American Sign Language Recognition and Analysis Using Deep Learning

A Project Report

submitted in partial fulfillment of the requirements
for the award of the degree of

Master of Technology

in

Computer Science

by

Saurabh Kumar Soni
(CS2426)

under the supervision of

Prof. Ujjwal Bhattacharya

Computer Vision and Pattern Recognition Unit

Indian Statistical Institute

Kolkata – 700108, India

June, 2026

CERTIFICATE

This is to certify that the dissertation entitled **"American Sign Language Recognition and Analysis Using Deep Learning"** submitted by **Saurabh Kumar Soni** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science (with specialization in Data Science)** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Prof. Ujjwal Bhattacharya
Computer Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata – 700108, India

ACKNOWLEDGEMENT

I am deeply grateful to my advisor, *Prof. Ujjwal Bhattacharya*, Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, Kolkata. His sharp insights, patient mentorship, and encouragement were essential to the completion of this work.

I sincerely thank all faculty members and researchers at the Indian Statistical Institute for fostering a vibrant academic environment and providing the resources and freedom necessary for meaningful exploration.

Finally, I would like to express my heartfelt thanks to my family, friends and batchmates for their unwavering support and encouragement throughout this journey. Their patience, understanding and constant motivation has been a source of strength for me. This achievement would not have been possible without their presence and belief in me at every step.

Saurabh Kumar Soni
CS2426
Indian Statistical Institute
Kolkata – 700108, India

June, 2026

ABSTARCT

In this work I build a system that recognizes isolated American Sign Language (ASL) words, and I use it to ask one fairly direct question: when training data is scarce, is it better to look at the video pixels or at the geometry of the signer's body? To find out, I train two very different models on exactly the same clips. The first is appearance-based. Every frame is run through standard preprocessing and a ResNet50 backbone pre-trained on ImageNet, which turns it into a 2048-dimensional feature vector, and a Bidirectional LSTM then reads that sequence over time. The second model never sees a pixel. It works only on MediaPipe keypoints, the tracked coordinates of the body and the two hands, and feeds them to a Transformer encoder. So both models have to learn the same two things, the shape of the hands in each frame and the way those shapes move across frames, and both are trained, validated and tested under one identical protocol. What I care about throughout is a recognizer that is accurate but still light enough to be useful in practice, so it could eventually make communication a little easier between people who sign and people who do not.

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Full Form
SLR	Sign Language Recognition
ASL	American Sign Language
BiLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
CCE	Categorical Cross Entropy
WLASL	Word-Level American Sign Language
GCN	Graph Convolutional Network
GRU	Gated Recurrent Unit
TGCN	Temporal Graph Convolutional Network
ReLU	Rectified Linear Unit
GELU	Gaussian Error Linear Unit

Chapter 1

Introduction

For most of us language means speaking or writing. For Deaf and hard-of-hearing communities it usually means signing. Sign language [2] uses no sound at all; it is a complete, natural language built out of what you can see and move. A single sign mixes manual features such as the shape of the hand, where it is, and how it travels, with non-manual ones like facial expression and body posture, and all of these carry meaning together.

These languages are expressive and complete, and yet a wall still stands between people who sign and people who don't. The wall is there in places like a clinic, a classroom or a bank and even a government office. It quietly keeps the Deaf community from getting services that the rest of us do not really think about.

Automatic Sign Language Recognition is one way to break down that wall. That is why it has become such a topic in computer vision and deep learning.

Teaching a machine to read signs is really hard. It is harder, than teaching a machine to recognize actions. A sign exists in space and time at the time. It also changes fast. If you bend a finger a little or move your hand a centimeters the word can become a completely different word.

Work on Automatic Sign Language Recognition usually splits into two kinds:

- **Continuous SLR:** This is where the model tries to read sentences or long conversations.
- **Word-Level SLR:** This focuses on video clips or specific gestures to identify single words. These are also called "glosses" in linguistics.[5]

This dissertation uses learning to read single isolated words, [21] in American Sign Language.

1.1 Motivation

What drew me to this problem is simple. Every day millions of Deaf and hard-of-hearing people face the challenge. Imagine walking into a doctors office, bank or school and finding that nobody understands your language. For people this is a normal part of life. It keeps a community from accessing services that should be easy to get. Technology to translate signs already exists. It has some drawback:

- Most current models use video files that need powerful and expensive computers to run.
- This means that ordinary people can not use technology to translate signs on a smartphone or tablet when they are outside.

Technology to translate signs was my motivation. I wanted to create a system that's fast and light enough to run on a regular phone or tablet. By using tracked body coordinates of video files this becomes possible for technology to translate signs [15]. It is what could turn technology to translate signs from a lab demonstration into something that can fit in your pocket and help people, in real-life situations with technology to translate signs.

1.2 Problem Statement

Deep learning has made progress in vision. Moving a sign recognizer from a benchmark to daily use still has big problems:

- **Large size Videos slow down computers:** Most models rely on high-resolution videos or huge 3D video networks to read signs. They are pretty accurate. They use a lot of memory and processing power. This makes them hard to run on a smartphone or tablet because they are too big and use many resources.

  Sign recognizer models need to be smaller and more efficient.

- **Background noise and environment shifts:** Models that use video get distracted by things that are not the sign. If the background changes or the lights dim or the signer wears clothes or stands further back the models accuracy drops a lot. The model needs to focus on the hand shapes and geometry. Without that the network learns information and makes mistakes.

- **Not enough video examples per word:** Sign language datasets cover words but have few clips for each word. For example a dataset might have hundreds of words. Only a few short video clips per word to train the model. This lack of data makes it easy for deep neural networks to make mistakes.Sign language models need data to work well.

- **Missing the flow of time is a challenge:** Sign language is about movement and gestures are linked to what happens before and after them. A single picture by itself doesn't mean much. The model needs to track the hand movement over time and understand the context. If a model can't do that it will guess the word. Sign recognizers still have a way to go to be useful in life. They need to understand the flow of time to work well.

  Sign language models are not yet good enough.

1.3 Research Objectives

My goal is to create and study deep learning methods that make word-level American Sign Language recognition more accurate. I want to do this without making the models too big. Here are my objectives:

- **Look into systems that use coordinates:** I will see how well systems work that use the location of joints in the body like the hands and face. I will use tools like MediaPipe to find these locations. This way I can use image data.
- **Make the models work better with data:** I will create a system that helps the models understand the space around the person signing. This system will be based on the size and shape of the body. It will help the models focus on the shapes of the hands. I will also try ways to change the data like rotating the joints to help the models learn better.
- **Evaluate Hybrid Spatial-Temporal Transfer Networks:** I will test models that use a combination of pictures and sequences, like ResNet50 and Xception with special models called Bidirectional LSTMs. These models will help the computer understand what is happening in each frame of the video. They will also help the computer understand the order of the signs.
- **Analyze Attention Mechanisms vs. Recurrent Layers:** I will compare two kinds of models one that uses attention and one that uses layers called Bidirectional LSTMs. I will look at how parts the models have how complicated they are, how fast they can make predictions and how well they work with different amounts of data.

1.4 Thesis Contributions

The main things this work contributes are the following:

- The original paper includes a **ResNet50 + BiLSTM** model which was trained and evaluated on a 10-gloss subset of the WLASL dataset[14].
- I re-implemented this ResNet50 + BiLSTM pipeline and evaluated it on a curated 20-gloss subset of WLASL (only glosses with at least 10 videos) to test how well such appearance-based models recognize signs from short video clips. [1]
- Implemented a **purely skeleton coordinate-based (pose-based) Transformer Encoder model** designed for isolated Sign Language Recognition. [8]
- It is an ultra-lightweight, attention-driven network built to bypass the immense processing weights of standard 3D CNNs, making it ideal for edge processing on regular smartphones.
- Instead of reading heavy, raw video frames like a 3D CNN, this model processes a pre-extracted timeline of 2D/3D skeletal numbers (saved as `.npy` files) representing the signer's body posture and finger movements.
- My tests proved that this pose-based attention model uses much less processing power and generalizes much better on small amounts of data than heavy video-based models.

1.5 Thesis Organization

The rest of the dissertation is laid out like this:

- **Chapter 2:** This chapter covers the background that is really important. It talks about deep transfer learning and bidirectional LSTMs. It also talks about coordinate keypoint tracking and the foundations of self-attention mechanisms.
- **Chapter 3:** This chapter looks at work that has been done on sign language recognition. It compares the approaches that focus on how things look and the models that are based on the pose of the body.
- **Chapter 4:** This chapter explains the methodology, in detail. It talks about how the signing space's normalized and how the coordinates are changed to make things better. It also talks about the modified Transformer architecture.
- **Chapter 5:** This chapter outlines how the training was done. It talks about the steps that were taken to prepare the videos and the software that was used. It also talks about the settings that were used for all the models.

