth Gowda
Assistant Professor in Electrical Engineering and Computer Engineering
Thesis Supervisor

John Sampson
Associate Professor of Computer Science and Engineering
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

Continuous sign language recognition is inherently a difficult task. Unlike translation of traditional auditory language, sign language is a visual-spatial language which requires processing of significantly more information. Developers have been attempting to solve this problem for decades to help the nearly 430 million people around the world who suffer from hearing impairment [1]. The most common technical method of tackling this problem is to use a machine learning model that has been trained on a robust dataset. A number of prominent models and algorithms exist that can be applied on videos of sign language to learn and predict the transcription of words and sentences in sign language with varying rates of success, measured through word error rate (WER). This paper explores two different deep learning models of continuous sign language recognition – an encoder model paired with a connectionist temporal classification loss function and an encoder-decoder model with a cross-entropy loss function – to investigate which model produces a lower word error rate and why.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis supervisor Mahanth Gowda and his Ph.D. student Songhe Wang for their guidance and support during this undertaking. Dr. Gowda encouraged me to fearlessly explore a challenging research topic in the pursuit of learning and gaining experience at a higher degree of academic reading and writing. Similarly, I am so grateful for Songhe's help over the past year in helping me understand how deep learning works and how it can be applied to sign language recognition. I truly appreciate his willingness to allow me to investigate a small portion of his graduate level research and his dedication to helping me and developing the experimental models in PyTorch. I would also like to thank my honors advisor John Sampson who has kept me on track throughout my college career, even when I encountered unexpected difficulties along the way. His academic and professional advice has been invaluable.

Finally, I would like to sincerely thank my family, my roommates, my mentors, and my friends. Without their endless support during my time at Penn State, I could not have completed this journey, nor would I be the person I am today.

Chapter 1

Introduction

The field of natural language processing (NLP) has experienced steady growth since its inception in the early to mid-20th century [2]. Beginning with simple machine translation, advancements included concepts like tokenization – breaking textual data into important elements – and soon after, machine learning utilizing decision trees and probabilistic algorithms. In the early 21st century, a shift towards deep learning became the preferred method of NLP because deep learning is inherently more efficient at solving ambiguous problems that cannot be tackled by simply using rules and fixed criteria [2]. Instead of a programmer providing rules for decision trees, deep learning is able to understand the process of mapping an input to an output through many layers of interconnected nodes. These layers combine to create algorithms called neural networks that work together to analyze and interpret data. Although deep learning is a relatively new field, due to the rapid expansion in both computing power and sophisticated algorithm design, the applications of this technology are seemingly endless. Utilizing deep learning methods, advances have been made in NLP with regards to neural machine translation as well as computer vision methods with regards to image and video captioning [3].

As these advances have greatly improved speech to text and language translation, in recent years, researchers have been applying the aforementioned techniques and learning methods to sign language (SL) recognition as well. An important distinction to note is that sign language translation involves taking written or spoken language and converting it to SL while sign language recognition involves interpreting and understanding sign language performed by a

person, and converting it into written or spoken language [4]. The latter describes the topic of this paper. However, sign language recognition is a more difficult task for computers to perform for the following reasons:

1. Visual-spatial complexity: Sign languages are 3-dimensional and involve hand movements, facial expressions, and body language, increasing the amount and complexity of input data to track.
2. Data limitations: There is much less data available for sign language than spoken languages which makes it harder to accurately train machine learning models.
3. Language variability: It is estimated that there are over 200 different sign languages used around the world, each with their own unique syntax and contexts [3].

Additionally, there are a multitude of factors to consider related to the input data used – type of sign language, method of feature extraction, input length – as well as factors related to the method of recognition and the specific model employed. Overall, many components of the system and the way they are combined impact the overall accuracy of the recognition model. These components will be further explored in Chapters 2 and 4.

For this paper, I will be investigating some of the various technical models that exist to successfully perform continuous sign language recognition. One of the most well researched papers on this subject was written by Koller et al. and published in 2015 [5]. This paper states in the abstract that it intends to be a starting point to newcomers in this field [5]. As someone who is very interested in this technology, especially as a result of having partially deaf family members, I wanted to design a basic experiment that expanded upon this work while serving as a deep dive into this field for an inexperienced researcher like myself.

In summary, the goal of this paper is to perform an investigation into current methods of continuous sign language recognition with a specific focus on comparing two different deep learning models. These two models will be implemented in PyTorch and evaluated based on the error rate produced when the model is applied to a large dataset of continuous input video signing under realistic conditions.

Chapter 2

Background Information

Sign Language Introduction

Sign language (SL) in its most basic form as a nonverbal method of communication has existed for centuries of human history. Today, sign languages are the native languages of the deaf and hard-of-hearing (DHH) communities worldwide. While rural and tribal languages have developed in different contexts, in the last 200 years, sign languages have emerged on national scales from deaf schools and have spread beyond national borders [6]. Some sign languages have even been used in international contexts such as American Sign Language (ASL) and International Sign. As previously stated, there are now estimated to be over 200 recognized sign languages, but some of the national languages are more widely known and used. Although these languages are relatively young, SL is highly structured and sign linguistics was established as an academic discipline from the 1960s onwards [6]. Thus, early research was focused on establishing how signs functioned like vocabularies and syntax of spoken languages.

Although many sign languages have now been formalized, the nature of communication and factors involved to express a signer's thoughts explain the difficulty humans, let alone computers, face when trying to understand sign language. Unlike simple hand gestures, sign languages have 5 overall parameters: handshape, location, palm orientation, body movement, and non-manual signals (NMS) [3]. Additionally, NMS cover a range of features such as facial grammar, body orientation, head turns and shakes, eyebrow movements, nose wrinkling, mouth movements and so on [3]. Due to this complexity and the fact that many of these parameters do

not have equivalents in written or spoken grammar, sign languages require encoding of both visual and spatial data for machine analysis.

Glossing

While translation is the process of conveying the meaning of a message from one language to another, glossing is unique to SL. Although sign languages are not written languages in the sense that we cannot "sound out" a sign by reading, glossing has been created as a written form of sign language that does not have the same structure as the spoken language equivalent [3]. Instead of spoken languages based on speaking and hearing, the visual-gestural method of SL requires glossing as an intermediate step to translation to preserve the meaning of sentences without compromising rules of language grammar. Glossing terminologies have been developed to include crucial information like location (LOC), palm orientation (PO), and number of repetitions (+). An example of glosses and meanings of spoken sentences can be found in Table 1. [3]. Additionally, the database used in this experiment from [5] contains gloss annotations of SL video as ground truth labels for training. Analysis of this corpus can be found in Chapter 3.

