Live Non-isolated Sign Language Recognition Using Transformers

Daniel Yang
Stanford School of Engineering
dy92634@stanford.edu

Ethan Farah
Stanford School of Engineering
efarah@stanford.edu

Abstract

The inability to express verbal language communication is a monumental barrier for thousands of individuals globally. The utilization of American Sign Language (ASL) offers some partial solution to this problem, enabling human communication through hand gestures. However, the infrequent familiarity with sign language among the general public still results in substantial hurdles for those unable to speak. This is made worse by the fact there is not currently any live on-demand sign language interpreters. In this research project, we attempt to address the major challenges facing engineering a sign language interpreter. Our goal is to develop a model that can accurately translate either live video feed or video files of ASL into English text. We trained several models with different architectures and training sets and evaluated their performance and potential uses in the real world. From existing literature, we found there is currently no accurate live ASL translation models. Our work builds off existing video classifiers that identify ASL words from videos. The inputs for our models were either live video feeds or existing videos of ASL and the outputs are a sequence of translated English text. We found frame analysis with Google's MediaPipe Hand Recognition provided the best performance on live video to text translations given its lightweight nature. For longer video to sequence translations, a Vision Transformer (ViT) model with transfer learning provides the most accurate results. These two models were trained to recognize letters from the ASL alphabet, and far exceeded existing benchmarks. For the expanded ASL vocabulary with high volume of classes, the Video Vision Transformer (ViViT) model or our Isolated MediaPipe Transformers Model is more suitable. We've also identified several data augmentation and generative techniques to enhance our models' translation ac curacies as well as isolated several challenging points for further exploration.

1. Introduction

One of the primary applications with enormous positive potential for computer vision is interpreting human gestures and motions. When applied to the realm of American Sign Language, there is ample space for computer vision to eliminate existing linguistic barriers for the deaf and hard of hearing community. We plan on creating a sign language interpreter, which can both translate real time through live video feed and analyze existing video files. We want to discover the most effective model architecture at capturing human hand motions by comparing different architectures. This would guide the most effective implementation of this model on smaller devices and under different contexts.

1.1. Problem Statement

We aim to solve the problem of non-isolated live sign language interpretation. The problem of live video interpretations is capturing important parts of the video (hands, arms, face, etc.) and with the context of the surrounding frames in addition to previous and following signs. The sheer amount of data presented by each video is astounding, given the frames, pixels, and dimensions. This memory usage has severely limited our computers' abilities to preprocess and manipulate our datasets. Our inputs to the models are videos of sign language sentences and words. The output would be a translation of a particular word/sentence.

The other significant challenge for this project is the lack of sequence to sequence data. Most existing neural machine translations (nmts) are trained on an abundance of sequence to sequence data in both the original and target language. However, ASL possesses no such large datasets. Thus, all models are trained on short videos that capture only one word. A live ASL translator would have to capture not only the class label for words, but also distinguish changes or transitions between words.

After some initial tests and models, we've decided two key tasks must be accomplished for a viable live ASL interpreter. Firstly, correct classification of all 26

letters of the English alphabet enabling users to spell any words in speech. Secondly, an extended model able to identify a large quantity of gestures corresponding to words in a vocabulary. The latter necessitates processing and identifying motion of the hands and body.

We started by first training classifiers. The general approach for our models was to train a high accuracy classifier on either images or videos, run these classifiers on a subset of frames from test data with further algorithmic scaling, and pass a noisy sequence into a word model to obtain our fluent translation. To ensure our project is not lagging behind existing work, we ensured our classifiers outperform existing baseline models. Our baseline model for videos is the ACM paper which achieved an accuracy of 62.79% on short videos of ASL words. Our baseline for identifying ASL letters from images is 83%, accomplished by an IEEE paper by Raval and Gajjar. After outperforming the baseline models, we moved to developing a live ASL interpreter. This led us to developing several video processing techniques to continuously apply our classifiers to a high resolution video stream.

2. Related Works

2.1. MediaPipe Coordinate Classification

MediaPipe is a hand landmark identifier developed by google [17]. It's a real time video processor that is able to locate key points of the hands in videos. MediaPipe gives coordinates of the palm and each finger found in the input image. The key coordinates of the hand locations can be used to train models to recognize and process human hand gestures. We plan to explore the potential for MediaPipe within our lightweight live ASL interpreter. The following studies also feature the use of MediaPipe.

A study by Paul and Walid compares three different model architectures to interpret sign language, a ResNet CNN, a gated recurrent network (GRU), and the long short term memory (LSTM) [13]. The study finds the LSTM performing the best out of the three. All three networks are trained on data preprocessed by MediaPipe, making all the inputs tensors of hand coordinates. The tensors are then formatted and passed into their respective models. All three models have identical dropout rates, loss function calculations, activation functions, and optimization. The LSTM, which performed better than the GRU and the ResNet CNN achieved f1 scores of around 93.8% across multiple classes. The models are trained on images with 26 classes, corresponding to the 26 letters of the English alphabet. This serves as a great benchmark for our MediaPipe alphabet classifier. They also trained a LSTM for videos comprised of three classes, and achieved an accuracy of 94.3%.