- **Chapter 6:** This chapter presents the results of the experiments. It shows the results of the tests that were done on groups of words. It also talks about what happens when some things are not normalized. It looks at how well the system can handle a lot of data and how complicated it is to run.

- **Chapter 7:** It shows the conclusion of the dissertation. It summarizes the points and suggests what people can do for future research. This is based on the data constraints and the architectural limitations of sign language recognition and the sign language recognition systems

Chapter 2

Preliminaries

Reading a sign from a video is actually quite simple. It comes down to two things.

First what does the hand look like in a picture?

Second how does it move from one picture to the next?

This chapter explains the deep learning ideas I use to figure out both parts. I used two approaches: (i) One approach uses a -trained model called ResNet50. It feeds into a type of model called a Bidirectional LSTM. This works directly with the video. (ii) The other approach uses a Transformer model. It looks at the skeleton coordinates of the hand. It uses something called self-attention to understand the movement. The hand and video are what I focus on. The deep learning ideas help me understand the hand in the video.The two approaches help me answer my questions, about the hand.

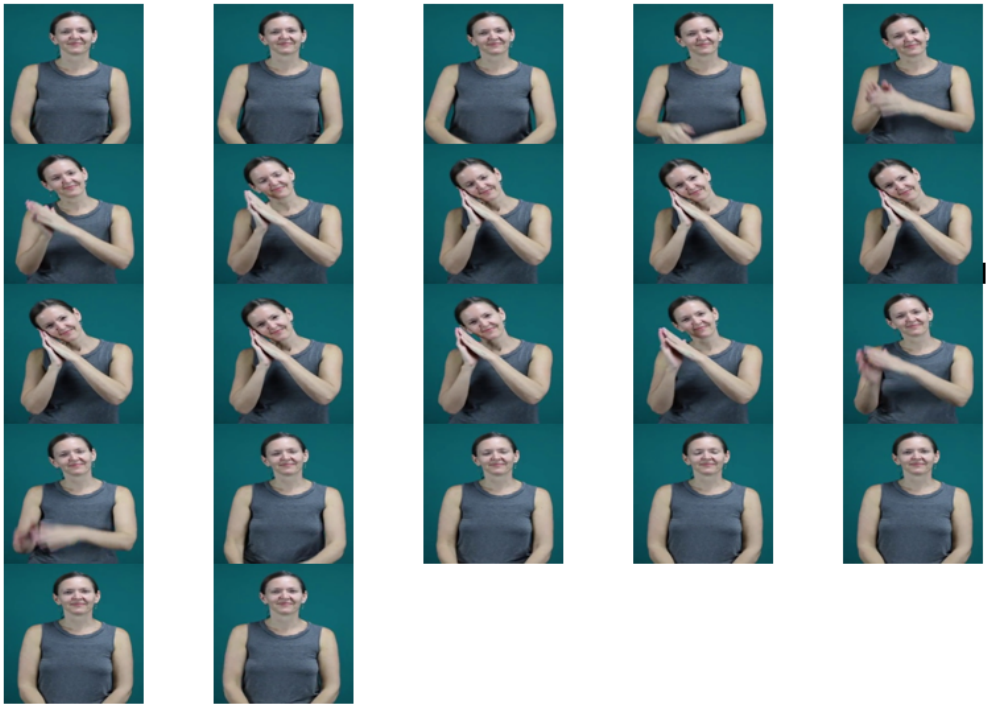

Figure 2.1: 22 Frames of Sign Word "Bed".

2.1 Deep Transfer Learning using ResNet50

Training a deep network from scratch is hungry for data, on the order of thousands of clips per class, and that is exactly what sign datasets like WLASL [3] cannot offer. The usual way around this is **transfer learning**.

So instead of starting from random weights, I take **ResNet50** mujtaba2020, a 50-layer convolutional network that has already been trained on millions of ImageNet photos. Its trademark is the residual block.

Very deep networks tend to choke on the *vanishing gradient problem*: as the error signal travels back through layer after layer it shrinks towards zero and learning stalls. ResNet50's answer is a shortcut that lets a block's input skip straight to its output,

$$Y = f(x) + x \tag{2.1}$$

Those skip connections keep training stable even when the network is very deep. To begin with I do not retrain ResNet50 at all. I freeze it and treat it as a frame-by-frame feature extractor, squeezing each raw frame into one compact vector, which saves a great deal of training time.

2.2 Recurrent Neural Networks and Bidirectional LSTMs

Once ResNet50 has turned each frame into a vector, something has to read the whole sequence and make sense of the gesture's timeline. A plain Recurrent Neural Network (RNN) tries to do this by carrying a hidden memory state from one frame to the next, but in longer clips it tends to forget the early frames.

Long Short-Term Memory (LSTM)

An LSTM [10] gets around this forgetting with a dedicated **cell state** $c(t)$ and three gates that decide how information moves through it:

- **Forget Gate** $f(t)$: Decides what piece of past information to throw away.
- **Input Gate** $i(t)$: Decides which new features from the current frame to store in the cell state.
- **Output Gate** $o(t)$: Decides what the next hidden state output $h(t)$ should be.

Bidirectional Processing

In signing, what a gesture means depends not only on what came before it but on what comes right after. A plain LSTM only reads the clip forwards, start to finish. A **Bidirectional LSTM (BiLSTM)** [11] runs two hidden layers over the frames at the same time:

- A **forward layer** that reads the gesture sequence from frame 1 to T.
- A **backward layer** that reads the gesture sequence backward from frame T to 1.

At every time step the two outputs are joined together, so at any moment the model sees both the past and the future context of the hand movement, not just one side of it.

2.3 Coordinate Keypoint Tracking

There is a different route that skips heavy pixel video altogether: track the signer's skeleton [24] instead. A tool like MediaPipe looks at the frames and reads off 2D landmark coordinates (x, y) for the important joints. For isolated SLR the joints I care about are:

- **Face/Head Landmarks:** These help me track facial expressions and give me a reference point.
- **Body/Pose Landmarks:** I use these to map the shoulders, elbows and wrists which shows me how the arms are moving.
- **Hand Landmarks:** Each hand has 21 points that are tracked including fingers and knuckles so I can see the small details.

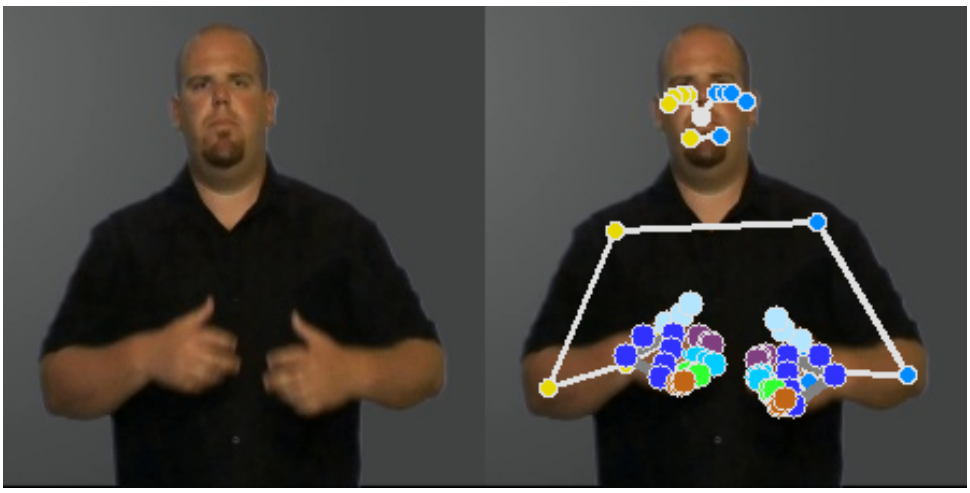

Figure 2.2: Video frame after applying MediaPipe keypoints.

The figure shows an example: the original picture on the left and the detected landmarks on the right. Each dot is a landmark, which is a named joint or a point on the face stored as an (x, y) coordinate with a depth z as well, which is not visible in 2D. The white lines connect these points like from the shoulder to the elbow to the wrist and across the shoulders and even the small bones in each finger. You can see three groups in the picture: some points on the the **face** (nose, eyes, corners of the mouth), the **upper-body pose** like a white trapezoid over the shoulders, elbows and wrists, and two dense clouds of about 21 points on the **hands** These groups are what the pipeline simplifies into one vector that's 225-dimensional: the pose, the left hand and the right hand.

2.4 Transformer Models and Self-Attention Mechanisms

A BiLSTM walks through the frames one after another in a chain. The **Transformer** does something different: through **self-attention** it looks at every frame at once, so it can connect a moment near the start of the clip to a moment near the end without having to pass information hop by hop.