Table 1: Example of Glosses and Spoken Sentence Meanings [3]

No.	Gloss	Spoken Sentence	Selected Gloss Meaning
1	IX-loc:j OVER/ AFTER EXAGGERATE IX-3P:j LATER_2 USE COAT RAIN COAT USE UMBRELLA BOOT PANTS_3	People go too far: they use umbrellas, wear rain coats, and put boots and pants on	IX: index, IX-loc: points to a location.
2	IX-1p SAY part:indef UP-TO-NOW IX-3p:I 5“looking for words” IX-3p:I fs-PAUL KNOW IX-3p:I POSS-1p (1h) GOOD/ THANK-YOU FRIEND	I said to Paul in the email, “you know, you have been a good friend of mine”	fs: fingerspell, POSS-1: Possessive pronouns (my, mine, etc.,).
3	IX-1p LOOK:p 5“resignation” FINE IX-1p WAIT++	Thought “ugh” and ended up waiting again.	++: repeat sign.

DGS

German sign language, or Deutsche Gebärdensprache (DGS), is the indigenous sign language of Germany. While it is a minority language with an estimated 50,000 native signers, the data used for this paper relies on SL video from a German weather forecast as it one of the largest publicly available datasets of continuous sign language with gloss annotations [7, 5]. Standardization of the language in recent years eventually lead to legal recognition of the language in 2002 and since then, some television programs now include an interpreter signing in DGS [7]. While the language is similar to other European sign languages, additional information about the composition and uniqueness of the language can be found in Herbert’s work [7]. While the findings presented in this paper are only based on DGS, future work could possibly apply the techniques used to ASL as well.

Continuous Sign Language Recognition

As previously stated, continuous sign language recognition (CSLR) involves taking continuous video data of signing and outputting a machine transcription to the text or speech of a spoken language. One important distinction to note would be the difference between word-level and sentence-level sign language recognition. Word-level sign language recognition is a much simpler discrete task that requires a smaller dataset for training. In contrast, sentence-level sign language recognition uses temporal information and requires more complex architecture to consider the grammatical structure of the entire input sentence. Due to continuous data without time boundaries for each gloss, CSLR is a weakly supervised problem in which the system must learn the corresponding relation between the image time series and sequence of glosses [8]. Thus, this task requires a larger dataset and well-defined annotations of input data – a requirement fulfilled by the RWTH-PHOENIX database. In the subsequent sections, I will explain the major steps and components involved in CSLR.

Feature Extraction

Feature extraction is an important first step in the CSLR system. A feature is a measurable property or characteristic of input data that can be utilized to differentiate between classes or categories. For this task, features of the input video are qualities of visual information contained in the video to distinguish different signs. These may include locations of body, hand, and facial expression/landmark points as well as handshape, center-of-gravity, and motion trajectory of pixels [4]. Features can be thought of a data structure that compresses meaning and reduces dimensionality of complex input. Through machine analysis of each frame, feature

extraction processes the raw input data into a multi-dimensional feature vector [3]. For example, OpenPose, a real-time human pose detection library for photos, is able to extract 25 body-joint key points, 21 key points for each hand, 70 facial landmark key points, along with (x, y) coordinates to extract an input vector of 274 points for each frame in a video [3].

Additionally, I briefly wanted to introduce a few different ways to achieve feature extraction, all of which are further detailed in [9]. The primary categories of processing methods are 2D and 3D feature extraction. For 2D, features are extracted from frames of video through computer vision and mathematical processes. These include:

1. Principal Component Analysis (PCA): PCA performs mathematical procedures on matrices to reduce dimensionality by extracting and compressing the most important information from the data table [9]. It is useful for high-dimensional datasets like videos and works by computing variables called principal components and using them to form linear combinations of original variables to capture as much variance as possible.
2. Histograms of Oriented Gradients (HOG): This is another computer vision technique for object detection and recognition that involves computing the gradient orientation of each pixel in an image and grouping the gradients into histograms [5]. The histograms form a descriptor for the specified image region, capturing information about edge and texture structure.
3. Convolutional Neural Networks (CNN): CNNs are a type of deep learning model that extract relevant features by using multiple layers of convolutional filters. This is accomplished by performing discrete convolutions on the image with filter values as trainable weights. Multiple filters are applied on each channel and form feature maps which are pooled together [10].

For feature extraction in this experiment, a deep convolutional neural network (DCNN) was utilized in both the experimental models. A more detailed explanation of the network is provided in Chapter 4.

Recognition

The next step in the CSLR process is taking the visual features as input and training a recognition model to learn the mapping between SL features and the corresponding text they represent. There are several available methods to create this model, but I focused on machine learning and deep learning methods. Machine learning methods use statistical models such as Hidden Markov Models (HMM) to recognize signs based on their features.

An HMM is a probabilistic model used as a classifier for time series data that takes in observed data and a set of hidden states. It is most applicable for modeling data with temporal variations where the sequence of events has dependencies on previous events. Thus, it is frequently used as a classifier for many recognition systems. Specifically in SL recognition, it models each gesture as a sequence of hidden states that generate the observed video signals [11]. The goal of an HMM is to decipher the highest probability sequence of hidden states that generated the observed output sequence. Although early attempts at CSLR were influenced from automatic speech recognition methods using HMMs, results were poor [12].

Following these early attempts, researchers began experimenting with the use of deep learning methods. While machine learning relies on algorithms and statistical models to learn from experience, deep learning with neural networks can combine multiple layers of artificial neurons to process larger amounts of unstructured data and efficiently recognize complex

patterns. Neural networks are especially well suited for handling tasks with fixed size input and output vectors such as taking an input image and outputting a vector of probabilities for each output class it has been trained on 3. For videos, this is done by applying the neural network on a fixed number of frames [3]. Furthermore, Subburaj et al. found that that deep neural networks gave the best results for CSLR compared to other approaches because they have the ability to self-learn and self-associate if provided a large enough dataset for training [13].

Recurrent Neural Networks (RNN) are a specific kind of neural networks that are designed to process sequential data where the order of data matters. RNNs use hidden states which allow the output to be a function of the current input and output from the previous timestep, so the RNN can make predictions based upon all prior inputs [3]. By maintaining an internal state of the RNN that is updated at each timestep, the system is "remembering" information from previous timesteps, and the internal state is functioning as memory which is very useful. In practice, RNNs have shown superior performance to HMMs on handling complex dynamic variations in sign recognition [8].

While RNNs are powerful architectures for NLP-related tasks due to their ability to continuously process information over the course of a certain number of timesteps, they do not operate well over longer sequences of input data. This is due to the vanishing gradient problem during training. When the network has many layers and the gradients computed during backpropagation become very small through the multiplication of smaller gradients together, the network may become unstable and difficult to train [12]. To account for this and remember longer-term dependencies of dozens to over 100 timesteps, a special type of RNN called a Long Short-Term Memory (LSTM) can be used.