Another study by Luna and Jimenez serves as a great benchmark and framework for the work we are doing for sign language interpretation. The paper aims to highlight the most important features regarding interpreting sign language, in other words, what areas of images/frames should models focus on to make their classifications. The creators preprocess their training data through MediaPipe. In addition, they also use VisionAPI which is another video feature extraction tool we could experiment with to augment our dataset before training. The combined tensors from these two preprocessors are concatenated for their inputs. For their training model, they fine tuned SPOTER [3], a transformers based model that uses non traditional positional encodings and "Class Query," a randomly initialized learnable parameter before decoding instead of the traditional contextual sequence. They achieved a test accuracy of 62.79% on videos which serves as our most basic baseline. [9]

In addition to reviewing papers, we scoured the internet for "homemade" architectures to solve video recognition. One such example is the code of the first-place winner to the now-finished Kaggle-hosted and Google-sponsored competition under the title Isolated Sign Language Recognition, which ranked users' abilities to train models that could classify isolated videos of humans performing sign language into one of 250 different classes. Each of the $94,744$ videos provided for the competition had already been preprocessed using Google's MediaPipe API [6] to provide landmarkers. The first-place user, Hoyso48, identified 118 specific landmarks to train on and used a cycle of 3 1-dimensional convolutional layers followed by 1 transformer layer in TensorFlow to achieve a validation accuracy of approximately 80% on isolated video segments. This model offers a great blueprint for our own custom model architecture for classifying video words, specifically the authors insight of tracking not just position, but also instantaneous velocity and acceleration over time. [7]

Another solution to this competition, also written in TensorFlow, by Mark Wijkhuizen uses a custom-built embedding layer followed by a transformer layer and lastly by a classifier layer. [16] A hallmark of this solution, besides the model architecture, was the very efficient and straight forward data processing, specifically in terms of determining the dominant hand at use in a given video.

2.2. Non-Landmark Based Hand Classification

In a study by Bantupalli and Xiem, human gestures in videos are classified to their respective sign language interpretations (Bantupalli) [2]. The data was gathered

from custom recorded videos, where videos were broken down into frames then converted to images. To classify the images, the authors used Inception [15], a CNN based image recognition model developed by Google. The authors pass both the outputs from the Softmax layer and the Pool layer into a LSTM for classification. The results were compared and the authors achieved a test accuracy of 91% using Softmax and 58% using the Pool Layer. This inspired us to pursue some LSTM variant for our initial models.

We discovered another study that tries to classify ASL alphabet letters. Raval and Gajjar designed a convolutional neural network with Softmax to map images to their respective letters. They ended up accomplishing a final classification accuracy of 83%, serving as a benchmark for our ASL alphabet classifier. Crucially, they made use of a skin color mask, where HSV or Hue Saturation Value is used to detect skin color of hands to apply masking. Further blurring to control for noise and dilation with erosion is applied to enable better generalization of the model to a greater selection of skin tones (Raval) [14].

Another study conducted by Deepali performed SVD to classify images into their respective alphabet letter classes. We don't plan to explore SVMs in this project but we'll take their 95.31% accuracy as a good baseline [10].

3. Methods

3.1. Data Preprocessing

3.1.1 Image and Video Preprocessing

We began our project with a simpler problem: individual character recognition. One of the first steps was to generate even more training data for our ASL character alphabet. Since our "ASL Alphabet" dataset from Kaggle contained no backgrounds for hands, we added backgrounds with random RBG values so our models learn how to separate background noise. Additionally, we stretched, flipped, rotated, scaled, and translated the images. We applied a random combination of these transforms to a sample from our dataset. Through these extra transform combinations, we were able to increase our dataset by 20% arriving at just over $100,000$ images. For our extended vocabulary model of videos, a different approach was necessary. To compress the amount of data needed, each short video (around 4 seconds each) was evenly divided into 16 frames. Each frame was converted to 1 channel (grayscale) with 224 by 224 dimensions. Our final training tensors were shaped N x 16 x 1 x 224 x 244.

3.1.2 MediaPipe Preprocessing

For models that take MediaPipe coordinates, images (and later videos) are first passed through Google's MediaPipe algorithm to generate a set of "landmarks" and their respective x, y, and z coordinates. Specifically, the following landmarks were tracked and stored for training: left hand, right hand, lips, pose, and nose; lips, pose, and nose landmarks were occasionally omitted, as they are not fully necessary for letter translation (which is done by the hands).