Scaled Dot-Product Attention

The heart of a Transformer is the attention layer. Each frame's vector is turned into three vectors, a **Query** (Q), a **Key** (K) and a **Value** (V). To get the attention weights the model takes the dot product of the Queries and Keys, scales it down by the dimension d_k, runs it through a Softmax, and multiplies the result by the Values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.2}$$

In other words the formula tells the model how much frame A should attend to frame B, however far apart the two are on the timeline.

Positional Encoding

Since attention sees all the frames together, it has no built-in sense of order. The fix is to add a **positional encoding**, fixed or learned, straight onto the input features. This stamps every pose vector with its place in time, so the model can still tell how the sign unfolds from start to finish.

Multi-Head Attention and Decoder Modifications

Running attention several times in parallel, which is what **multi-head attention** [13] means, lets the network follow more than one thing at a time, say hand-shape changes on one head and the path of the arm on another.

Chapter 3

Related Work

Past attempts at isolated SLR tend to sit in one of two camps. There are **appearance-based** methods that work on the raw video pixels, and **pose-based** methods that work on tracked joint coordinates. Each camp has its own trade-offs in accuracy, training speed and how much compute it demands.

3.1 Visual Appearance-Based Frameworks

Appearance-based methods stay with the raw video. They treat SLR as a heavy video-classification problem and make the model learn everything straight from the pixels.

2D CNN + RNN Hybrids

The classic recipe pairs a convolutional network with a recurrent one. A strong pre-trained image model, ResNet50 [12] or Xception [1], is used as a feature extractor: it runs over the clip and turns each frame into a dense 2048-dimensional vector.

Those per-frame features then go into a recurrent model[18], an LSTM or a BiLSTM, which traces the movement over time. This captures static hand shapes and textures fairly well, but it overfits easily and has trouble with long-range patterns, partly because it is stuck reading the clip one frame at a time.

3.2 Skeletal Pose-Based Models

Appearance models burn a lot of compute on things that have nothing to do with the sign, the room behind the signer, their clothes, the lighting. That waste is what pushed a second line of work towards **skeletal pose data** instead. [8]

Rather than feeding in pixels, these systems use trackers such as MediaPipe [6] Holistic or Apple's Vision API to reduce the signer to a lightweight stick figure. The tracker scans each frame and pulls out 2D (x, y) landmarks for the face, the torso and all 21 joints on

each hand. A multi-megabyte video collapses into a small matrix of numbers, which forces the model to attend to the geometry of the sign and nothing else.

There have been a few different ways of feeding these coordinates to a model:

- **Recurrent Pose Models (Pose-GRU / Pose-LSTM):** These try to read the coordinate streams step-by-step. While they are incredibly lightweight, their accuracy is historically low because raw coordinate numbers can easily get messy if the signer shifts positions or stands slightly tilted.
- **Graph Convolutional Networks (GCNs):** Since the human skeleton is naturally a connected graph, models like *Pose-TGCN* treat [3] the joints as nodes and bones as edges, passing message signals along the skeletal paths to learn the dynamics of the gesture. Tunga et al. combined this setup with text-based models (GCN-BERT) [4] to capture spatial and temporal features separately, which boosted prediction accuracy on the WLASL dataset.

3.3 The Rise of Pose-Based Transformers

For a long time pose-based models [4] trailed the appearance-based ones on accuracy. The Transformer changed that. Where an LSTM reads coordinates one step at a time, a Transformer takes in all the frames together and can connect a hand shape at the beginning of a clip to a movement at the end in a single pass.

Work such as *SPOTER* showed that with a sensible spatial normalization, one that centres the coordinates around the signing space [23], a purely coordinate-based Transformer can match heavy 3D CNNs [7]. Drop the cost of processing raw video, trim the decoder down to predicting a single word, and you are left with a fast, pocket-sized recognizer that could actually be used in the real world.

Chapter 4

Proposed Methodology

This chapter sets out how the recognizer is actually put together. To compare two honestly different ways of attacking ASL, I built two model streams and pointed both at the same job: predicting **20 sign words (glosses)**. Every clip is trimmed to a fixed **22 frames** so the inputs stay uniform. The two architectures are:

- A hybrid **ResNet50 + Bidirectional LSTM (BiLSTM)** model that uses a two-phase transfer learning approach to extract features frame-by-frame.
- A coordinate-based **ASL Transformer Encoder** that throws away pixels completely and only tracks skeletal joint numbers.

4.1 Data Representations and Formats

The two models look at the very same videos in completely different ways:

ResNet50 + BiLSTM Input (Visual Feature Vectors)

For the hybrid model I never push raw video into the LSTM. Each frame goes through a pre-trained ResNet50 first. Think of ResNet50 as a trained eye: it takes a $224 \times 224 \times 3$ frame and hands back a flat 2048-dimensional vector that summarises what is in it,

$$\vec{x}_t \in \mathbb{R}^{2048} \tag{4.1}$$

A whole sign video therefore becomes a list of 22 such vectors. So that the ImageNet-trained backbone gets the input it expects, every frame first goes through the standard channel-wise mean subtraction and preprocessing. The vectors are written out into three CSV files, `train.csv`, `val.csv` and `test.csv`, one 2048-number row per frame.

Transformer Input (Skeletal [22] Tracking Numbers)

The Transformer throws the images away. It uses a tracker like MediaPipe to reduce the signer to a list of numbers, saving each frame as a **225-dimensional vector** that records the (x, y, z) coordinates of the upper body:

- **Body Pose (Indices 0–98):** 33 landmarks mapping the shoulders, elbows, and wrists.
- **Left Hand (Indices 99–161):** 21 detailed landmarks tracking every finger joint and knuckle.
- **Right Hand (Indices 162–224):** 21 detailed landmarks for the other hand.

Every landmark comes with an (x, y, z) coordinate, so a clip for this model is nothing more than a $(22, 225)$ matrix: 22 frames, each a row of skeleton numbers.

4.2 Comprehensive System Architecture

Hybrid ResNet50 + BiLSTM Network

This design keeps the spatial part and the temporal part separate. The 22 frames first go through the ResNet50 backbone. It was pre-trained on ImageNet dataset. A TimeDistributed wrapper sends each frame through the CNN's global pooling layer. The layer has 2048 dimensions. The cached 2048-dimensional features are reduced to 128 with PCA whitening, and the resulting sequence then goes through a LayerNormalization step. After that it reaches the recurrent part.

The recurrent part has two stacked Bidirectional LSTM layers:

- **BiLSTM Layer 1:** It contains 256 hidden units. This results in 512 combined forward and backward output units. It returns the sequence layout because return sequences is set to True.
- **BiLSTM Layer 2:** It contains 128 hidden units, which gives 256 output units. When you set return sequences to False it removes the timeline part. Combines all the time information into one vector. This vector summarizes the video block.

That summary vector then goes through a 256-unit dense layer with a ReLU activation and an L2 weight penalty. After a Batch Normalization step and a fairly aggressive Dropout ($p = 0.5$), a last dense layer maps everything onto the 20 output classes.

ASL Transformer Encoder Model

The Transformer keeps neither convolutions nor recurrence; it relies on **self-attention** to see every frame at once. A dense layer first projects the 225 raw coordinates into a hidden size of 128 ($D_{\text{MODEL}} = 128$). Because attention has no sense of order on its own, a **positional encoding** is added on top so each frame carries a time stamp. The sequence then passes through 3 Transformer encoder blocks, each with 4 attention heads, which is what lets the model link a hand shape at the very start of the clip to a pose at the very end. Finally the output of the prepended CLS token is combined with an attention-pooled summary of the sequence and sent to a dense head that predicts the gloss.

4.3 Data Augmentations for Small Datasets

Sign datasets [14] are really limited you only get a clips for each word, which means the system is likely to overfit. To deal with this I relied heavily on data augmentation when training the system.

Skeletal Coordinate Augmentations

For the Transformer, we apply algebraic transformations directly to the (x, y) coordinate numbers:

- **Gaussian Noise:** We add a Gaussian Noise to the joint coordinates, which is, like adding tiny random numbers to simulate a shaky camera.
- **Random Mirroring (X-Flip):** We do Random Mirroring, which means we flip the x-coordinates by multiplying them by -1 and swap the left and right hand data.
- **Coordinate Dropout ($p = 0.5$):** We use Coordinate Dropout, where we randomly remove a landmarks in some frames so the system has to guess the correct sign even if the signers hand is blocked or blurry.
- **Speed Warping ($\pm 20\%$):** We also use Speed Warping, where we make the gesture a bit faster or slower. The system learns to ignore the signing speed of a person and focus on the Sign datasets and the signs themselves. This way the system can learn from the Sign datasets. Become better at recognizing signs.

4.4 Training and Optimization Strategies

Both models have the basic setup for learning and fixing their mistakes across the 20 classes.

