An AI-Based Sign Language Translation System

Mrigank Khandelwal[#], Julia Gumieniak[#], Kausha Trivedi[#]

[#]*Khoury College of Computer Sciences, Northeastern University*

[1]khandelwal.mr@northeastern.edu

[2]gumieniak.j@northeastern.edu

trivedi.kau@northeastern.edu

***Abstract*— This project aims to integrate Computer Vision and Natural Language Processing to classify images of ASL hand signs to their respective characters and to generate meaningful sentences from the keywords signed. We employ Convolutional Neural Networks (CNNs) to classify ASL hand signs and we feed the output words to a Pegasus transformer to generate sentences from the signed words. The project attempts to bridge the communication gap between those who are hard of hearing and the rest of the world by eliminating the need to sign filler words thus making the process of communication faster and more efficient.**

***Keywords*— American Sign Language, Natural Language Processing, Convolutional Neural Networks (CNNs), Transformers, Computer Vision, Pegasus Transformer, OpenCV**

I. Introduction

Communication is a fundamental human right. Although most of us communicate by using spoken language, individuals that are deaf or hard of hearing express themselves through sign language. Sign language can consist of hand gestures, facial expressions, and body movements. American Sign Language (ASL) is a visual language primarily utilized in the United States and Canada [12]. It has its own system and rules, similar to other languages. In order to facilitate communication using ASL, one person must sign while the other person interprets those signs. When using ASL, speed and accuracy are of the utmost importance.

The first goal of this project was to enable accurate translation of ASL hand signs into words, thereby enhancing communication between the deaf and hard-of-hearing and the rest of the population. Our dataset consists of ASL signs ranging from A to Z, which serves as the foundation of our project. We curated this ASL image dataset and augmented it with pre-existing data. Our next objective was to automatically generate sentences based on these signed words.

We propose a system that involves the utilization of CNNs to implement translation of ASL hand signs and then passes the outputs to the Pegasus transformer for sentence generation. The subsequent portions of this report present the related work, our problem statement and methods, experiments and results, discussion and conclusion, and finally future scope.

II. Related work

Our initial source for ASL classification was a GitHub repository titled 'Sign-Language-To-Text-Conversion' [3]. Analogous to our project, they worked with a CNN when translating signs into words. This repository was informative in understanding the potential mechanisms that can be employed in translating ASL signs to words. Additionally, many other approaches have been leveraged to convert sign language into words.

Another source we identified converts sign language to English and vice versa [4]. They trained a CNN in the form of the ResNet50 model, used rolling average prediction to recognize their words without jitters in their prediction, and obtained a validation accuracy of 95%. A third paper utilized Single Shot Multi Box Detection (SSD), Inception V3, and Support Vector Machine (SVM) for translating sign language [5]. They claim an accuracy rate of 99.9% for their hybrid model. An additional approach to sign language translation involved using a transformer neural network to analyze both hand movements and facial expressions [6].

Furthermore, considerable research has been done on generating sentences from keywords. In 2018, Ziang Xie wrote a paper that serves as a guide on text generation [7]. This paper first mentioned earlier methods of text generation such as rule-based systems and probabilistic models. Next, it reviews encoder-decoder models, which is necessary for the transformer we utilize in our project. Finally, Ziang explains how to find solutions when text generation models do not behave as expected. This guide gives an excellent explanation for how text generation models work and is a good resource.

III. Problem statement and methods

This project aims to develop an AI tool that can generate complete sentences based on keywords signed in ASL from a live video feed. Keywords in this context refer to the bare minimal words of a sentence that can recreate the sentence without any additional contextual input. We divide this project into three submodules: Classification, Sentence Generation and Live Video System. The first segment deals with classification of ASL images into alphabets. The second module takes these keywords signed by joining these alphabets and generates a sentence based on those words. Finally, the Live video system attempts to use the first two modules in real time.

A. Classification

The first component of this System is a Sign Language Classifier that takes as input an image of a hand signing an alphabet in ASL and predicts the alphabet. The first step for this component as is for any machine learning application is to understand and finally develop a suitable dataset. After our preliminary literature survey, we discovered that many

analogous projects that attempt similar classifications often use images from one person's hands. This leads to overfitting. To combat this, we developed a new ASL dataset using data available online from various sources along with our own images. Thus, this way we developed a diverse dataset to combat overfitting.