LSTM is a type of RNN developed to handle the vanishing gradient problem and capture longer-term dependencies in sequential data. Using a series of gating mechanisms and a second hidden state called a memory cell, LSTM can be used as a replacement for RNN and has been shown to be more effective at modelling complex sequences such as sign language recognition [3]. When combined in certain configurations, LSTMs can become ever more powerful as a part of a sequence-to-sequence model. A sequence-to-sequence model is another deep learning architecture that is used to map input sequences to output sequences of variable length. In this instance, used to map sign language videos to corresponding gloss sentences. Furthermore, in the work presented by Ananthanarayana et al. [3] the researchers concluded that LSTM models are more capable of handling long-term dependencies when compared to Vanilla RNN.

These models are a type of neural network architecture where the whole input sequence is encoded and passed to a decoder to produce the expected word in each time step. Both the encoder and the decoder models are created using LSTMs. The encoder LSTM reads the entire source sequence and by the end of it, the hidden state of the encoder includes a context vector that is a summary of the input [12]. The decoder LSTM has inputs of the previous output and the hidden state, but unlike the encoder, has an additional input of the context vector to predict the next output [12]. Chapter 4 analyzes how sequence-to-sequence models function, but the model essentially aims to maximize the probability of a correct target word sequence.

Loss Function

One final component of the system is necessary to deal with the issue of alignment in our system as well as optimized training of the model. After the feature extractor and sequence-to-

sequence learning model, we are left with a predicted output sequence from the model, but there is no prior knowledge of where the signs occur in the image stream. To better train the model, a loss function can be used quantify how accurate the predicted class labels are compared to ground-truth labels [13]. To account for the fact that these sequences may have different lengths, the loss functions employed may use an alignment feature as well. Alignment-free algorithms are able to collapse the output sequence to better approximate the ground-truth labels, demonstrated visually in [14]. By minimizing loss and increasing accuracy of the predicted sequence, the loss function serves as a training optimization tool that improves the performance of the model.

Measurement

There are a few possible methods of measurement and values that can be used to evaluate the performance of the CSLR method quantitatively. These include BiLingual Evaluation Underscore (BLEU) scores, Character Error Rate (CER), and Word Error Rate (WER) [13]. While each metric has its advantages and disadvantages, WER is the most widely accepted and used indicator of performance in this field as it measures the least number of operations – substitution, deletion, and insertion – needed to transform the reference sequence into the hypothesis [8]. Equation (1) demonstrates how this value is derived.

$$WER = \frac{\#substitutions + \#deletions + \#insertions}{\#words\ in\ reference} \quad (1)$$

Chapter 3

Dataset

The dataset used in this paper is the RWTH-PHOENIX-Weather 2014 corpus cited by the Rhine-Westphalia Technical University of Aachen in [5]. This is a large publicly available dataset representing unconstrained real life sign language. All videos were aired by the German public TV station PHOENIX from 2009-2013 during the weather forecast portion of daily news broadcasts [5]. Unlike constrained vocabulary datasets, the video recordings feature phenomena found in typical conversations like false starts, hesitations, and dialect variations. Additionally, this database is comprised of 9 signers, 1081 sign vocabulary, and 7k sentences with over 10 hours of data and 960k frames [15]. Lighting conditions, backdrop color, signers' dark clothing, and the position of the sign language interpreter were controlled by the television studio to ensure consistency over different videos. All videos in the dataset have a resolution of 210 x 260 pixels and 25 frames per second [5]. Although this means the temporal and spatial resolution is low, this dataset remains to be one of the largest and most widely used in the field currently, especially due to its size of "real life" data. An example of the video frames can be found in Figure 1.

Figure 1: Example video frame from RWTH-PHOENIX [15]

The dataset is comprised of German Sign Language (DGS) and translated into German for the ground truth annotations on the training set. Of these ground-truth captions, there are 1,078 unique words [3]. The data features a gloss annotation scheme introduced in Chapter 2 in which the annotation describes the meaning of a sign rather than its appearance [5]. An important consideration to note would be the fact that the subject matter of the sentences is confined to weather-related sentences. As show in the world cloud for the DGS dataset, there are many weather-related words with a high degree of frequency. This may have induced a more effective training of the model as a confined subject matter reduced variability in the number of unique words presented.

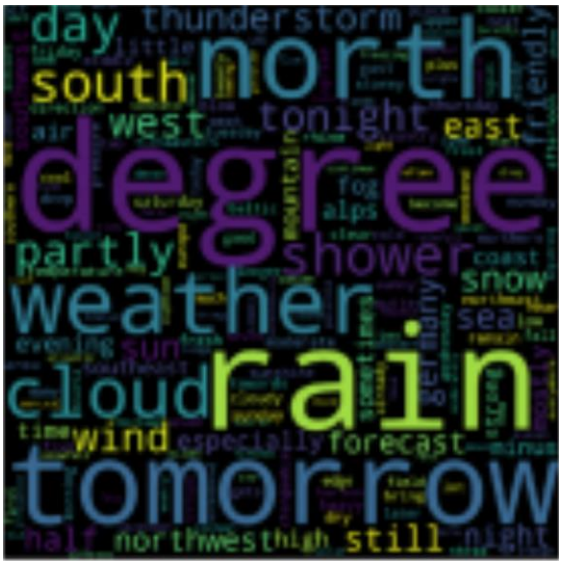

Figure 2: Word cloud for dataset [5]

The corpus contains both a single-signer dataset as well as a multi-signer (MS) dataset that is larger and contains input from 9 different signers. We will be using the multi-signer setup with additional statistics provided in Table 2.

Table 2: Corpora Statistics of RWTH-PHOENIX-Weather MS [15]

	Phoenix MS		
	Train	**Dev**	**Test**
# signers	9	9	9
duration [hours]	10.71	0.84	0.99
# frames	963,664	75,186	89,472
# sentences	5,672	540	629
# unique sentences	5,672	540	629
# running glosses	65,227	6,032	7,089
vocabulary size	1,081	467	500

As shown in Figure 3, while the signers' appearances and abilities may vary, the environment remains controlled.

Figure 3: Example images and percentage of data from individual signers [15]

In summary, the RWTH-PHOENIX 2014 multi-signer corpus was chosen as it is ultimately one of the largest publicly available datasets used for continuous video sign language recognition at this time. The statistics and images provided offer a glimpse into how robust this dataset is and how the conditions mimic real world sign language as opposed to a rudimentary vocabulary set.