Stratified K-Fold Cross-Validation

I want to make sure the models really work well and are not just remembering specific things.To do this the pipeline can do a 5-fold stratified cross-validation. It combines the training and validation data. Splits it into five equal parts each part having the same ratio of classes across the 20 glosses.The data is loaded when it is needed and frames are read from the disk, in small batches which keeps the memory use low and stops the computer from crashing.

Frozen Backbone and Decoupled Training

One thing to note is that ResNet50 is never fine-tuned. It stays the same from start to finish. Is only used to get features from images so the weights it learned from ImageNet never change. The process is divided into two parts. This is only about when the features are made, not about changing anything:

- **Offline feature extraction:** Each of the 22 frames is passed through the ResNet50 base one time and 2048 dimensional vectors for each frame is saved to disk as a CSV file. This step only needs to be done because the ResNet50 base never changes, which keeps the cost of training low.
- **Training the head only:** The saved features are made to be similar and smaller with PCA whitening from 2048 to 128 and the part of the model that looks at the features over time, which is made up of two BiLSTM layers, one with 256 units and one with 128 units and a dense classifier is actually trained. This part uses the Adam optimizer with a learning rate of 0.0001. Stops early if the accuracy on the validation set does not get better for 30 times.

Freezing the backbone is a deliberate choice for a dataset this small. Unfreezing a high-capacity CNN on a handful of clips per class tends to overwrite the useful ImageNet features and overfit almost at once, so the model is simply never given the chance to do that.

Categorical Cross-Entropy Loss and Regularization

Isolated SLR is a multiple-choice problem, where one word is correct. The loss used is Categorical Cross-Entropy. When using labels with $C = 20$ classes the loss is calculated as:

$$\mathcal{L}_{CCE} = -\sum_{c=1}^{C} y_c \cdot \log(\hat{y}_c) \tag{4.2}$$

To keep the weights small and manageable an L2 penalty is added to the loss, on the layers. This penalty helps prevent complexity. The model stays focused on the movement of the sign itself because of this penalty. The L2 penalty value used is ($\eta = 1 \times 10^{-4}$).

Chapter 5

Implementation

This chapter is the hands-on part: how the two models are actually run, trained and validated across **20 sign words (glosses)**. I chose 20 sign words because most of the 1000 words, in the set have a few clips. The sections below explain how we split the data build the MediaPipe tracking system and the exact settings and techniques used for each model.

5.1 Dataset Partitioning and Class Structure

The dataset has video files and matching skeleton-track files for $C = 20$ sign words. To ensure the results are reliable we keep the splits separate:

- **Training Set:** Comprises 70% of the video blocks. The models use this data to learn gesture patterns, hand shapes and body movements.
- **Validation Set:** Comprises 15% of the data, used during training loops to tune learning rate values and apply early stopping regularizations.
- **Testing Set:** Comprises 15% of the held-out sample items, analyzed only at the very end to check real-world generalization performance.

All three splits are made in a stratified way so that no class is over- or under-represented; the proportion of clips per sign is the same in the training, validation and test sets.

5.2 Video Pre-processing and Feature Extraction Pipelines

Since the two models want such different inputs, there are two separate preprocessing pipelines.

Image-Plane Pre-processing (ResNet50)

For the pixel pipeline, OpenCV reads the videos straight from their folders. People sign at different speeds, so to line clips up I sample exactly **22 frames** evenly across each one. For the **ResNet50 + BiLSTM** the frames stay at the usual ImageNet size of 224×224, and rather than a plain divide-by-255 they get the standard ImageNet mean subtraction and channel scaling that the pre-trained layers expect.

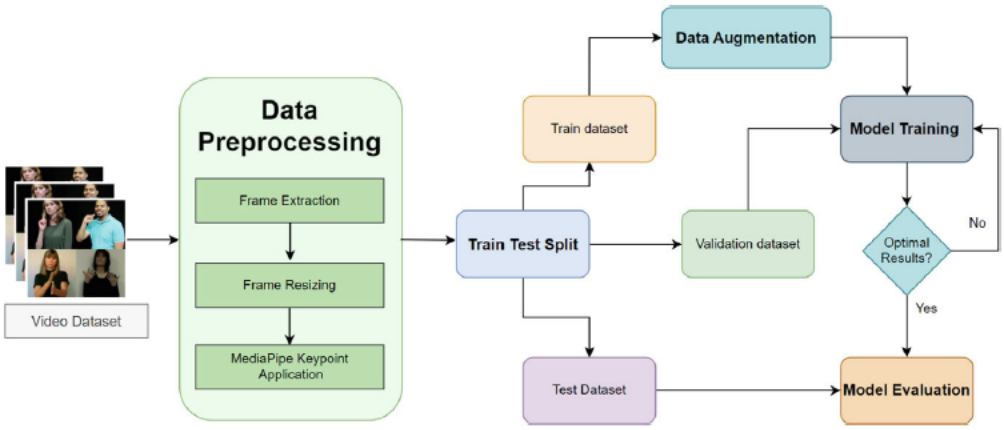

Figure 5.1: Overview of ResNet50 + BiLSTM Model.

ResNet50 Image Preprocessing Steps: ResNet50 is, in effect, an **image-recognition brain** that has already seen millions of ImageNet pictures, so it is very good at picking out shapes, edges, hand silhouettes, textures and posture. When the script hands it a `.jpg` frame, here is what happens:

- **It looks at raw pixels:** ResNet50 receives a grid of 224×224 pixels with 3 color channels (Red, Green, Blue). That is **150,528 raw color numbers** ($224 \times 224 \times 3$) per single frame.
- **It scans for patterns:** The frozen convolutional layers of ResNet50 scan these pixels. They look at the shape of your hand, how your fingers are bent, where your arm is positioned, and your overall posture.
- **It strips away the "picture" and summarizes it:** Because we excluded the topmost layer (the part that guesses "cat" or "dog"), ResNet50 doesn't try to classify the image. Instead, at its very final step (Global Average Pooling), it condenses all 150,528 pixel color numbers into a single list of **2,048 key visual features**.

Those 2,048 numbers are basically a numeric fingerprint of how that one frame looks.
Content of `split.csv` file. A machine cannot keep thousands of high-resolution frames in RAM during training, so ResNet50's summaries are written to a `.csv` file instead. By

that point **the images are gone**; the file is plain text, thousands of lines long, and each line stands for **one frame**. A row looks like this:

Class	Video ID	Feature 1	Feature 2	···	Feature 2048
bed	video_train_04	−0.4125	1.8921	···	0.0451

Table 5.1: Example of a single feature row stored in a split CSV file.

With **22 frames per video**, the CSV holds 22 rows in a row for `video_train_04`, then 22 for `video_train_05`, and so on. At training time the BiLSTM never touches the images; it reads the CSV line by line, pulls 22 rows at a time to rebuild one clip's timeline, and treats those 2,048 numbers as a sequence. By following how the numbers shift from frame 1 to frame 22, it learns the **movement** of the sign.

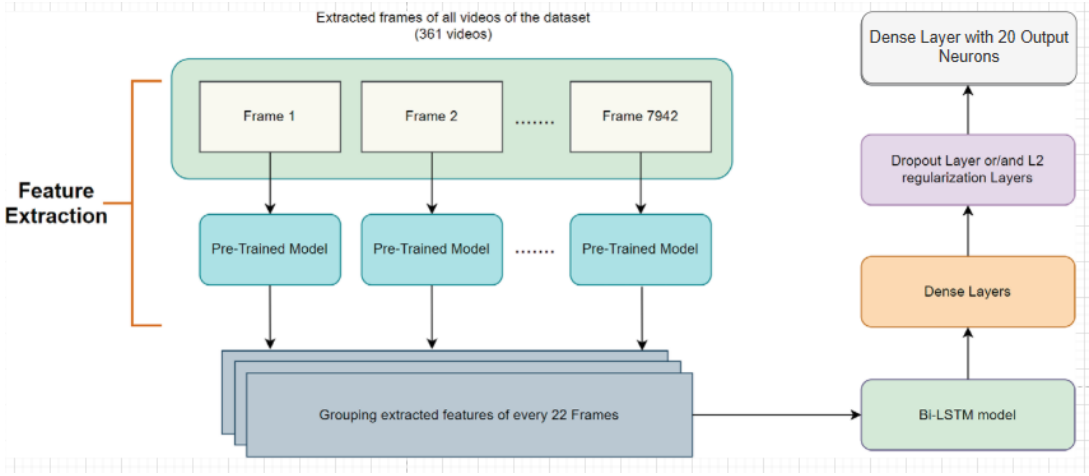

Figure 5.2: ResNet50 + BiLSTM Architecture.