Fig. 1 An example image that we captured for our dataset. This image depicts the ASL sign for the letter 'A'.

The next step is the classification itself. We identified two methods of classification. The first would be to train a CNN model on these images. This may give us good accuracy however, we were concerned about the potential tradeoffs associated with this approach which include slower training time, slower prediction time (since we need this model to work in real-time) and the model's potential to learn noise. Another approach would be to detect "landmarks" from images of hands and feed these landmarks into our Machine Learning model. Landmarks here refer to x,y and z axis coordinates of points on the hands (fingertips, palms and knuckles).

We chose the latter approach and in our preprocessing, we convert images of hands into landmark arrays using the OpenCV library. They can be 1-dimensioned flattened arrays of landmarks or 2-dimension landmark arrays with data points of type [x,y] depending on the model. The advantage of this method is that now our input data size is just [86][batch_size] or [42,2, batch_size] as opposed to [128,128,1][batch_size] in image based classification. Owing to a smaller input size, we can also attempt to use simpler ML models such as Random Forest and SVMs!

We experimented with SVMs, RandomForest, 1D Convolutional Neural Networks and 2D Convolutional Networks. For our 2D CNN, we used 2 convolution layers each followed by a maxpool layer. We used the Relu Activation for the convolutional layers and used Softmax activation for the last Dense Layer. We used the Adam Optimizer coupled with Cross Entropy Loss

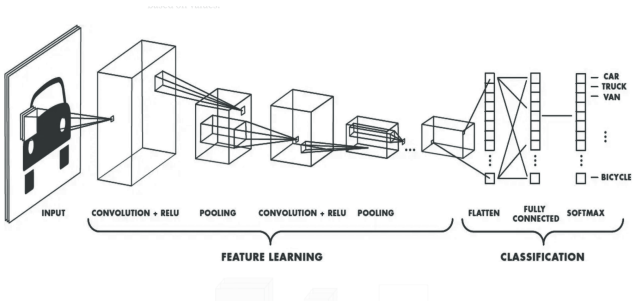

Fig. 2 Architecture of the CNN used. Here the input is a [42,2] 2D array consisting of the X and Y coordinates of the landmarks.

The signed letters are combined into words. Words are separated by breaks in signing. These keywords are sent off to the Sentence Generation module for input.

B. Sentence Generation

The second component of the Translation System is to generate accurate and meaningful sentences from the signed words. This component takes as input a list of words and is tasked to generate a sentence for each list. This is essentially a "Paraphrasing" task. For example, we would require our system to take an input of ["rain", "Boston", "till", "nine:"] and generate a sentence analogous to "There is rain in Boston till nine o' clock."

During an initial analysis of Natural Language Processing Techniques that might be effective in paraphrasing and summarization, we discovered Hidden Markov Models (HMMs), Transformers (BERT, BART, Pegasus) , Word Vectors (Word2Vec, Fasttext) and GPT (GPT-2). A brief comparison of these techniques is as follows in Table 1.

TABLE I
COMPARISON OF NLP TECHNIQUES FOR SUMMARIZATION/PARAPHRASING

Summarization Techniques		
Techniques	**Benefits**	**Trade Offs**
HMMs	Fast and efficient	Limited to Extractive Summarization
Transformers	Bidirectional; Ability to Capture Long-range Dependencies; Support Abstractive Summarization	Training and Fine-tuning is resource intensive.
Word Vectors	Fast and efficient	May capture context incorrectly.
GPT	Support Abstractive Summarization	Not bidirectional; Training and Fine-tuning is resource intensive.

After weighing the benefits and tradeoffs, we decided to go ahead with Transformers owing to their Bidirectionality and performance on summarization tasks.

Transformers are sequence to sequence neural networks that have been revolutionary in the field of Natural Language Processing. They are trained on massive text corpuses, often millions of web pages. Unlike traditional models, transformers introduce a bidirectional self-attention mechanism that allows

them to consider the entire input sequence simultaneously, capturing long-range dependencies effectively. This bidirectionality is crucial for tasks such as language translation and summarization, where understanding the context from both ends of a sequence is paramount. The transformer architecture further incorporates an encoder-decoder structure, where the encoder processes the input sequence, and the decoder generates the output sequence.