Chapter 4

Model Architecture

Experiment Introduction

After considering many of the available options for CSLR, ultimately, I chose to compare 2 similar sequence-to-sequence models. I analyzed the WER performance of an Encoder model with a CTC loss function against an Encoder-Decoder model with a cross-entropy loss function. The motivation for comparing these specific models was to investigate the efficacy of the decoder structure and whether the CTC loss function was able to achieve similar results. The use of a decoder versus the use of a CTC loss function presents 2 different ways of aligning input with output so I was curious to see which method would prove to be more accurate. The experiment and models implemented were constructed in PyTorch and based off some widely used and available open-source models. In the following sections, I will provide a broad overview of how these architectures function.

Encoder Model

As mentioned in Chapter 2, an encoder architecture is an important component of a sequence-to-sequence method for achieving CSLR. In this experiment, the encoder model was constructed with 2 other deep learning models that perform data preprocessing and gesture recognition. The first model is a CNN used to achieve feature extraction. A CNN uses multilayer superposition to extract low-level features into relevant features. It is made up of many layers in general, but primarily convolutional and pooling layers. The convolutional layers perform

convolution operations to extract features from an input or prior layer while pooling layers constantly shrink the data's space size, reducing the number of features and computations [13]. The final fully connected layer, also known as a dense layer, is used to take the high-level learned features and classify the input image. A simple CNN diagram is represented in Figure 4.

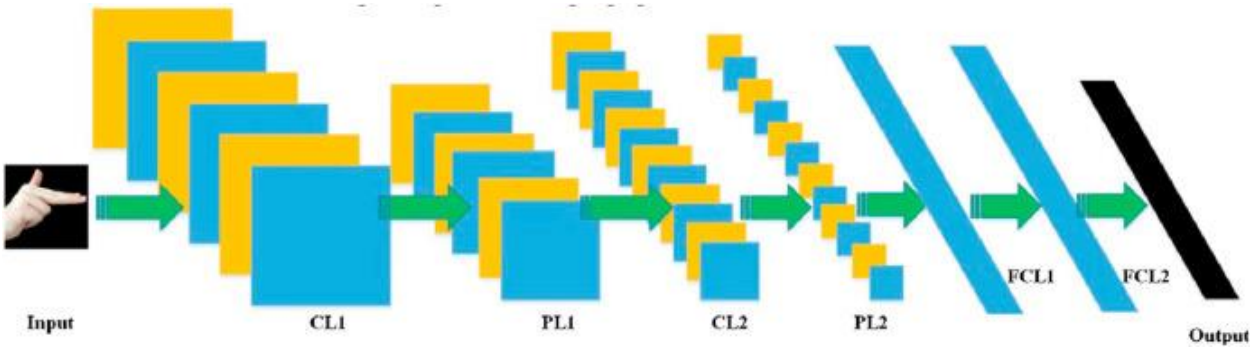

Figure 4: CNN diagram showing convolution, pooling, and fully connected layers [13]

The CNN for this experiment was an implementation of ResNet, which is one of the most popular CNN architectures used for image classification tasks, featured here in [16]. This powerful neural network was created by He et al. and utilizes a residual function to match the predicted value of a model with the actual value better. In doing so, even as deeper neural networks are constructed, this residual functionality works to maintain a high degree of accuracy to reduce the impact of the vanishing gradient problem. In fact, as shown in Figure 5, when the ResNet model was trained on a large image database, the deeper layered model had less training error instead of more, demonstrating that it is able to address the degradation problem. Overall, the ResNet model outperformed plain networks for image validation and has been modified for use as a feature extractor in many other works as well. The residual network used for the feature extractor is ResNet-18 with 18 layers to extract 512 features. Once the fully connected layers transform the visual features into a feature vector, this vector is inputted to the LSTM encoder.

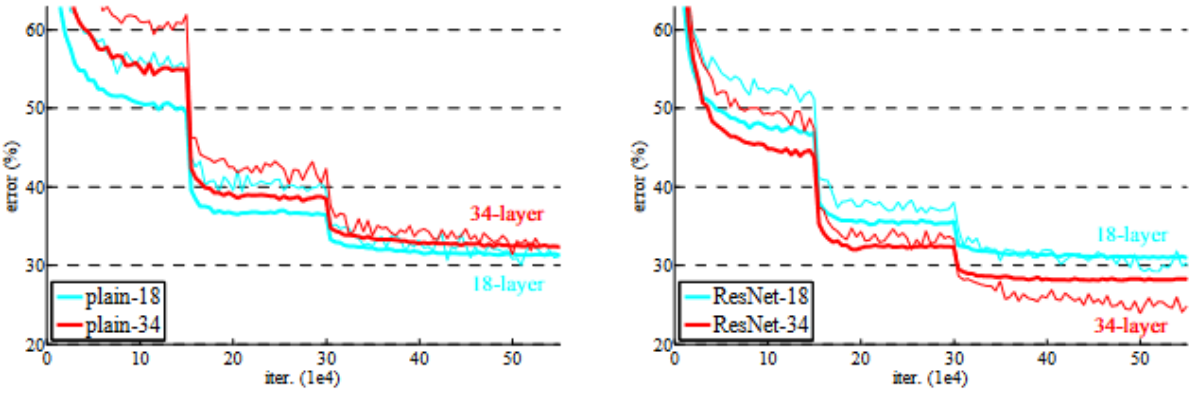

Figure 5: Error in Plain Networks Versus Residual Networks [16]

At this stage, the input to the LSTM is a sequence of feature vectors where each vector corresponds to a frame in the continuous SL video. As each sequence is processed one at a time, the model outputs a hidden state which is fed back into the LSTM as the input for the next time step. This hidden state carries semantically rich visual information of the current signing input as well as past feature vectors [3]. Since the LSTM is a type of RNN, it is able to process this sequential data between time steps and incorporates multiple memory gates to control the flow of information out of the memory cell. This gate functionality makes the LSTM more suitable to handle the long-term dependencies presented by signs that occur over 1-3 seconds on average [12]. After numerous layers of computation further detailed in [12], the final output of the LSTM encoder is a context vector represented as 3D matrix of batch size, sequence length, and dimensionality. Batch size is the number of sequences processed in parallel which depends on GPU but for our experiment was between 2-4 sequences. Sequence length refers to the number of frames per video. Dimensionality is the size of the hidden state at each time step.