Before performing any augmentations, each data point is normalized about its 17th left hand landmark as a sort of reference point, with NaN values being either filtered out or replaced with a predetermined pad value [7]. For video inputs, we implement a dynamic frame handling algorithm that first removes all entirely empty frames then either crops down to, pads up to, or repeats a given video as many times as possible in a set maximum frame length (generally 64 from the Google Isolated Sign Language Recognition Kaggle competition dataset) [16]. In some tests, we compute the first derivative (velocity) and the second derivative (acceleration) [7] across frames for each landmark to capture the dynamics of hand, lip, and pose movement:

$$v_t = x_t - x_{t-1}$$
$$a_t = v_t - v_{t-1}$$

where x_t represents the landmark position at time t. Lastly, before performing any augmentations, left versus right side dominance was determined by comparing the number of non-NaN left-sided landmarks and comparing that to the number of non-NaN right sided landmarks. [16]

3.2. Character Recognition Model Architectures

We trained multiple models, four were trained to identify ASL alphabet letters and six were trained to classify words from our expanded vocabulary. Our ASL alphabet classifiers are trained on images whereas our expanded vocabulary models are trained on videos. In the following sections, we outline and detail the model architecture for some of our significant models. These models could be conjoined to enhance total translation ability from ASL to spoken/textual English.

This character recognition portion of our study is intended to be a base point and key component of live ASL translation both with direct image useage and MediaPipe useage. To experiment with live translation, we perform a comparative study by training multiple image-identification models.

3.2.1 Note on Activation Functions

We will be employing various activation functions throughout our model development process, namely ReLU, LeakyReLU, GELU, and SiLU:

Function	Formula	Output
ReLU	$f(x) = \max(0, x)$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$(-\infty, \infty)$
GELU	$f(x) = x\Phi(x)$ $\Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$	$(-\infty, \infty)$
SiLU (Swish)	$f(x) = \frac{x}{1+e^{-x}}$	$(-\infty, \infty)$

3.2.2 ViT Model

With our augmented dataset for ASL letters, we engineered our first model by finetuning a Vision Transformers (ViT) computer vision architecture [5]. Images are broken down into P smaller C x C panels forming grid, going from N x W x H to N x P x C^2. These panels are then sequentially passed into our transformers encoder and into our multi-headed attention mechanisms.

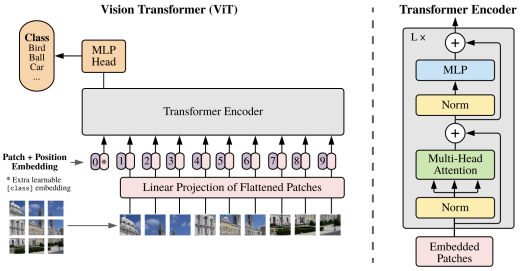


Figure 1: Architecture of the Vision Transformer (ViT) model.

The above visualizes our model architecture. In our eventual model we chose to divide our images into a 4 by 4 grid. We were able to take advantage of transfer learning by starting with pretrained weights from the ImageNet dataset [4].

3.2.3 Isolated MediaPipe Character Model: Linear Neural Network

The first MediaPipe character recognition model we created is a fully connected neural network, which starts by taking the pre-processed inputs into a dense layer that maps them to a higher dimensional space, facilitated by an activation function from the above choices and batch normalization to ensure effective forward propagation of gradients. Dropout is used after each activation to reduce overfitting risks. The network architecture progressively reduces the dimensionality through three fully connected layers, incorporating the same non-linearity and normalization strategies, and ending with a layer that outputs class scores, which are transformed into probabilities using softmax.

3.2.4 Isolated MediaPipe Character Model: CNN, LinNN, normNN

The second significant model we trained was a MediaPipe character recognition model employs a 1-dimensional convolutional neural network (CNN) that processes the input with 16 filters, each with a kernel size of 3 and padding of 1, maintaining the dimension while capturing basic spatial features. This is followed by batch normalization, and max pooling. A second 1-dimensional convolutional layer increases the depth to 32 channels, allowing the model to learn more complex patterns before another round of activation, batch normalization, and max pooling. The network ends with a set of fully connected layers where the features are flattened and passed through dense layers with dropout in between to prevent overfitting, and ultimately using a final classification layer that outputs probabilities for each class through a softmax activation.

3.3. Gesture Recognition Model Architectures

The following subsections detail the two models we created to recognize a wider vocabulary of ASL gestures. Our ViViT model was trained on our raw videos whereas the inputs for FFIMpGIM were our MediaPipe landmark coordinates from the Google Isolated Sign Language Recognition dataset.

3.3.1 ViViT Model

ViViT is a transformer model architecture utilized for video classification [1]. ViViT utilizes spatiotemporal attention and patch embeddings followed by a temporal self attention block. The spatiotemporal attention mechanism is based off the ViT Model described above, and captures interactions between captured tokens in the same time frame. Afterwards, the output of the spatiotemporal self-attention block is passed into the temporal self attention mechanism, which captures interactions between different time steps.

Because video tensors take up an excessive amount of space, we only parsed our videos with one depth channel, as detailed in the data preprocessing step. This drastically cut down on the space we needed to store our tensors. We created a custom CNN convolutional layer to perform this operation.