The problem statement here was to expand on keywords to generate sentences which is a form of Abstractive Summarization. Abstractive Summarization refers to summarization by generating an entirely new sequence based on the learnings from the input sequence. Popular transformers such as BERT and BART excel at another form of summarization called Extractive Summarization which is summarizing an input sequence by collating the most important parts of the sequence. We decided to go forward with the Pegasus Transformer which is the State of the Art Transformer for Abstractive Summarization.

The Pegasus Transformer uses a Masked Language Model (MLM) as its base encoder. It introduces a Gap Sentence Generation (GSG) decoder which essentially masks sentences from the input and tasks the decoder to predict these sentences from the rest of the input. The GSG decoder of the Pegasus transformers contribute to its excellent performance on Abstractive Summarization tasks.

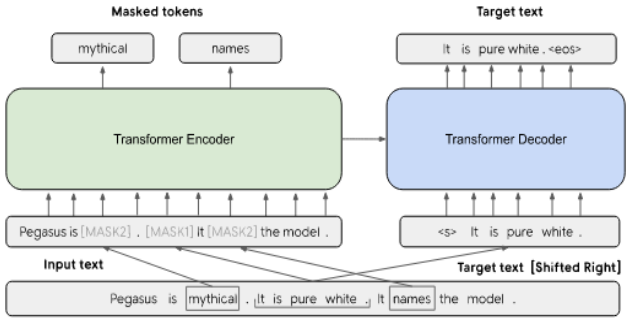

Fig. 3 The Pegasus transformer. The MLM encoder and the GSG decoder can be seen in action here. The MLM encoders mask tokens in the input sequence. The GSG decoder masks the sentence "it is pure white" and tasks the decoder to predict this sentence.

C. Live Video Feed

The last part of the project is to integrate both these systems and use them on real time feed. We used OpenCV for live video capture via a webcam. In our experiments, we used our laptop's camera. We use OpenCV to detect hands in the video feed. If hands are detected, we extract the landmarks and feed it to our ASL classifier. It generates words and once a period is detected (based on pause in signing), the words are passed to the Sentence Generator and it generates a complete sentence.

IV. Experiments and Results

A. Classification

Our dataset comprises approximately a 1000 images for each letter in the alphabet. We used OpenCV to get the landmarks of hands which are the X and Y coordinates of fingertips, knuckles and palms. For the RandomForest model and the Support Vector Machine model, we flattened these landmarks consisting of X and Y coordinates into a 1D array and fed this as the input to the model. For the CNN, we used a 2D array of size (42,2) as input. CNNs gave us the best accuracy amongst all the other methods and we decided to use CNNs in our final model.

TABLE 2
Comparison Of Accuracies

Accuracy		
Model	**Input Size**	**Validation Accuracy**
Random Forest	batch_size * 84	89%
Support Vector Machine	batch_size * 84	90.53%
Convolutional Neural Network	batch_size * [42,2]	96.3%

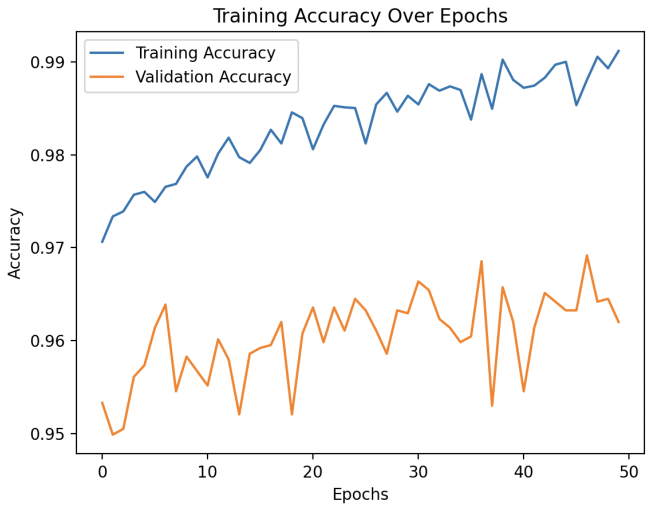

Fig. 4 Training and Validation accuracies for the CNN model.

CNNs gave us the best accuracy amongst all the other methods and we decided to use CNNs in our final model. We have 26 classification classes, one for each letter in the alphabet. Thus, it becomes important to further analyse which classes perform well and which ones may benefit from some additional data.