The next step in this model is to convert this high-level representation into a probability distribution over output classes. This is achieved with a softmax classifier which is a

classification model often serving as the final layer in a recognition neural network model [17]. This fully connected layer uses a weight matrix and a bias vector the size of the vocabulary to compute the probability distribution, detailed in [8]. Essentially, the function takes the dot product of the encoder output and the weight matrix, adds the bias vector, and applies the softmax function element-wise to produce a probability distribution over the vocabulary. Further explanation of the softmax equation can be found in [18], but an example of the basic formula can be found in equation (2). The output of the LSTM is turned into categorical probabilities of gloss labels with *K* classes where *P* is the probability of label *i* at time *j*. The final step in the model tackles the issue of alignment and optimization.

$$P_{ij}^{c} = [\sigma_{cls}\left(\phi\left(h_j^c\right)\right)]_i = \frac{e^{[\phi\left(h_j^c\right)]_i}}{\sum_{k=1}^{K} e^{[\phi\left(h_j^c\right)]_k}} \quad (2)$$

CTC Loss

The connectionist temporal classification (CTC) loss is an objective function that integrates all possible alignments between the input and target sequence [8]. Since there is no prior knowledge of where the signs occur in the unsegmented image stream, CTC loss presents a solution by summing over the probability of all possible alignments of the label sequence that could give rise to the ground truth label sequence. First, CTC decodes the sign gloss sequence from the probability distribution generated by the previous layer by introducing a blank label as an assistant token to explicitly model the transition between 2 neighboring signs [17]. This is represented by equation (3) where the CTC alignment π becomes a sequence of blank and gloss

labels with length *N*. If *x* represents the image sequences at each timestep *n*, the probability distribution per time-step is given by:

$$\Pr(\pi|x) = \prod_{n=1}^{N} \Pr(\pi_n|x) = \prod_{n=1}^{N} \mathrm{P}^{c}_{\pi_{n,}n} \quad (3)$$

Next, to obtain the final decoded sequence, CTC defines a many-to-one function β that removes repeated labels and blanks from the predicted output. The probability of now observing the target sequence *y* is obtained by computing the sum of probabilities of all possible alignments, represented by $\beta^{-1}(y)$.

$$\Pr(\mathrm{y}|\mathrm{x}) = \sum_{\pi \epsilon \beta^{-1}(y)} \Pr\,(\pi|x) \quad (4)$$

To demonstrate this with a simple example shown in Figure 6, a CTC loss function is applied to an alignment of predicted characters to reduce repeated and blank labels represented as ε using the many-to-one property.

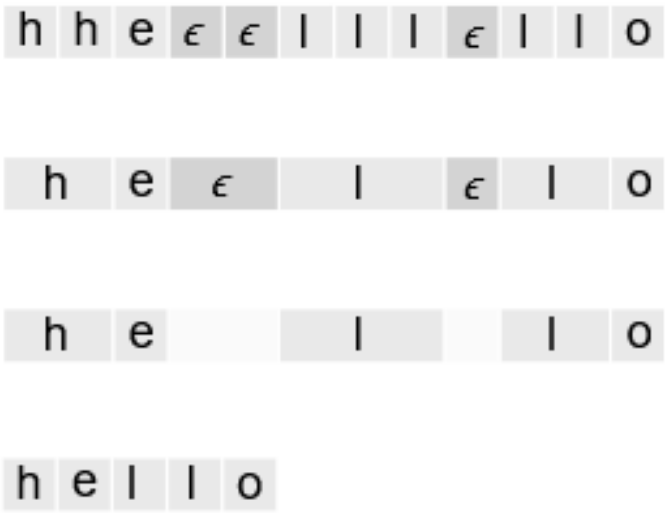

Figure 6: Visualization of CTC [14]

The final step towards calculating loss–a scalar value representing the distance between the predicted sequence and the true label sequence–is applying the loss function. This is calculated as the negative log probability of correctly labelling the image sequence *x* as the target sequence *y*, shown in equation (5).

$$\mathcal{L}_{CTC}(x, y) = -logPr(y|x) \quad (5)$$

The lower the loss, the better the model is at predicting the desired output. The loss is used to optimize the similarity between training samples and class wise prototypes. Furthermore, the model updates the internal feature and weight parameters based on loss value during training to reduce loss and optimize overall performance. In doing so, the predicted probability of correct labels gradually increases and the training of the network will carry on with convergent assignment [17]. In practice, this is accomplished with dynamic programming techniques for forward-backward propagation.

Encoder-Decoder Model

This model is frequently used in sequence-to-sequence methods of recognition and the outcome of an early RNN Encoder-Decoder model used by Fang et al. [19] indicated that the dual LSTM model could successfully capture important features of ASL. As the name suggests, the Encoder-Decoder model is similar to the previous Encoder model in its use of the initial CNN for feature extraction and an LSTM structure for encoding input frames. By the end of reading the input sequence, the final output of the encoder LSTM is still an information rich context vector.

However, the difference in this model is the feeding of this encoded feature representation to a decoder instead of a softmax classifier as demonstrated previously. Unlike the encoder, in addition to the previous output and the hidden state, the decoder has an additional input of the context vector in order to predict the next output. Equations (6) and (7) further describe the Encoder and Decoder hidden units in the model. Here, x_t is the input at time t, y_{t-1} is the previous output, h_{t-1} are the hidden outputs, and c is the context vector – encoder output at last input time step.

$$h_t^{Encoder} = RNN\left(x_t, h_{t-1}^{Encoder}\right) \quad (6)$$

$$h_t^{Decoder} = RNN\left(y_{t-1}, ch_{t-1}^{Decoder}\right) \quad (7)$$

Once this context vector is fed to the decoder LSTM, it is used as the initial hidden state and the decoder generates an output for the start symbol, which is fed back into the decoder as its input for the next step to predict one word at a time. The decoder continues to generate the variable output sequence step by step using the recurrence relation of considering both the current input and previous hidden state until the sequence is complete, indicated by an end token. Additionally, another distinct feature of the decoder is its ability to align the input and output sequence. While this was previously done by the CTC loss function in the encoder model, the decoder utilizes an alignment mechanism to directly model the alignment between each input frame and the corresponding output gloss. With this mechanism, an alignment distribution is computed using a compatibility function that measures the similarity between the decoder output at the current time step and each of the encoder output features. This entire model can be visualized in Figure 7. Symbol meanings are detailed in Table 3 and the figure shows the

recognition process from feature extraction, to encoder, to decoder, to the final fully connected layer.