Skeletal Tracking and Keypoint Generation (Transformer)

The Transformer pipeline keeps no image grids at all and works only from numeric landmarks. Each frame is run through the **MediaPipe Tasks API**, with the PoseLandmarker (Lite) and HandLandmarker tasks called in a separate extraction loop.

(i) Pose Landmarker. The Pose Landmarker locates the main points of the body.

- **Keypoints:** Estimates 33 3D landmarks across the human body (e.g. nose, shoulders, hips, knees, wrists) in image coordinates and real-world 3D coordinates.
- **Model Bundles:** Available in Lite, Full, and Heavy versions, allowing you to balance speed, size, and precision. The Lite version is optimized specifically for on-device, real-time tracking.
- **Use Cases:** Posture analysis, gesture categorization, movement tracking, and fitness/rehabilitation applications.

(ii) Hand Landmarker. The Hand Landmarker is built for the finer detail of the hands.

- **Keypoints:** Tracks 21 hand-knuckle coordinates within detected hand regions, generating both 2D and 3D world coordinates.
- **Handedness:** Unlike the Pose solution, it specifically identifies whether a detected hand is left or right.
- **Architecture:** Uses a two-stage approach: it first runs a palm detection model, then applies the hand landmarker model only to the cropped hand regions.
- **Use Cases:** Sign language recognition, contactless interface control, and granular gesture manipulation.

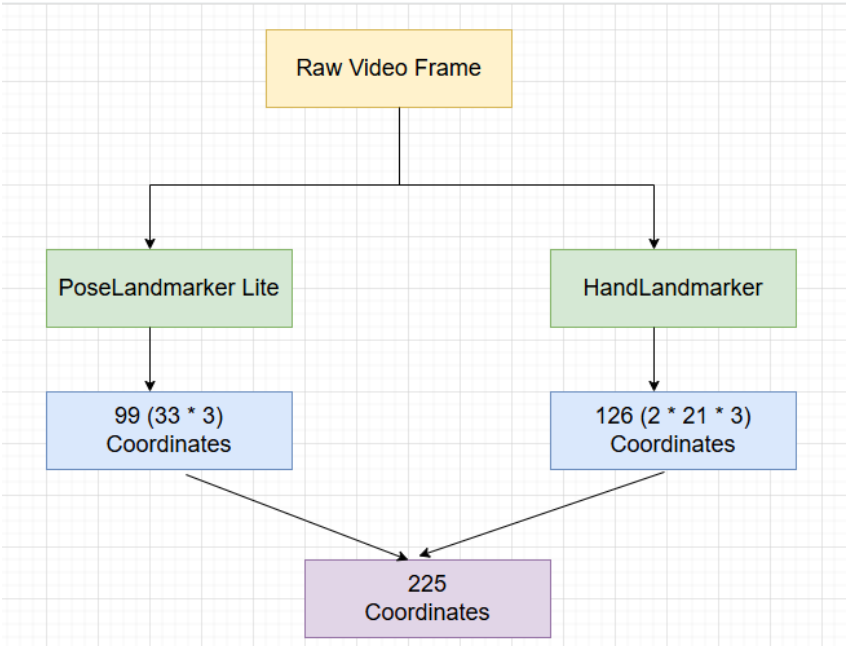

Figure 5.3: MediaPipe Keypoint Extraction.

For every single image frame processed, the Tasks API extracts 3D space tracking properties (x, y, z) across human body nodes:

- **Pose Structural Points (99 values):** This is like a map of 33 points on the body, including the torso shoulders, face and other parts like the eyes, ears and nose as well as the elbows and wrists.
- **Left Hand Matrix (63 values):** This is like a map of the bones and joints in the left hand including the knuckles, finger bones and fingertips. It has 21 points that help figure out the coordinates.
- **Right Hand Matrix (63 values):** This is the same as the hand but for the right hand.

All of this information gets put into a flat **225-dimensional vector**, which is saved as a file `video_id.npy` for each class. Sometimes when there is motion blur, MediaPipe has trouble finding a hand or a limb. That can cause problems, with the extraction of the Tasks API.

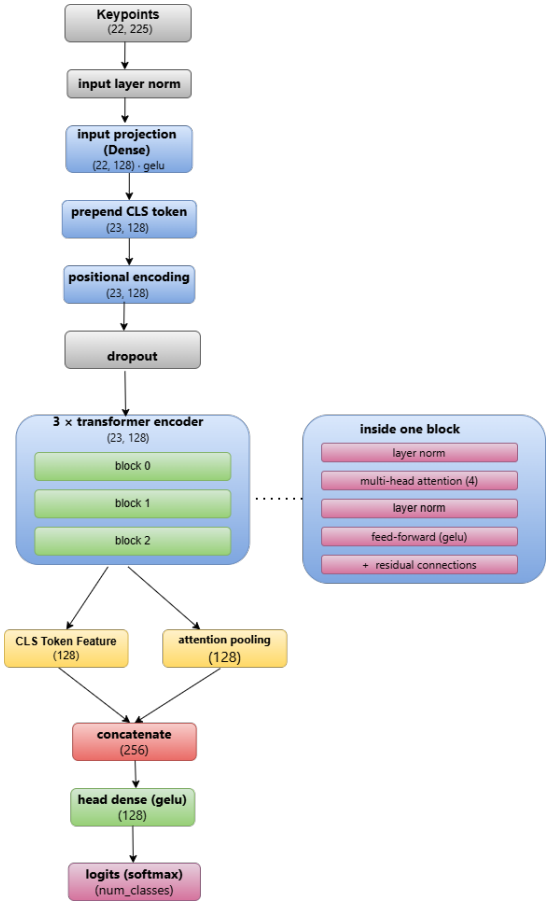

Figure 5.4: Transformer-based Architecture.

This is the ASLTransformer, the coordinate-based network I wrote for reading movement. It is worth walking through how a sequence of skeleton numbers becomes a predicted word.

Layer-by-Layer Architectural Breakdown

1. Data Ingestion & Linear Projection

- **keypoints (InputLayer) → (None, 22, 225):** The ASLTransformer network receives a batch of videos. Each video is made up of 22 frames. Each frame has a lot of coordinate information. 225 Metrics to be exact. This includes 33 points for the body 21 points for the hand and 21 points, for the right hand all with 3 dimensions (x, y, z) coordinates.

- **input layer norm → (None, 22, 225):** The ASLTransformer network then normalizes each frames coordinates. This helps to keep everything on the scale so one joint does not dominate the others just because it has large coordinate values.

- **input projection (Dense) → (None, 22, 128):** A 225-dimensional coordinate space is a lot to handle for self-attention. This fully connected layer linearly projects those 225 joint numbers into a clean, uniform latent embedding dimension ($D_{\text{model}} = 128$) using a GELU activation. This is where the model forms its initial internal representation of each frame's hand shape and body pose.

2. Classification Token & Temporal Calibration

- **prepend CLS token → (None, 23, 128):** A single learnable classification (CLS) token is added to the front of the sequence, growing it from 22 to 23 positions. This token is not derived from any frame—it is a blank, trainable slot that, through the attention layers, learns to aggregate information from all 22 frames into one summary vector used later for classification (the BERT/ViT convention).

- **pos_enc (PositionalEncoding) → (None, 23, 128):** Transformers have no inherent sense of order; without this, the model would treat the frames as an unordered pile. This layer injects fixed sine and cosine waves directly into the vectors, hard-coding chronological time so the attention layers can distinguish a hand moving up from a hand moving down.

3. The Core Transformer Encoder Stack

- **transformer_block_0 to transformer_block_2 → (None, 23, 128):** The architecture stacks 3 identical Transformer Encoder Blocks. Each block contains two engine rooms:

  - **Multi-Head Attention (4 heads):** This lets the network look at the 23-position timeline at the same time. It figures out how important the pose is at one frame compared to every frame. This helps catch long-range patterns in the trajectory.
  - **Feed-Forward Network (FF_DIM = 256):** It applies -linear changes to the attention outputs looking at each position one by one. This is done with GELU activation. Each sub-layer is covered with layer normalization and residual connections which makes the training more stable.

4. Dual Aggregation & Classification Head

- **CLS token feature → (None, 128):** After looking at every frame, across all three blocks, the CLS token now holds a summary of the sign that it learned.

- **attention pooling → (None, 128):** In parallel, an attention-pooling layer computes an average of all 23 positions. This gives a summary that complements the first one. The Dual Aggregation and Classification Head uses these summaries to classify the sign.

- **concatenate → (None, 256):** The CLS summary and the attention-pooled summary are concatenated, giving the classifier two complementary views of the clip.
- **head_dense (Dense) → (None, 128):** A dense hidden block of 128 neurons with GELU activation that adds a final layer of abstraction. It uses an L2 regularization penalty (1e−4), followed by dropout, to prevent overfitting on the 20-class subset.
- **logits (Dense) → (None, num_classes):** The final classification layer maps the 128-dim vector directly to the target classes as raw confidence logits (softmax applied via the loss).