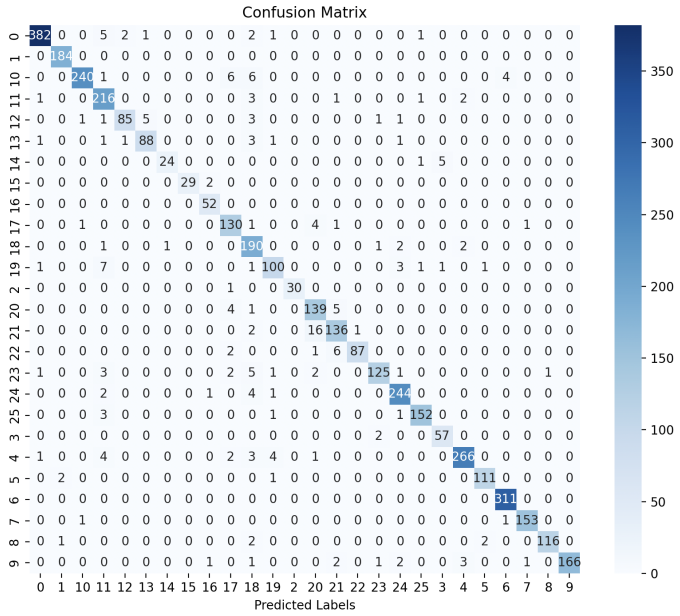

Fig. 5 A confusion matrix of the classification of alphabets.

From this, we can conclude that the model does not perform well on some characters, for instance on C, P and Q while it performs exceptionally well on others such as A.

B. *Sentence Generation*

Initially we tried using a Pretrained Pegasus Model which was fine tuned for paraphrasing called "Tuner007". This gave us good results.

Input: `rain atlanta till nine`
Output: `There is rain in Atlanta until nine`

We gave the model a list of such incomplete inputs and tasked the model to generate a fully complete sentence for each such input.

We also attempted to finetune our own custom model. The first step was to select a corpus to fine tune the pretrained pegasus-large model. For this, the Microsoft Research Paraphrase Corpus (MRPC) was selected. MRPC is a benchmark dataset widely used in natural language processing and machine learning. Created by Microsoft Research, MRPC is specifically designed for paraphrase identification, containing sentence pairs annotated with binary labels indicating whether they convey the same meaning. Fine-tuning the Pegasus-large pretrained model was computationally expensive. We froze the encoder weights to overcome this resource limitation. However, we obtained sub-optimal results with substantial hallucinations after fine-tuning for 5 epochs.

Input: `rain boston`
Output: `rain is forecast to continue through the weekend in Boston and New York city.`

C. *Live Video Feed*

OpenCV's hand detection is accurate and is able to detect hands from live videos with great precision. We were able to display the generated sentence after a period was detected.

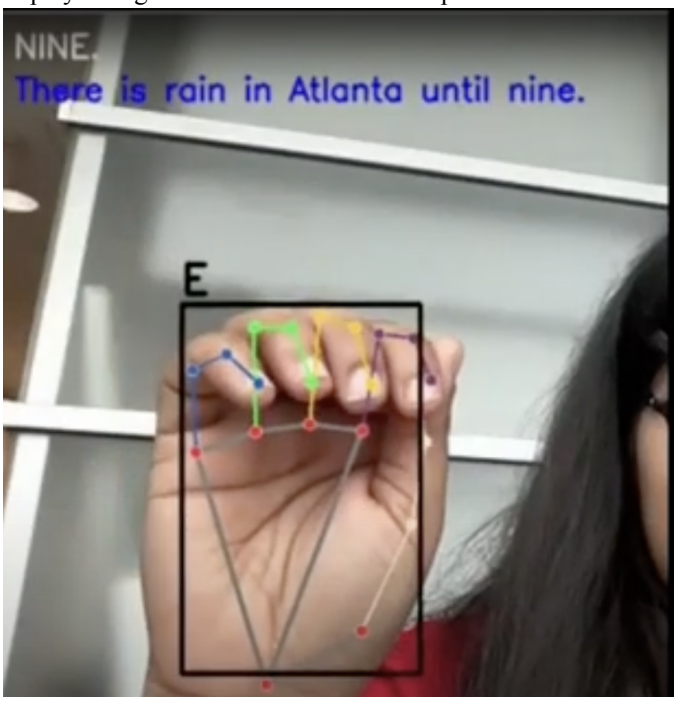

Fig. 6 A screenshot of the system in action on a live video feed. The completed sentence in blue is generated using the signed keywords, the latest being "nine".