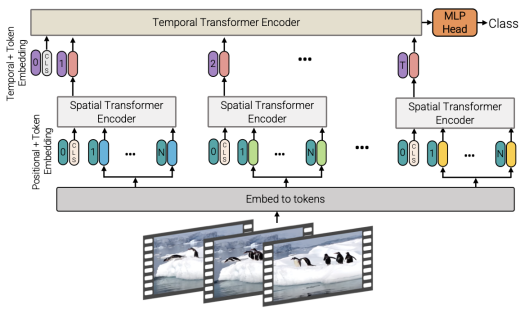
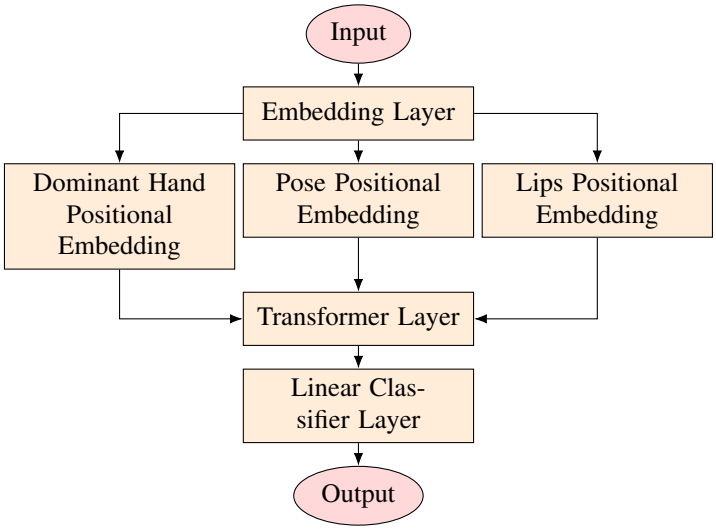


Figure 2: Architecture of the ViViT model.

3.3.2 Frankenstein-more-like-Freakystein Isolated MediaPipe Gesture Identification Model (FFIMpGIM)



FFIMpGIM's most unique feature is its complex embedding layer [16] that processes raw input data (specifically spatial and sequential features) into a form that the following transformer layer will be able to process, focusing on capturing both static and dynamic aspects of the predetermined dominant hand landmarks, lip landmarks, and pose landmarks. It begins with positional embedding, assigning a unique vector to each possible position in the input sequence to help the model understand frame ordering. Then we perform feature-specific embedding to relate spatial features and relationships between body parts. Learnable weights are set and adjusted during training to determine the emphasis placed on each landmark type. Lastly, the combined and weighted landmarks are passed through a set of linear layers and returned to the PyTorch's dynamic Transformer layer [12].

3.4. Training

3.4.1 Learning Rate Scheduler

Cosine Annealing: The learning rate is adjusted using a cosine annealing schedule, which decreases the learning rate following a cosine curve between initial learning rate settings and zero. This creates better performance through encouraging convergence by reducing the learning rate as training progresses:

$$\text{lr}(t) = \text{lr}_{\text{max}} \left(1 + \cos\left(\frac{\pi \cdot t}{T}\right)\right)/2$$

where t is the current epoch, and T is the total number of epochs. All of our models feature a scheduler which periodically updates our learning rates after each epoch.

3.4.2 Loss and Optimization

Cross-Entropy Loss: Most of our models use the categorical cross-entropy loss, which compares the predicted probability distribution across classes with the actual distribution (one-hot encoded):

$$\text{Loss} = -\sum_{c=1}^{C} t_c \log(p_c)$$

where t_c is the target probability for class c (1 for the true class and 0 for others), and p_c is the predicted probability for class c.

Customized Cross-Entropy: We started with our normal cross entropy loss function. But we soon discovered that certain sign language classes are often confused with each other. For example the letters of m and n are very similar and gestures for sad and friendly are often confused. On our final trained models, we implement the custom cross-entropy loss function with defined cost matrix W of size $C \times C$ where C is the total number of classes. Each element W_{ij} represents the penalty for classifying an actual class i as class j. To penalize for misclassification between c_1 and c_2, we can increase $W_{c_1 c_2}$ and $W_{c_2 c_1}$ to higher than one.

$$\text{Loss} = -\sum_{c=1}^{C} W_{tc} \cdot t_c \log(p_c)$$

3.5. Post-Processing

For our ASL alphabet classifier models, there are several crucial postprocessing steps to formulate coherent translation of the input videos. Suppose $l_1, l_2, ...l_C$ are the logits produced by our model for C different classes. We want to scale each logit by the frequency of which letters appear

in English text. Thus, more common letters with have a higher chance of being represented and vise versa. After processing Ethan's old essay on *Hamlet* by Shakespeare, let $f_1, f_2, ..., f_C$ be the frequency count that each letter of classes C appears. The following logit scaling is then performed

$$\text{for } i \in C, l_i' = l_i \dot{(}1 + \frac{log(f_i)}{\sum_{z=1}^{C} log(f_z)})$$

We pass the scaled logits through Softmax to calculate our probability distribution for each letter.

3.6. Live Video Feed Processing

Live video feed processing with our classifiers requires addressing several challenges. Because there is no large dataset on ASL to English sentence translations, we cannot train sequence to sequence models. Instead, we adapt our classifiers to process video streams.