5.3 Hyperparameter Environments and Training Routines

The two models use specialized optimization settings tailored to their distinct internal processing layouts.

ResNet50 + BiLSTM Hyperparameters. This model trains on pre-extracted `.csv` feature files rather than raw frames: every frame is passed once through ResNet50 (image preprocessing → 2048-dimensional feature vector) during a separate extraction step, and those cached features are reused for training, saving substantial execution time. Before the recurrent layers, the 2048-dimensional features are standardized and reduced with PCA whitening (fit on the training frames only) to combat the curse of dimensionality on a small dataset.

```
CSV_DIR        = "/kaggle/input/datasets/sauravkumar/mydata1"
LABEL_MAP_PATH = "/kaggle/input/datasets/sauravkumar/mydata1/label_map.json"
SEQ_LEN        = 22       # 22 frames per video
N_FEATURES     = 2048     # ResNet50 global-average-pooled feature vector
N_PCA          = 128      # PCA-whitened components fed to the BiLSTM
NUM_CLASSES    = 20       # 20-gloss dataset size
BATCH_SIZE     = 8
MAX_EPOCHS     = 250      # early-stopped on validation accuracy (patience 30)
LEARNING_RATE  = 0.0001   # Adam optimizer
L2_REG         = 0.001    # kernel L2 penalty
DROPOUT        = 0.5      # applied after the projection and the BiLSTM
```

The recurrent head reads the PCA-whitened 128-dimensional feature sequence and is built from two stacked Bidirectional LSTM layers (256 and 128 units), followed by a dense block and a softmax classifier. It is kept heavily regularized, with dropout and an L2 penalty, because there are only about 13 clips per class and the model would otherwise overfit very quickly.

ASL Transformer Encoder Hyperparameters. Because coordinate arrays are lightweight, the Transformer operates directly on the keypoint sequence of length $T = 22$ frames. The model is trained for up to 110 epochs with a batch size of 8 using the AdamW optimizer with a warmup–cosine learning-rate schedule, and is early-stopped on validation accuracy.

```
KEYPOINT_DIR   = "/kaggle/input/datasets/sauravkumar/mydata2/keypoints"
LABEL_MAP_PATH = "/kaggle/input/datasets/sauravkumar/mydata2/label_map.json"
TRAINING_MODE  = "standard"   # "standard" or "kfold"
K_FOLDS        = 5
SEQ_LEN        = 22           # 22 frames per gloss
FEATURE_DIM    = 225          # 33 pose + 21 + 21 hand landmarks, x 3 (x, y, z)
D_MODEL        = 128
NUM_HEADS      = 4
FF_DIM         = 256
NUM_LAYERS     = 3            # 3 transformer encoder blocks
DROPOUT_RATE   = 0.20
BATCH_SIZE     = 8
EPOCHS         = 110
PATIENCE       = 30
PEAK_LR        = 3e-4
WEIGHT_DECAY   = 3e-4
LABEL_SMOOTH   = 0.10
```

The encoder prepends a learnable CLS token to the 22-frame sequence (length 23 with the token), applies sinusoidal positional encoding, passes the sequence through 3 encoder blocks of 4-head self-attention, and forms the final representation by concatenating the CLS token's output with an attention-pooled summary before the classification head.

5.4 Algorithmic Evaluation and Cross-Validation Framework

To really see how well our system works and to avoid problems with the data we have two ways to train our system: the way and the cross-validation way. We pick one of these ways based on what we need to do with our experiment.

Standard Single-Split Training Mode. The standard pipeline separates the dataset along predefined partition margins, keeping the training, validation, and testing set strictly isolated throughout the epoch loop. This way we can see how well our system is doing away and we can track how well it is learning. We use a kind of model called ResNet50 + BiLSTM. By splitting our data one time, we can train our system faster and we can see how well it is doing at each step.It minimizes repetitive training steps while allowing clean, per epoch loss tracking.

Stratified K-Fold Cross-Validation Mode. When I am writing my dissertation and I need to make sure my statistics are correct I use a 5-Fold Stratified Cross-Validation framework. This mode puts all my training and validation data together in one place so I do not get metrics that only work for one set of Cross-Validation data. The Cross-Validation pipeline works as follows:

- **Stratified Split Allocation:** I divide my pool of Cross-Validation data into 5 separate parts using a special function that makes sure the class distribution of my 20 target glosses is the same in every part of the Cross-Validation.
- **Iterative Loop Optimization:** I build my network 5 times from scratch. Each time I do this I pick one part of the Cross-Validation to be my validation set. I use the other 4 parts of the Cross-Validation for training my network.
- **Statistical Output Aggregation:** After I finish all 5 loops of the Cross-Validation I take the score from each loop. I average them. I also look at how much the scores vary in the Cross-Validation, which gives me an idea of how my system's working with the Cross-Validation. I tested with a separate test set to see how well my model really works with the Cross-Validation.
- **Canonical Weight Recovery:** I save the parameters from the best loop of the Cross-Validation, as my best checkpoint, which is called bestmodel.keras so I can use it to evaluate my system against the test set that I have not seen before with the Cross-Validation.

Chapter 6

Experimental Results

This chapter tells about what the two methods did on the isolated American Sign Language task. Both methods were tested on the subset so it is a fair comparison. I looked at the American Sign Language task using the metrics for the American Sign Language task: Top-1 accuracy, Top-3 accuracy and how stable the numbers for the American Sign Language task are, across the cross-validation folds.

6.1 Dataset Compilation and Extraction Pipeline

WLASL covers close to 1000 glosses, but I kept only the 20 that have more than 10 videos clips each, which gives the models a fighting chance of learning something. Table 6.1 gives the breakdown.

Dataset	Videos	Glosses	Train	Val	Test
WLASL	295	20	214	43	38

Table 6.1: Dataset Statistics.

Everything was tested on the WLASL benchmark. The raw video clips and their information were downloaded from the `WLASL_v0.3.json` file and the data was prepared in the following stages:

- **Automated Video Ingestion:** A special script called `video_loader.py` was used to read the JSON list rows and set up download links. It saved the raw video files to the local computer.
- **Temporal Alignment & Frame Extraction:** A `frame_extractor.py` tool based on OpenCV was used to get frame sequences from each video clip. To make sure all clips had the same structure, 22 equal frames were taken from each videos active part. This way the main movement of the sign was captured. The still parts, at the

beginning and end were ignored. The `WLASL_v0.3.json` file was used to get the raw video clips and their metadata.

- **Dataset Partitioning:** The pre-processed frame sequences were split into training, validation and testing subsets.The split ratio was **70:15:15**. This way the classes were balanced across all partitions.

- **Downstream Model Ingestion:** We used the frame arrays as inputs to test and improve two models:

  ResNet50 + BiLSTM hybrid network that focuses on appearance and an the coordinate-based **ASL Transformer** model.

6.2 Experimental Setup

To keep fair comparison , both the appearance-based (ResNet50 + BiLSTM) and the pose-based (ASL Transformer) models were trained, validated and tested on same hardware and software. Everything was run on a Kaggle kernel.

Hardware and Software Environment

- **Computational Hardware:** We used a NVIDIA T4 GPU with 16GB of VRAM along with 2 virtual CPUs and 13GB of system RAM to speed up execution.

- **Operating System & Runtime:** Our setup was an Ubuntu 22.04 LTS container running Python 3.10.x.

- **Deep Learning Frameworks:**We executed and tested the core models using TensorFlow 2.15.x and Keras 2.15 with NumPy 1.24.x for matrix operations and scikit-learn 1.2.x for splitting data doing PCA whitening and calculating classification metrics.

- **Visual Data Parsing:** We used OpenCV 4.8.x for decoding video frames cropping and resizing them and MediaPipe 0.10.x for extracting landmarks all done before the actual training.

Paradigm Input Specifications. The training loops were matched, but what actually gets fed into the sequence model is fundamentally different in each case:

- **ResNet50 + BiLSTM Stream:** Individual video frames are passed through a frozen, pre-trained ImageNet ResNet50 base. The spatial feature maps are squashed via global average pooling into 2048-dimensional appearance vectors across the 22 frames, cached locally as a CSV matrix. Before the recurrent stage these features are standardized and reduced with PCA whitening (fit on the training frames only,

2048 → 128 components) to counter the curse of dimensionality, then passed into a heavily regularized recurrent head of two stacked Bidirectional LSTM layers (256 and 128 units) with dropout, ending in a dense classifier.