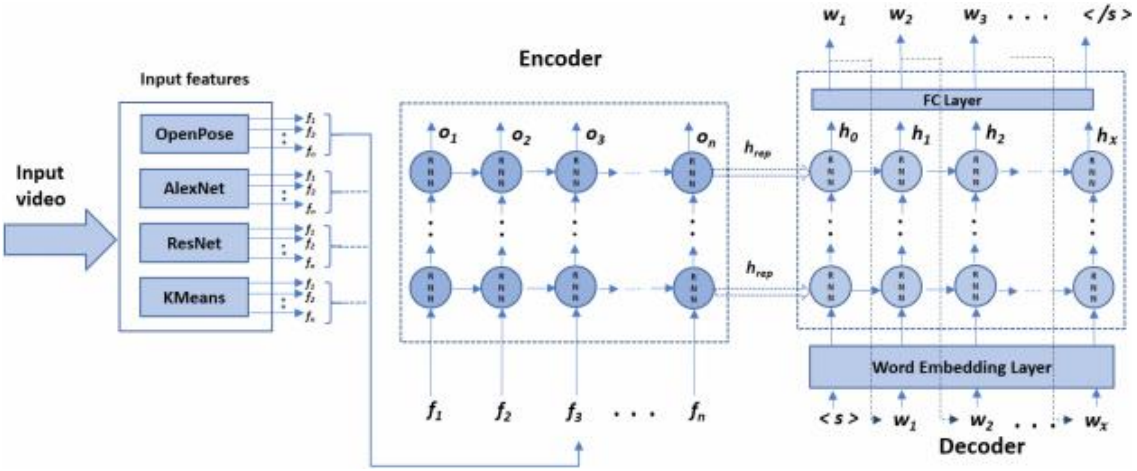

Figure 7: Visualization of Encoder-Decoder Model [3]

Table 3: Notations for Sequence-to-Sequence Models [3]

Symbol	Meaning
n	Input sequence length
x	Output sequence length
f_n	Feature vectors for each frame
o_n	Output vectors from encoder
h_{rep}	Embedding from encoder
w_x	Predicted words
<s>	Start sentence token
</s>	End sentence token

The fully connected layer of this model performs the same task of applying the softmax activation function to produce a probability distribution over possible output classes. The gloss with the highest probability is chosen to generate the predicted output.

Cross-Entropy Loss

Similar to CTC, a loss function is required in the model to quantify degree of accuracy of the predicted output compared to the ground truth sequence and to optimize training. Categorical cross-entropy is a loss function commonly used in multiclass classifications tasks like CSLR and is simpler to train than CTC, but requires alignment of input and output sequences [13]. Hence, it is best used for tasks where the input and output sequences have a one-to-one alignment which was achieved by the decoder. While the cross-entropy loss also measures the difference between the predicted probability distribution and the true probability distribution of the output sequence, the computation to produce loss is simpler than CTC. In order to minimize the loss, the model is penalized logarithmically, yielding a large score for large differences closer to 1 and small scores for differences closer to 0. A perfect model has a cross-entropy loss of 0. In equation (8) the basic formula for loss is given where t_i is the true class distribution and p_i is the predicted class distribution. The log function produces negative values on the range [0,1) so loss becomes the negative log-likelihood of the predicted probability for the true label, calculated for all classes and summed for total loss. As with all loss functions, the goal is to minimize loss which is accomplished by iteratively updating the model's parameters during training until convergence – when the loss function no longer decreases significantly [13].

$$\mathcal{L}_{CE} = -\sum_{i=1}^{n} t_i \log (p_i) \quad (8)$$

Model Comparison

Now that both of the models in this experiment have been briefly explained and analyzed at a high level, they can be compared to develop a hypothesis on which one may perform better. While a multitude of other possible methods for CSLR exist, deep learning-based approaches like CNN, RNN, and LSTM have been proven to provide better recognition accuracy when evaluated against them [13]. This increases confidence that both models will produce a low WER, but the differences of each one must be considered.

The encoder with CTC loss model is comparatively a far simpler model than the encoder-decoder. The reduced size and complexity of the architecture stems from the lack of the second LSTM decoder unit. This absence results in less training parameters which accelerates training and results in reaching convergence earlier. Additionally, a model that is easier to train can save time and resources, factors that have thus far been overlooked. Finally, the ability of this model to handle input sequences where alignment is not clearly defined through the use of CTC makes this model appear especially promising for this task. By eliminating the need for explicit alignment between the input and output sequences, the training process is further optimized.

However, one disadvantage presented by the CTC loss function is the possibility of the spike phenomenon. The spike phenomenon is an overserved behavior that causes only a few key frames to contribute to the final result and makes the visual module lose its discriminative power for the other frames [17]. In its attempt to predict the probability distribution of all possible output sequences, CTC can become overconfident in predicting a particular symbol. This results in a sharp increase, or spike, in the distribution for that symbol which negatively impacts the model's accuracy. While a number of factors can influence the occurrence of the spike phenomenon, it is a known issue and widely exists during training with CTC loss [17]. Some

proposed techniques for addressing this issue that were not used in this experiment include adding additional gloss segmentation to enhance the visual module, reducing the degree of non-linearity, or utilizing a less aggressive optimization algorithm [17].

The aforementioned advantages of the encoder model alluded to the increased complexity of the encoder-decoder model with cross-entropy loss. However, favorable features of this second model include the ability to learn alignments better and a possibly more accurate loss function. First, the encoder-decoder explicitly models the conditional probability distribution over the target sequence as apposed to the CTC loss model which marginalizes over all possible alignments. The decoder and cross-entropy loss function work together to compare the predicted probability with the true distribution at each timestep during training to more effectively learn the alignment between input and output sequences. This explicit alignment results in better performance for processing longer input sequences. In contrast, while the CTC loss function is able to handle variable length input sequences through the use of blank and repeated tokens, the implicit alignment assumption can result in errors. Finally, the encoder-decoder model consists of more layers and increased depth in models often results in better performance since it has a larger capacity to capture complex data patterns than shallow models [3].

Nevertheless, this heightened complexity leaves more room for error in the practical design of the model and can lead to less efficient or challenging training. While it was previously established that LSTMs mitigate the risk of vanishing or exploding gradients, deeper networks still have a higher chance of experiencing them [3]. This leads to degraded performance, difficulties experienced during training, and slower convergence.

Clearly both of these models offer a unique combination of advantages and disadvantages. However, the encoder-decoder model is significantly more researched and widely

accepted as the preferred method of performing CSLR. Nonetheless, no study was found to have definitively proven whether the encoder-decoder model is superior to the encoder with CTC model. Similarly, there is limited research directly evaluating the use of a decoder model versus a CTC loss function with regards which alignment method results in greater accuracy. Nonetheless, due to its ability to process longer input sequences, mange more complex tasks, and offer more precise alignment, the encoder-decoder model is expected to produce a lower WER.

Evaluation Metric

As demonstrated in Chapter 2, WER is the metric used to measure accuracy and performance of the models in this paper. The models were developed to produce this final percentage of words incorrectly recognized by the system based on the final output sequence and the ground truth label sequence. Alternative metrics were considered, but were not successfully implemented.

Chapter 5

Results

Ultimately, both model implementations were able to successfully achieve CSLR to similar degrees of accuracy. As demonstrated in Table 4, the encoder model coupled with the CTC loss function produced a WER of 23.2% while the encoder-decoder model paired with the cross-entropy loss function produced a WER of 29.7%. Contrary to our speculative hypothesis, the encoder model proved to be slightly more successful based off of these values. While this result was unexpected, there are a variety of factors to further investigate that could explain this outcome.