For our image classifiers on ASL letters, we create matrix to implement a sliding window approach. The matrix W is dimensions of T x C, where T is the duration of the window and C is the number of classes. We extract between 10 and 30 frames every second from our video feed. For every processed frame, we first shift all rows up one in W, then we load the weighted probabilities into the last row of index T - 1. Then we compute the average probability across all columns:

$$\bar{p}_j = \frac{1}{T} \sum_{i=0}^{T-1} W_{i,j}$$

If the probability corresponding to class c exceeds our threshold 0.8, we append it to our output sequence, otherwise append a space.

$$\text{Output} = \begin{cases} \text{class } c & \text{if } \bar{p}_c \geq 0.8 \\ \text{space} & \text{otherwise} \end{cases}$$

The reason for this window approach is to avoid sudden input shifts. For a character to be successfully rendered, it must appear for some minimum duration. This enables the model to distinguish spaces from transitions.

Our video classifiers follow a similar approach. After extracting 30 frames every second, the last 64 filled frames in our video stream are fed into our video classifiers. We append label to output sequence if probability threshold of 0.6 is exceeded.

3.7. Fluent Translations

The previous subsection details how the classifiers are fed information from video streams. However, the direct outputs from these classifiers are not necessarily fluent. To ensure proper translation of our model, we feed the outputs into a language model. We could have trained a simple BERT model to turn our classifier outputs into their correct translation, but decided the OpenAI API for GPT4o works just as well. The following tables outlines the inputs and output sequences we get with OpenAI's API.

Table 1: Word model for alphabet

Input	Output
iiiiii aaammmm gggg-rrrooooattttt	i am groot
sssnmaaarrrrtttiiiieeesss ddddeeemmtttt-teeeemmccccctttt-teeeeddddd	smarties detected

Table 2: Word model for words

Input	Output
i hungry	i am hungry
i go football	i go to the football game

4. Dataset Selection

4.1. ASL Alphabet

To train our models to recognize the letters of the alphabet, we started with a dataset on Kaggle compiled by Nagaraj [11] titled "ASL Alphabet." This dataset contained over 87,000 images spanning 26 classes for the alphabet. We further augmented the dataset by using our own hands. Several videos were taken in different lighting for each class. These videos were parsed by sampling two frames every second and the frame image resized to fit out model.

4.2. Expanded Vocabulary

There are over 40,000 words in sign language. We wanted to supplement our model with recognizing some of the most common words. These gestures can only be represented by video so we started with a dataset found on Kaggle called the World Level American Sign Language (WLASL), which consists of approximately $12,000$ short videos of humans performing signs of common words (2000 words total) that is already split into training, validation, and test sets. While this dataset is very diverse in the vocabulary it provides and convenient in the way it is presented, it only leaves with about 5 to 6 examples per word, which is very sparse. [8]

As predicted, using this dataset as given resulted in extremely high losses (> 7.5) that didn't drop and extremely

low validation accuracies ($< 0.004\%$, which is roughly a 1 out of 2000 chance, equivalent to guessing) after 200 epochs of training on models trained with different hyperparameters and architectures (layers chosen from the following: 1D Conv, ReLU, LSTM, Linear, and Dropout). We searched for another dataset that contained fewer words/labels and more density of examples, which we landed on by using an open sourced data set from a Kaggle competition. The dataset contains $94,744$ different examples with 350 different labels.

5. Results and Discussion

The primary metric used to evaluate the quality of our models is *accuracy* on test data. Simply, we look to maximize the ratio of images/videos the model correctly identifies out of the total images/videos. We also qualitatively adjust our model by interacting with its live implementation we engineered.

5.1. Character Recognition Models

Table 3: Models vs Baseline for Letter Identification

Model	Test Accuracy
Bantupalli (baseline)	91%
Raval (baseline)	83%
Paul (baseline)	93.8%
Deepali (baseline)	95.31%
MediaPipe Linear NN	98.76%
MediaPipe CNN	98.57%
MediaPipe + Norm*[1]	88.9%
ViT	99.84%

*Model was coded but was not included in methods due to model architecture insignificance.

From the table seen above, we can comparatively evaluate our models against each other and our baselines. The accuracy measures what percent of ASL letters in images the model was able to successfully guess. Most of our models handily beat the baseline models with our best version of the ViT model achieving a test accuracy of 99.84%. This suggests our modifications to parameters, loss, and data were successful

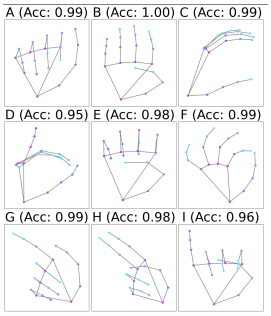


Figure 3: Class Visualization MediaPipe Isolated Character Best Model

This figure helps us visualize and simplify exactly what the MediaPipe is sending to our models. We have plotted the hand coordinates of the palms and the fingers for this visual