V. Discussion and Conclusion

In this project, we developed an AI system that uses CNNs for classification of ASL images into characters and a Pegasus Transformer that generates sentences from words signed. One key highlight of this project was the dataset. A lack of readily available diverse ASL hand signs dataset urged us to create our own dataset, combining smaller readily available datasets online. We attempted to detect hand landmarks from images of hands in our preprocessing and feed that into our CNN model. This significantly reduced training time without compromising accuracy. Overfitting was a major challenge and we worked on improving our data iteratively in addition to adding dropout layers. We experimented with various Transformers for Sentence generation and decided to use Pegasus Transformer for its prowess in Abstractive Summarization. We attempted to fine tune it on the MRPC dataset however we were limited by our computational resources.

VI. Future Scope

This project could benefit greatly by increasing the size and diversity of the training data. It would also be interesting to compare the performances of image classification using CNNs and our implementation of landmark detection and landmark classification. Additionally, fine-tuned transformers such as Tuner007 are specifically designed for paraphrasing.

Fine-tuning the Pegasus model on the MRPC dataset is a promising avenue and it could be explored further based on the availability of resources. Lastly, an additional feature that we found really interesting and at the same time could integrate coherently with this system is a Hidden Markov Model based autocorrect for signed words. During our experiments, we found that often one letter of a word would be misclassified which may impact Sentence Generation. Such an autocorrect system would make the system more robust to these misclassifications.

References

[1] Jingqing Zhang, Yao Zhao, Mohammad Saleh, Peter J. Liu, "*PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization"*, International Conference on Machine Learning, 2020.

[2] Tuner007 pretrained Pegasus model for paraphrasing, Available: http://github.com/tuner007/paraphrase_sentences

[3] emnikhil Sign Language to Text Conversion. Available: https://github.com/emnikhil/Sign-Language-To-Text-Conversion

[4] V. D. Avina, M. Amiruzzaman, S. Amiruzzaman, L. B. Ngo, and M. A. A. Dewan, "An AI-Based Framework for Translating American Sign Language to English and Vice Versa," Information (Basel), vol. 14, no. 10, pp. 569-, 2023.

[5] R. H. Abiyev, M. Arslan, and J. B. Idoko, "Sign Language Translation Using Deep Convolutional Neural Networks," KSII transactions on Internet and information systems, vol. 14, no. 2, pp. 631–653, 2020.

[6] David Wan, Mohit Bansal, FactPEGASUS: Factuality-Aware Pre-training and Fine-tuning for Abstractive Summarization

[7] G. Strobel, T. Schoormann, L. Banh, and F. Möller, "Artificial Intelligence for Sign Language Translation – A Design Science Research Study," Communications of the Association for Information Systems, vol. 52, 2023.

[8] Z. Xie, "Neural Text Generation: A Practical Guide," arXiv.org, 2017.

[9] MRPC dataset: https://huggingface.co/sgugger/glue-mrpc

[10] BERT transformer used with the MRPC dataset: https://www.kaggle.com/code/deepaksinghrawat/finetuning-bert-on-mrpc-corpus-using-fastai

[11] https://docs.opencv.org/3.4/index.html

[12] "What is American Sign Language?" nad.org.

Link To Github Repository

The code and instructions for exploring this application can be found at . `https://github.com/Mrigankkh/ASL-2-Text`

Python Libraries Used

1) OpenCV - it stands as a linchpin in the project's architecture, orchestrating real-time video capture, image processing, and hand landmark detection.

2) MediaPipe - The integration of MediaPipe adds a layer of sophistication, streamlining hand tracking and landmark detection.

3) Keras and PyTorch - The CNN for Image Classification was developed in Keras.

4) MatplotLib: Used to plot Accuracy graphs for CNNs.

5) Transformers - The Pegasus Transformer was used, fine-tuned using this library.

6) Datasets - Used for the Glue-MRPC dataset

7) Numpy - Used for manipulating (padding , reshaping, etc) the inputs to the CNNs.