Table 4: WER of Experimental Models

Model Name	**WER**
Encoder with CTC Loss	23.2%
Encoder-Decoder with Cross-Entropy Loss	29.7%

First, the additional depth and complexity of the encoder-decoder may have led to vanishing or exploding gradients which adversely affected the training of the model. Next, both of these models are extremely modular in the sense that each system consists of many smaller components, models, and layers of computation. For each of these stages of the CSLR process, there are a plethora of alternative elements that could have been implemented instead. Therefore, it is possible that some of the specific design choices, unique to the encoder-decoder with cross-entropy loss model, negatively impacted the WER. A final source of error could stem from implementation error in the development of these models in PyTorch in which the decoder, cross-entropy loss function, or other fully connected layers may have been created inefficiently.

While there are other factors that could have contributed to this result, these three were identified as the most likely sources of error.

Although there is no true or expected value available to use in calculating percent error, the success of this experiment can be recognized through comparisons with top-performing CSLR models which were applied to the same RWTH-PHOENIX-Weather database. The work of Koller et al. [5] that was used as a starting point, was only able to produce a WER of 53%. At the time of publication in 2015, this paper was the first presentation of system design on a large data set with real-life applicability [5]. However, the field has advanced immensely in the years since and approximately 50 more papers utilizing this dataset have been published. Koller et al. published a new paper in 2019 [20] investigating multi-stream CNN-LSTM-HMM models and the use of a dual-stream generated a significantly higher WER of 24.1%. Finally, Chen et al. [21] have created the current best-known CSLR model that achieved a WER of 18.8% which is higher than either of the 2 models evaluated in this experiment. Therefore, both models evaluated in this experiment performed exceptionally well by comparison.

Table 5: WER of Alternative Models [21, 17, 20, 5]

Model Name	**WER**
Two Stream-SLR	18.8%
Self-Mutual Knowledge Distillation	20.5%
Dual Stream CNN -LSTM-HMM	24.1%
Statistical Model with CMLLR	53%

Chapter 6
Conclusion

This paper was ultimately successful in satisfying its three-pronged purpose: investigate the field of continuous sign language recognition as a newcomer, analyze the structure and functionality of 2 distinct but widely accepted deep learning models for CSLR, and conduct an experiment to evaluate the word error rate of each model. Both models were able to perform the task to a relatively high degree of accuracy when compared with the current leading methods for CSLR. However, the end result of the experiment contradicted the hypothesis that an encoder-decoder model with a cross-entropy loss function would perform better than an encoder model with a connectionist temporal classification loss function. The most significant factors that may have caused this result were identified as excessive architecture complexity, overall suboptimal design choices, or implementation error. While the findings of this experiment are insufficient to determine the superiority of one model over the other, continued investigation of this topic is still warranted.

Further analysis would attempt to determine a more precise justification of the result and future work would include more rigorous testing with increased data analysis. Regardless, there is considerable potential for further research into the design and testing of these 2 models as well as other approaches or model combinations. As this field continues to advance rapidly and new techniques are discovered, the question of what is the most optimal method of sign language recognition remains unsolved. Yet researchers continue this important work in the hopes of a future where the communication divide between non-disabled people and hearing-impaired signers is brought closer together.

BIBLIOGRAPHY

[1] N. Amangeldy, S. Kudubayeva, A. Kassymova, A. Karipzhanova, B. Razakhova, and S. Kuralov, “Sign Language Recognition Method Based on Palm Definition Model and Multiple Classification”, Sensors, vol. 22, no. 17, 2022. https://doi.org/10.3390%2Fs22176621

[2] P. Johri, S. Khatri, S. Kumar, A. Al-Taani, M. Sabharwal, S. Shakhzod, and A. Chauhan. “Natural Language Processing: History, Evolution, Application, and Future Work,” Proceedings of 3rd International Conference on Computing Informatics and Networks, pp. 365-375, January, 2021. https://doi.org/10.1007/978-981-15-9712-1_31

[3] T. Ananthanarayana, P. Srivastava, A. Chintha et al., “Deep learning methods for sign language translation,” ACM Transactions on Accessible Computing (TACCESS), vol. 14, no. 4, pp. 1–30, 2021. https://dl.acm.org/doi/10.1145/3477498#d1e1172

[4] S. Ong and S. Ranganath, “Automatic Sign Language Analysis: A Survey and the Future beyond Lexical Meaning,” IEEE transactions on pattern analysis and machine intelligence, pp. 873-91, 2005. http://doi.org/10.1109/TPAMI.2005.112

[5] O. Koller, J. Forster, and H. Ney, “Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers,” Computer Vision and Image Understanding, vol. 141, pp. 108–125, 2015. https://doi.org/10.1016/j.cviu.2015.09.013

[6] A. Kusters and C. Lucas, “Emergence and evolutions: Introducing sign language sociolinguistics,” Journal of Sociolinguistics, February 2, 2022, vol. 26, pp. 84-98. https://doi.org/10.1111/josl.12522

[7] M. G. Herbert, “A New Classifier-Based Morpheme in German Sign Language (DGS),” in University of Pennsylvania Working Papers in Linguistics: Proceedings of the 39th Annual Penn Linguistics Conference, Philadelphia, PA, USA, Vol. 22, Iss. 1, Article 15, 2016. pp. 128-136. https://repository.upenn.edu/pwpl/vol22/iss1/15

[8] R. Cui, H. Liu, and C. Zhang, “Recurrent Convolutional Neural Networks for Continuous Sign Language Recognition by Staged Optimization,” Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China, 2017, pp. 7361-69. https://doi.org/10.1109/CVPR.2017.175.