5.2. Parameter Selection for MediaPipe

Table 4: Hyperparameters for Linear NN and CNN Grid Search

Hyperparameter	**Values**
Learning Rate (lr)	0.001, 0.0005, 0.0001
Batch Size	16, 32, 64
Hidden Size (linear)	128, 256, 512
Dropout Rate	0.3, 0.5, 0.7
Optimizer	Adam, SGD, RMSprop, Adagrad
Number of Epochs	5, 8, 11

5.3. Training non-MediaPipe Models

The parameters for our ViT and ViViT models were trained through trial and error as well. Below are the hyperparameters used:

Table 5: Hyperparameters for ViT and ViViT Models

Hyperparameter	Values
Learning Rate (ViT)	0.01, 0.001
Learning Rate (ViViT)	0.001, 0.0001
Batch Size (ViT)	16
Batch Size (ViViT)	2
Number of Epochs	10
Optimizer	Adam

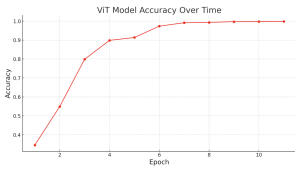


Figure 4: Accuracy of the ViT model

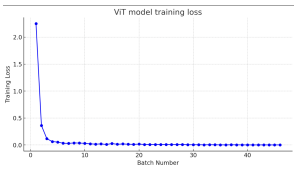


Figure 5: Loss of the ViT model

Figures 4 and 5 depict the loss and training values for our ViT models over time. The steep dropoff for loss in training suggests very health behavior.

5.4. Gesture Recognition Models

Table 6: Models vs Baseline for Video Identification

Model	Test Accuracy
Paul [3] (baseline)	94.3%
Luna [13] (baseline)	62.79%
Hoyso [250] (baseline)	80%
ViViT [400]	28.9%
LSTM: flattened videos* [400]	1.4%
Transformers: flattened videos* [400]	1.2%
MediaPipe: Custom Encoder [250]	61.83%
MediaPipe: Only Vel/Acc [250]	0.4%
MP: Custom Encoder + Vel/Acc [250]	73.56%

*Model was coded but was not included in methods due to model architecture insignificance.

The accuracy measures what percent of gesture videos the model was able to successfully guess. The comparisons in the above table are not entirely fair. Having more classes will make training a high accuracy model naturally a lot more difficult. The numbers in brackets next to the model name indicate number of classes for the classifier. The baseline models, with exception to Hoyso, had significantly less class labels than our models. Our ViViT model was trained to identify 400 words in the sign language vocabulary, which is exceedingly difficult. Overall, our MediPipe model with our custom encoder and augmented data for velocity and acceleration performed the best among our models, achieving an accuracy of 73.56%. However, we fail to beat our baseline model of Hoyso, which achieved an accuracy of over 80% despite having the same amount of classes [7].

5.5. Analyzing Losses and Accuracies

After we finalized both the MediaPipe linear NN and CNN models with the optimal hyperparameters according to the grid search while employing 5-fold cross validation, we yielded the loss and accuracy over time seen in figure 4. All of the losses and accuracies begin to converge early, avoiding issues with exploding or vanishing gradients.

With peak model accuracy at 98.76% and 98.57% for the linear and CNN models respectively, we were ready to commence with live camera testing.

We were able to complete training and validation on both the MediaPipe transformer model and the ViViT model as seen in figures 6 and 7 after very significant failures with other attempted models (which could barely peak past 0.04% accuracy—essentially guessing).

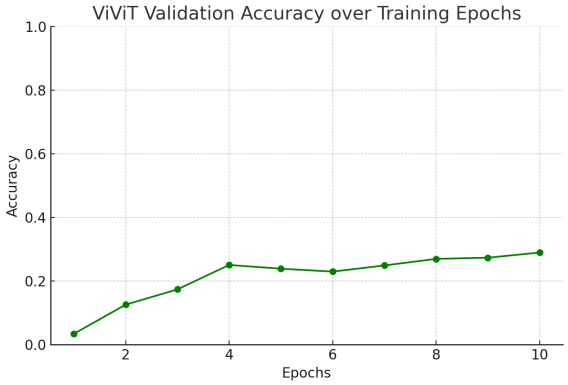


Figure 6: Validation accuracy over time for ViViT

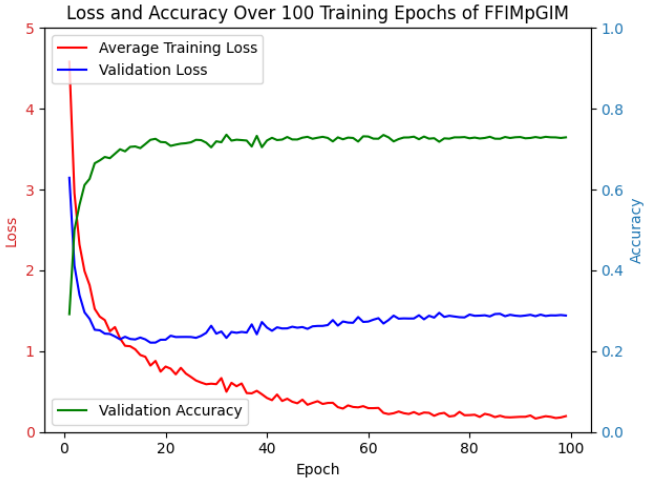


Figure 7: Training loss, validation loss, and validation accuracy over time for FFIMpGIM