- **ASL Transformer Encoder Stream:** Raw pixels are discarded immediately by a MediaPipe Holistic pipeline. Each frame is converted into a flat 225-dimensional skeletal vector (33×3 holistic pose points, 21×3 left-hand landmarks, 21×3 right-hand landmarks). The coordinates are first pose-normalized (centered on the shoulder midpoint and scaled by shoulder width) so the model sees gesture geometry rather than the signer's position or size. The normalized 22-frame sequence is mapped into a 128-dimensional embedding, a learnable CLS token is prepended (length 23), sinusoidal positional encoding is added, and the sequence is processed by a 3-block, 4-head Multi-Head Self-Attention Transformer Encoder stack.

6.3 Training Configuration

Both models are written in TensorFlow/Keras and trained on Kaggle. The main settings are:

ResNet50 + BiLSTM

- Optimizer: Adam; learning rate = 1×10^{-4}
- Batch size: 8; max epochs: 250
- Patience: 30 (early stopping on validation accuracy, best weights restored)
- Loss function: `sparse_categorical_crossentropy`
- Input feature dimension: 2048 (ResNet50), reduced to 128 via PCA whitening
- Regularization: L2 kernel penalty = 1×10^{-3}; `DROPOUT_RATE` = 0.5

ASL Transformer

- `D_MODEL` = 128; `NUM_HEADS` = 4; `FF_DIM` = 256; `NUM_LAYERS` = 3
- `DROPOUT_RATE` = 0.20; `BATCH_SIZE` = 8
- `EPOCHS` = 110; `PATIENCE` = 30
- Optimizer: AdamW with warmup–cosine schedule; `PEAK_LR` = 3e-4; `WEIGHT_DECAY` = 3e-4
- Loss: `sparse_categorical_crossentropy` with label smoothing = 0.10

6.4 Evaluation Metrics

The 20 glosses are chosen in a way that's fair and balanced so each type of sign has the same importance in the test. That makes **classification accuracy** a fair primary measure, and the word-level sign-language literature, including the WLASL benchmark also uses this measure. I used accuracy in two forms:

- **Top-1 accuracy:** This is when our system makes one guess. It is the correct sign. This is the way of looking at it like we do every day.
- **Top-k accuracy:** This is when our system gives us a signs that it thinks are correct and the real sign is one of them. I am looking at Top-3, which means our system gives us three signs to choose from. This way of measuring is more relaxed. It makes sense for a system that is supposed to help people. It is better because our system can give the user an options to pick from instead of just one guess.

Generally, for a test set of N clips,

$$\text{Top-}k \text{ Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\Big[y_i \in \hat{Y}_i^{(k)} \Big], \tag{6.1}$$

where y_i is the true gloss of clip i, $\hat{Y}_i^{(k)}$ is the set of the k glosses the model scores highest for that clip, and $\mathbb{1}[\cdot]$ is the indicator function, equal to 1 when the condition inside holds and 0 otherwise. When I set $k = 1$, it gives Top-1 accuracy.I keep an eye on the training and validation accuracy in the way when I am training my model and I look at the difference, between the two to catch overfitting early on.

6.5 Performance Comparison and Analysis

I compare how each model handles its type of input. On one hand I have the 2048-feature summaries. On the hand I have the 225-D skeleton coordinates. I want to see how each model performs with these types of input the 2048-D feature summaries and the 225-D skeleton coordinates.

Model Paradigm	**Train Acc (%)**	**Val Acc (%)**	**Test Acc (%)**	**Top-3 (%)**
ResNet50 + BiLSTM (10 Glosses)	79.37	69.56	52.17	—
ResNet50 + BiLSTM (20 Glosses)	85.84	30.25	21.06	52.21
ASL Transformer (10 Glosses)	100.00	95.61	93.46	—
ASL Transformer (20 Glosses)	92.23	61.31	47.56	85.58

Table 6.2: Performance comparison of models.

6.6 Analysis

Two things stand out in Table 6.2. First the ASL Transformer that uses poses does better than the ResNet50 + BiLSTM that uses appearances. It does better at every size and on every split. For example on the 20-gloss task the ASL Transformer gets **47.56%** but the ResNet50 + BiLSTM only gets **21.06%** correct. On the 10-gloss task, which is easier the ASL Transformer does even better, getting **93.46%** correct while the ResNet50 + BiLSTM only gets **52.17%** correct. Second, both models do really better on the training data but they do not do well on the test data. The difference between how they do on the training data and the test data is big but it is bigger for one model than the other. This difference is important because it tells us a lot about what's going on.

Generalization and Overfitting

The obvious thing we see in the results is how much each model overfits. Table 6.3 shows the difference between how each model does on the training data and the test data, for all four runs.

Model	Train Acc (%)	Test Acc (%)	Gap (pp)
ResNet50 + BiLSTM (10 Glosses)	79.37	52.17	27.2
ResNet50 + BiLSTM (20 Glosses)	85.84	21.06	64.8
ASL Transformer (10 Glosses)	100.00	93.46	6.5
ASL Transformer (20 Glosses)	92.23	47.56	44.7

Table 6.3: Generalization gap, measured as training minus test accuracy.

The hybrid is really bad at this. It looks at 20 glosses. It gets almost all of them right about (85.84%). Then it gets to the test part and it only gets 21.06% right, which is a big difference of almost **65 points**. The hybrid does on the validation part it gets about (30.25%) percent right, which is just a little better, than the test part. This tells me that the hybrid is not just having a day it is really failing to understand what it is looking at.

The Transformer is not great either it has a gap of about 45 percent when it looks at 20-glosses. But it does a lot better when it looks at things and when it only looks at ten examples the gap is really small only about (6.5 points, at 93.46% test accuracy). Figure 6.1 tells the same story as a curve. The training loss keeps getting smaller. The validation loss stops getting smaller and then starts to get bigger after a few rounds. This is what happens when a model is just memorizing things of really learning them. That is why I do not change the ResNet backbone and why I stop the training early. If I did not do these things the gap would be even bigger.

Figure 6.1: Training and validation loss over epochs.

Effect of Vocabulary Size

When we increase the number of glosses from 10 to 20 the task becomes more difficult for both the models. They are affected because the classification task becomes difficult in absence of sufficient data. The model starts overfitting. If data is sufficient then the training will be perfect. Even the data were less, Transformer model was able to beat Resnet50 + BILSTM model.

Top-1 versus Top-3

The Top-3 column is encouraging for both models. On 20 glosses the Transformer climbs from 47.56% (top-1) to **85.58%** (top-3), and the hybrid from 21.06% to 52.21%. Put another way, even when the Transformer's single best guess is wrong, the correct sign sits among its top three candidates around **six times out of seven**. That matters for how a system like this would actually be used: in an assistive tool it does not have to be right on the first attempt if it can show the user a short list to choose from, and on that measure the pose-based model is already close to usable.

Why the Pose-Based Model Wins

All four runs share the same videos, the same splits, the same frame count and the same training budget. The only thing that changes between the two models is what each one is allowed to look at. The hybrid reads ResNet's 2048-dimensional appearance features, which encode everything in the frame, the background, the lighting, the signer's clothing included. With only about 13 clips per class, the cheapest way for it to drive the training loss down is to seize on those incidental cues, and those cues do not carry over to new signers or new recordings, which is what produces the collapse on the test set. The Transformer never sees any of that. It works on 225 MediaPipe coordinates that have

already been pose-normalized to the signing space, so the only thing left to learn is the geometry of the gesture, where the hands are and how they move. Reducing the input to that geometry removes most of what there is to overfit, and self-attention then links a hand shape early in the clip to a pose later on. So the gap is not really ResNet against a Transformer; it is appearance against geometry under data scarcity, and geometry wins.

How Far to Trust These Numbers

One caveat keeps the results honest. The 20-gloss test set holds only 38 clips (Table 6.1), so each clip is worth about 2.6 percentage points. A 47.56% score is roughly 18 of 38 clips correct and a 21.06% score is about 8 of 38, and with counts that small the confidence intervals are wide, on the order of ± 13 to ± 16 percentage points at the 95% level. I would therefore not over-read the exact figures. The *gap* between the two models, on the other hand, does survive this uncertainty: treating the two test outcomes as proportions, the difference between 18/38 and 8/38 works out to about $z \approx 2.4$ ($p \approx 0.02$), so the Transformer's advantage is unlikely to be an accident of one lucky split, even if the precise accuracy values are best read as estimates. This is also why the stratified cross-validation reported earlier is a more reliable summary of the pose model than any single test split.

Chapter 7

Conclusion and Future Work

To wrap things up I want to talk about what I learned from comparing the two models be honest about the problems I had with the data and think about where this could go from here.

7.1 Summary of Contributions

During this project, I compared two very different ways of recognizing American Sign Language words one word at a time using a small set of 20 words.