[9] S. Suhajito, F. Wirayana, , I.G.P.K Negara, and A. Zahra, “Feature Extraction Methods in Sign Language Recognition System: A Literature Review,” in 2018 Indonesian Association for Pattern Recognition International Conference, Jakarta, Indonesia, 2018. pp. 11-15. https://doi.org/10.1109/INAPR.2018.8626857

[10] L. Pigou, S. Dieleman, PJ. Kindermans, and B. Schrauwen, “Sign Language Recognition Using Convolutional Neural Networks,” ECCV 2014 Workshops, Part I, vol. 8925, 2015, pp. 572–57. https://doi.org/10.1007/978-3-319-16178-5_40

[11] A. A. Ahmed and S. Aly, "Appearance-based Arabic Sign Language recognition using Hidden Markov Models," 2014 International Conference on Engineering and Technology (ICET), Cairo, Egypt, 2014, pp. 1-6. https://doi.org/10.1109/ICEngTechnol.2014.7016804

[12] Arvanitis, Nikolaos & Constantinopoulos, Constantinos & Kosmopoulos, Dimitris, “Translation of Sign Language Glosses to Text Using Sequence-to-Sequence Attention Models,” in 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), Sorrento, Italy, pp. 296-302, 2019. https://doi.org/10.1109/SITIS.2019.00056

[13] S. Subburaj and S. Murugavalli, “Survey on sign language recognition in context of vision-based and deep learning,” Measurement: Sensors, vol. 23, 2022. https://doi.org/10.1016/j.measen.2022.100385

[14] Hannun, "Sequence Modeling with CTC", Distill, 2017. https://distill.pub/2017/ctc/

[15] J. Forster, C. Schmidt, O. Koller, M. Bellgardt, and H. Ney, “Extensions of the Sign Language Recognition and Translation Corpus RWTH-PHOENIX-Weather”, in Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14), 2014, pp. 1911–1916. www.lrec-conf.org/proceedings/lrec2014/pdf/585_Paper.pdf

[16] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Microsoft Research, 2015. https://doi.org/10.48550/arXiv.1512.03385

[17] A. Hao, Y. Min and X. Chen, "Self-Mutual Distillation Learning for Continuous Sign Language Recognition," 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 2021, pp. 11283-11292. https://doi.org/10.1109/ICCV48922.2021.01111

[18] W. Chen, D. Grangier, and M. Auli, ‘Strategies for Training Large Vocabulary Neural Language Models’, in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2016, pp. 1975–1985. https://doi.org/10.48550/arXiv.1512.04906

[19] B. Fang, J. Co, and M. Zhang, “DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation,” ACM Conference on Embedded Networked Sensor Systems (SenSys), Delft, The Netherlands, 2017. https://doi.org/10.48550/arXiv.1802.07584

[20] O. Koller, N. C. Camgoz, H. Ney and R. Bowden, "Weakly Supervised Learning with Multi-Stream CNN-LSTM-HMMs to Discover Sequential Parallelism in Sign Language Videos," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 9, pp. 2306-2320, 2020. https://doi.org/10.1109/TPAMI.2019.2911077

[21] Y. Chen, R. Zuo, F. Wei, Y. Wu, S. Liu, and B. Mak, “Two-Stream Network for Sign Language Recognition and Translation”, in Advances in Neural Information Processing Systems, 2022. https://doi.org/10.48550/arXiv.2211.01367

ACADEMIC VITA

RAYHAN RAHMAN

rayhanrahman2000@gmail.com | Allentown, PA

EDUCATION

The Pennsylvania State University | Schreyer Honors College — **University Park, PA**
College of Engineering | Bachelor of Science in Computer Engineering — *Class of May 2023*
College of Liberal Arts | Minor in Economics

PROFESSIONAL EXPERIENCE

Intel — **Allentown, PA**
SOC Design Intern — *May 2022 – Aug 2022*
- Joined Xeon Networking Group specializing in 5G chip design; learned about design flow from RTL to tapeout
- Assessed partition of chip utilizing VI, Synapses Fusion Compiler, and Altair Flow Tracer to run timing scripts

Vanguard — **Malvern, PA**
Software Development Intern (Platform Management Team) — *Jun 2021 – Aug 2021*
- Created framework for a data ingestion stream that transports ~$5T of market data per day from Vanguard's Portfolio Management Tool into a testing environment using various AWS services and scripts in JSON/Java
- Collaborated on agile team with 6 interns to construct a web application called Coffee Chats to facilitate a seamless internal networking experience using Electron, Figma, HTML, and DynamoDB

Mehta Prep Academy — **University Park, PA**
Director of Mathematics Tutoring — *Mar 2020 – Aug 2021*
- Organized a team of 4 mathematics tutors in promoting services, implementing effective tutoring methods, and reviewing created lesson plans/lecture slides prior to sessions

PROJECTS

CRUD File System — **University Park, PA**
Systems Programming Project — *Feb 2022 – May 2022*
- Designed a file system in C involving low level memory manipulation and functioning as the bridge between a disk drive and user commands. Created mount, unmount, read, write, seek, cache, and network functionality.
- Wrote ~1000 lines of code and gained experience in procedural programming, UNIX, pointer arithmetic, dynamic memory allocation, and debugging using gdb

Computer Vision and Object Detection Programs — **Allentown, PA**
Python Applications of OpenCV Libraries — *Dec 2020 – Dec 2020*
- Developed 10 programs using OpenCV libraries, cascade classifiers, and computer vision principles to detect/display shapes, colors, and human faces from images or a live camera feed
- Programmed basic document scanner to detect sheet of paper and allow users to save scans as PDF files

Printed Circuit Board Design — **University Park, PA**
EE210 Final Design Project — *Nov 2020 – Dec 2020*
- Designed and assembled a 5-stage circuit that takes in audio input and allows a user to modify volume, bass, and treble levels with an additional switch for karaoke mode to remove the vocals of a song

LEADERSHIP AND INVOLVEMENT

Schreyer Scholar Assistant — **University Park, PA**
Admissions, Equity, and Inclusion Lead — *Mar 2020 – May 2021*
- Collaborated with Director of Admissions and Dean of Equity/Inclusion to coordinate and support recruiting events and inclusive programming within the College
- Spearheaded Logistics Committee for the 2-day honors orientation program to welcome 300 incoming first-year scholars through a series of group-activities, informational sessions, and a campus tour

Penn State Infusion — **University Park, PA**
Registration Director | Technology Manager — *Sep 2020 – Present*
- Work on executive board and manage a committee to host an annual intercollegiate Bollywood fusion dance competition for student teams across the United States; committee coordinates teams and judges
- Corresponded with local hotels to negotiate the best agreement for participants' stay during the event weekend

Schreyer Consulting Group — **University Park, PA**
Vice President of Student Engagement — *Dec 2019 – May 2022*
- Revised a 50-page consulting handbook of example case questions and interviewing resources for members by working with executive board, consulting organizations, and other key stakeholders

Alpha Kappa Psi Professional Business Fraternity — **University Park, PA**
Webmaster | Inductee Class President — *Apr 2021 – Present*
- Manage the fraternity website; update content and pictures frequently to assist the recruitment committee

HONORS, SKILLS, AND INTERESTS

Honors: Dean's List (6/7 semesters), Schreyer Scholarship, SEDTAPP Most Effective Design Award
Languages: Python (Proficient), Java (Proficient), C (Proficient), Verilog(Limited), MATLAB(Limited)
Technologies: Atlassian Stack, AWS tools, GitHub, Linux Environment, Microsoft Suite, Multisim, Ultiboard
Interests: Bollywood Dance, Cloud Computing, Drones, *Hot Ones*, Jon Bellion, National Parks, Sudoku