The training process of FFIMpGIM is rather interesting considering that validation accuracy converges very quickly and validation loss actually increases while training loss is still decreasing beginning at around Epoch 20. This indi-

cates that starting after Epoch 20, the model has stopped learning and started to overfit the data. This can be a result of multiple factors, be it due to the weight-decay mechanism and learning rate scheduler. The model becomes too specialized on the training data which severly harms future performance. Steps to consider would be stopping training earlier (around Epoch 20), using a new more specific loss function, and further tuning hyperparameters.

5.6. Live Testing

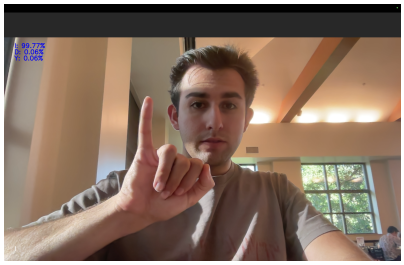


Figure 8: Live sign language recognition of the character "I"

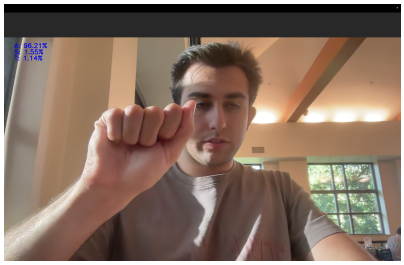


Figure 9: Live sign language recognition of the character "A"

When beginning live testing, as demonstrated in figures 8 and 9, we noticed quickly that although our frame landmark processing and translation logic worked quickly and correctly for the most part, as seen in figure the signs for A, M, N, and S look overwhelmingly similar. Our models could not distinguish between these four letters when signed traditionally according to the left column of Figure 10 whatsoever, always predicting A with near 100% accuracy. We quickly realized we had trained the model on less conventional finger spelling signs which are on the right column of Figure 9, which can be made out by the MediaPipe landmark diagrams.

Thus, to find a solution, we created our own dataset from scratch that had traditional signs in order to distinguish between more conventional signs for A, M, N, and S. This dataset, however, had far too little images (since we were faced with a time crunch) to train a proper model on, with

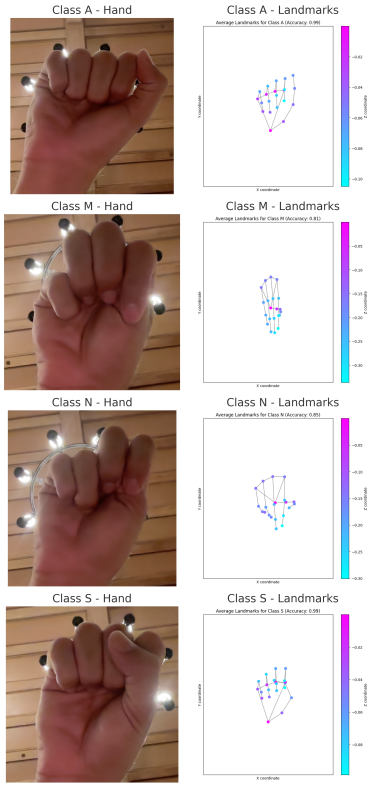


Figure 10: Traditional signing versus finger spelling

only 10 per class.

In order to deal with this all, we decided that instead of retraining our original models, we could generate a specific $AMNS$ character classifier that would only be invoked when the original MediaPipe model predicted A. Considering that the MediaPipe models are extremely efficient and quick, this does not cost us any extra time in live interpretation. To generate more training data for this classifier, we wrote an augmentation algorithm that performed a 32-fold augmentation: all possible combinations (presence/absence of) of a rotation by a random angle from 21 degrees to 79 degrees either clockwise or counterclockwise, a random scaling by a factor of 0.85 to 1.15, a random translation by -0.1 to 0.1 in both the x and y directions (as a reminder, MediaPipe operates with values from 0 to 1 that are normalized to the size of the image with $(0, 0)$ being the top left corner and $(1, 1)$ being the top right), and a left-right flip.

When trained without augmented data, the classifier was able to identify between A, M, N, and S with 53.12% accuracy. When trained with augmented data, it performed at a 72.26% accuracy (both training sessions were done with the same grid search and 5-fold cross validation to

yield, surprisingly, the same hyperparameters. Lastly, we trained a pure image-based classifier to deal with the AMNS problem, which yielded a staggering 98.01% for this minitask.

6. Conclusions

Throughout the course of this project, we arrived at a number of interesting conclusions regarding computer vision for sign language interpretation.