- The first was a hybrid **ResNet50 + BiLSTM** network that looks at raw video pixels (an appearance-based model).
- The second was an **ASL Transformer Encoder** that throws away pixels completely and only tracks skeletal joint numbers (a pose-based model).

I ran both of these methods under the conditions using the same small set of data to see which one worked better when there was not a lot of data to work with. It turned out that looking at pictures and looking at the positions of the joints were very different:

- **The Pixel-Model Problem (ResNet50 + BiLSTM):** This method had a lot of trouble recognizing videos it was only able to correctly identify the signs about 5.26% of the time. Though using a pre-trained ResNet50 model gave me a good starting point it caused some problems like caused "catastrophic forgetting" when I tried to use it with my small set of data. The model started to forget what it had learned before and instead of learning the shapes of the hands it started to remember things like the color of the background the lighting in the room and the clothes the people in the videos.
- **The Skeleton-Model Success (ASL Transformer):** On the other hand, the method that looked at the positions of the joints worked very well it was able to

correctly identify the signs about $59.00 \pm 1.62\%$ of the time and it was able to narrow it down to the top 5 signs, about 78.45% of the time. By using MediaPipe to turn each frame of the video into a list of 225 joint positions the Transformer model was able to do a great job of recognizing the American Sign Language words.

The big takeaway: If you are building a sign recognizer and do not have millions of videos to use in training, **a pose-based attention model is the better choice** by a wide margin. It runs faster, remove background noise, and draws far steadier decision boundaries.

7.2 Limitations

The single biggest obstacle in this work was the shape of the dataset. On paper WLASL spans up to 2,000 words, but the number of clips per word is wildly uneven.

I had originally hoped to scale the recognizer up to 1,000 words. Parsing the data brought a quick reality check: **more than 95% of the words had fewer than 10 clips each.**

Deep models need plenty of varied examples to learn anything solid. Asking a 1,000-class model to learn from 2 to 5 clips per word is simply asking too much; with so little variety it overfits almost at once, and standard K-fold cross-validation is not even mechanically possible, because there are not enough clips to fill separate folds. Given that, I narrowed the scope to a balanced subset of words with at least 10 clips each, so that the numbers I report are statistically honest.

7.3 Future Work

Building on this, and trying to get past the data problems, here are the directions I would most like to take it:

- **Moving to Better Data Lexicons (WLASL-alt):** To solve the data scarcity problem directly, the next step will be moving from the original dataset to an enhanced version called WLASL-alt (alternate). This cleaner archive fixes a lot of original labeling mistakes and adds tons of fresh, high-quality video examples. This will allow the models to comfortably scale up to hundreds of additional words without crashing from a lack of samples.

- **Anatomy-Aware Graph Neural Networks (ST-GCN):** Right now, the Transformer reads the 225 MediaPipe coordinates as a flat, random list of numbers. In the future,Anatomy-Aware Graph Neural Networks, also known as ST-GCN is something I am looking into. I want to model the skeleton like a graph. In this graph

the joint tracking dots are like nodes. The physical arm and finger bones are, like edges that connect these nodes. If I pass this data through a Spatio-Temporal Graph Convolutional Network or ST-GCN before it reaches the Transformer, it will help the computer learn the structure of the human body. This will make gesture tracking more accurate because the computer will understand how the body parts are connected. The ST-GCN will teach the computer about the body. The human body has a structure that ST-GCN will help the computer learn. MediaPipe coordinates will be used to make this happen. The Transformer will then use this information to track gestures. Gesture tracking will be more accurate because of this. The ST-GCN and Transformer will work together. It will help the Transformer understand the body.

- **Self-Supervised "Masked" Pre-training (BERT for Sign Language):** To stop relying on collections of labeled data we can use a trick that language models like BERT use. We can take a lot of sign language videos cover up some of the hand joints and make a computer program figure out where those hidden joints are. Sign language is what we are trying to teach the computer program. Once the computer program understands how people naturally move their bodies when using sign language we can use it with collections of labeled sign language data and it will work really well.

- **Multi-Modal Fusion (Combining Pixels and Poses):** Finally, a hybrid model that combines two things: the pictures of the person signing and the movements of their hands and body. Sign language is what we are trying to understand. If we just look at the pictures the computer program might get confused.. The pictures are good for seeing the face and lips of the person signing, which is important for sign language. We can make a system with two parts: one part looks at the hands. How they move and the other part looks at the face. Sign language is what we want to translate. This way we can make a system that understands sign language. We can combine the pictures and the movements to make a system for sign language. Sign language is what we are trying to understand. We can use a kind of computer program that has two parts: one part for the hands and one part, for the face. This will help us make a system that really understands sign language.

Bibliography

[1] G. D. Revankar, and S. S. Kumar, "American Sign Language Recognition Using Deep Learning Models" in Proceedings of International Conference on Information Technology and Applications,2025. https://link.springer.com/book/10.1007/978-981-96-1758-6

[2] "Sign language recognition—an overview," ScienceDirect Topics. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/sign-language-recognition

[3] D. Li, O. C. Rodriguez, Y. Xin, and L. Hongdong, "Word-level deep sign language recognition from video: a new large-scale dataset and methods comparison," in *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1448–1458, 2020.

[4] A. Tunga, S. V. Nuthalapati, and J. Wachs, "Pose-based sign language recognition using GCN and BERT," *arXiv preprint*, 2020.

[5] D. Li et al., "WLASL: WACV 2020 —Word-level deep sign language recognition from video: a new large-scale dataset and methods comparison," 2020.

[6] Y. Jiang, F. Li, Z. Li, Z. Liu, and Z. Wang, "Enhancing continuous sign language recognition with self-attention and MediaPipe holistic," in *2023 8th International Conference on Instrumentation, Control, and Automation (ICA)*, pp. 97–102, 2023.

[7] F. Nugraha and E. C. Djamal, "Video recognition of American sign language using two-stream convolution neural networks," in *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 400–405, 2019.

[8] L. T. Woods and Z. A. Rana, "Modelling sign language with encoder-only transformers and human pose estimation keypoint data," *Mathematics*, vol. 11, no. 9, 2023.

[9] F. Chollet, "Xception: deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 1800–1807, 2017.

[10] M. Banoula, "Introduction to long short-term memory (LSTM)," Simplilearn, 2023. [Online]. Available: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm

[11] E. Zvornicanin, "Differences between bidirectional and unidirectional LSTM," Baeldung, 2022. [Online]. Available: https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm

[12] H. Mujtaba, "Introduction to ResNet or residual network," Great Learning, 2020. [Online]. Available: https://www.mygreatlearning.com/blog/resnet/

[13] D. Zhang, D. Yin, L. Chen, J. Zhang, and J. Tang, "Graph-BERT: Only attention is needed for learning graph representations," *arXiv preprint arXiv:2001.05140*, 2020.

[14] Baskoro R (2021) WLASL (World Level American Sign Language) video

[15] Nikolas Adaloglou, Theocharis Chatzis, Ilias Papastratis, Andreas Stergioulas, Georgios Th Papadopoulos, Vassia Zacharopoulou, George J Xydopoulos, Klimnis Atzakas, Dimitris Papazachariou, and Petros Daras. A comprehensive study on sign language recognition methods. *arXiv preprint arXiv:2007.12530*, 2020.

[16] Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. Sign language transformers: Joint end-to-end sign language recognition and translation. *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10023–10033,*, 2020.

[17] Helen Cooper, Eng-Jon Ong, Nicolas Pugeault, and Richard Bowden. Sign language recognition using sub-units. *J. Mach. Learn. Res., 13(1):2205–2231*, July 2012.

[18] Runpeng Cui, Hu Liu, and Changshui Zhang. Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. *In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1610–1618* 2017.

[19] Runpeng Cui, Hu Liu, and Changshui Zhang. A deep neural framework for continuous sign language recognition by iterative training. *IEEE Transactions on Multimedia, 21(7):1880–1891* 2019.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *In International Conference on Learning Representations* 2020.

[21] Al Amin Hosain, Panneer Selvam Santhalingam, Parth Pathak, Huzefa Rangwala, and Jana Kosecka. Hand pose guided 3d pooling for word-level sign language recognition. *In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 3429–3439* 2021.

[22] Songyao Jiang, Bin Sun, Lichen Wang, Yue Bai, Kunpeng Li, and Yun Fu. Skeleton aware multi-modal sign language recognition. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pages 3413–3423* June 2021.

[23] Dimitrios Konstantinidis, Kosmas Dimitropoulos, and Petros Daras. A deep learning approach for analyzing video and skeletal features in sign language recognition. *In 2018 IEEE International Conference on Imaging Systems and Techniques (IST), pages 1–6. IEEE* 2018.

[24] Bruce Xiaohan Nie, Caiming Xiong, and Song-Chun Zhu. Joint action recognition and pose estimation from video.

In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1293–1301 2015.