- Models with raw video inputs instead of hand coordinates perform too slow for real time video applications
- Avoiding MediaPipe preprocessing may create higher classification accuracy, but tensors storing videos take up enormous space, which makes training difficult
- MediaPipe creates the best results when analyzing longer sequences of videos
- Classifiers can capture hands more accurately in different lighting and settings with data augmentation that performs random scaling, rotation, and manipulation to training data
- Custom loss functions can provide extra punishment to correct for commonly mistaken or confused classes
- Classifiers on images or short videos can be applied to longer video streams with sliding windows over frames
- Thresholds on logits and predicted probabilities can weed out inputs that don't fall into any of the classes
- Computer vision models can be readily appended with word models for fluent translations

Despite our MediaPipe preprocessed model performing slightly worse on classifying ASL letters, the video analog performed much better on classifying ASL vocabulary. This is likely because our non MediaPipe models (which don't have sets of coordinates as inputs) are observing too much background noise. The backgrounds for the hands in these videos likely add variance which is compounded over multiple frames, which explains the lower performance.

Our Vision Transformers ViT model outperformed all baselines for classifying images. ViT was our best performing model for identifying ASL letters in images. This is likely attributable to the heavy data augmentation on our dataset and more subdivision of the image for a longer sequence into our transformers model.

Our FFIMpGIM model also outperformed baseline models on video gesture classifications. FFIMpGIM was our best performing model for classifying our extended 250-word video vocabulary in ASL. Our model improves upon the baseline models by expanding the embedding layer to capture both static and dynamic aspects of the hands in our training videos.

6.1. Looking Forward

There are several areas for improvements building off our base model for sign language interpretation. Firstly, a custom trained word model able to parse the outputs of our classifiers can offer a drastic improvement with live processing speed. An unsupervised technique can be implemented where a model learns to reconstruct an original sentence from a corrupted version according to our model's outputs. Secondly, further model architectures can be explored for raw video inputs. For example, what if we swap the order of our two attention mechanisms from our ViViT models and our temporal self-attention block fed into our spatial self-attention block? Another idea to explore is synthesizing models. Some of our models struggle between a certain select classes. What if a separate more complex model was trained specifically only on these often confused classes. The resulting predictions from these two models would be synthesized for an even greater classification accuracy. Ultimately, there is ample room without heavy hands for creative explorations on further augmentation to these computer vision models. Applications for this realm of computer vision lends a hand to those who rely on ASL communication and will hands-down leave a significant and hands-on impact on society.

References

[1] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid. Vivit: A video vision transformer. *arXiv preprint arXiv:2103.15691*, November 2021.

[2] K. Bantupalli and Y. Xie. American sign language recognition using deep learning and computer vision. *IEEE International Conference on Big Data (Big Data)*, pages 4896–4899, 2018. Downloaded from IEEE Xplore.

[3] M. Boháček and M. Hrúz. Sign pose-based transformer for word-level sign language recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, June 2021.

[6] Google. MediaPipe: A framework for building multimodal applied machine learning pipelines, 2019.

[7] hoyso48. 1st place solution - 1dcnn combined with transformer. https://www.kaggle.com/competitions/asl-signs/discussion/406684, 2023. Accessed: 2024-05-17.

[8] D. Li, C. R. Opazo, X. Yu, and H. Li. Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison. *CoRR*, abs/1910.11006, 2019.

[9] C. Luna-Jiménez, M. Gil-Martín, R. Kleinlein, R. San-Segundo, and F. Fernández-Martínez. Interpreting sign language recognition using transformers and mediapipe landmarks. In *Proceedings of the 25th International Conference on Multimodal Interaction*, ICMI '23, pages 373–377, New York, NY, USA, October 2023. Association for Computing Machinery. Published: 09 October 2023.

[10] D. Mali, N. Limkar, and S. Mali. Indian sign language recognition using svm classifier. In *Proceedings of International Conference on Communication and Information Processing (ICCIP) 2019*, May 2019. Available at SSRN: https://ssrn.com/abstract=3421567 or http://dx.doi.org/10.2139/ssrn.3421567.

[11] A. Nagaraj. Asl alphabet. Kaggle dataset, 2018.

[12] A. Paszke, S. Gross, F. Massa, et al. PyTorch: An open source machine learning framework, 2019.

[13] S. K. Paul, M. A. A. Walid, R. R. Paul, M. J. Uddin, M. S. Rana, M. K. Devnath, I. R. Dipu, and M. M. Haque. An adam based cnn and lstm approach for sign language recognition in real time for deaf people. *Bulletin of Electrical Engineering and Informatics*, 13(1):499–509, 2024. Available at ResearchGate.

[14] J. J. Raval and R. Gajjar. Real-time sign language recognition using computer vision. In *2021 3rd International Conference on Signal Processing and Communication (ICPSC)*. IEEE, 2021.

[15] C. Szegedy et al. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[16] M. Wijkhuizen. Gislr tf data processing & transformer training. https://www.kaggle.com/competitions/asl-signs/notebooks?competitionId=406684&searchQuery=Mark+Wijkhuizen, 2023. Accessed: 2024-06-05.

[17] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.