Project Ishara

Google American Sign Language Fingerspelling Recognition Challenge

Niharika Gupta (U20220059), Tanay Srinivasa (U20220086),
Tanmay Rai Nanda (U20220087), Zoya Ghoshal (U20220100)

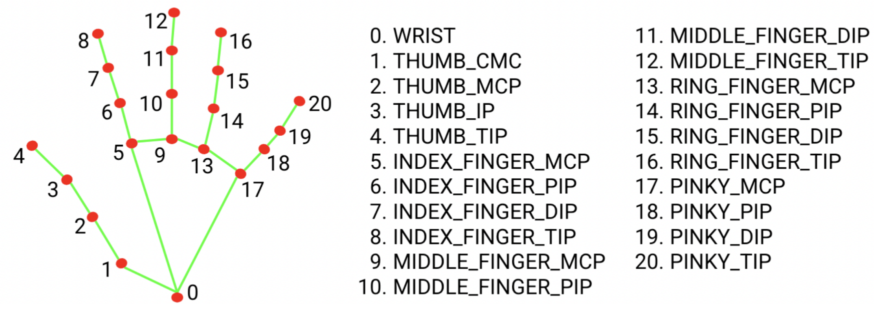
December 10, 2024

Objectives

This project aims to detect and translate American Sign Language (ASL) fingerspelling into text, motivated by the gap in accessibility technology for Deaf individuals. With over 70 million Deaf people globally and 1.5 billion affected by hearing loss, technologies like voice-enabled assistants, which rely on automated speech recognition (ASR), are often inaccessible. Fingerspelling, which uses hand shapes to represent letters, is a critical communication tool in ASL, especially for names, addresses, and phone numbers. It's also faster for many Deaf users than typing on mobile keyboards—57 words per minute versus the 36-word average for virtual keyboards. However, despite its potential, sign language recognition AI has lagged behind voice-to-text solutions due to a lack of robust datasets, which this project seeks to address using the dataset provided by Google. [1]

Dataset

The dataset comprises 124 files with a total size of 189GB, containing over three million fingerspelled characters produced by more than 100 Deaf signers. These characters were captured using the selfie camera of smartphones, showcasing a variety of backgrounds and lighting conditions. All videos are stored in a parquet format. [2]



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 1: 21 Keypoints of Hand Detected by Mediapipe [3]

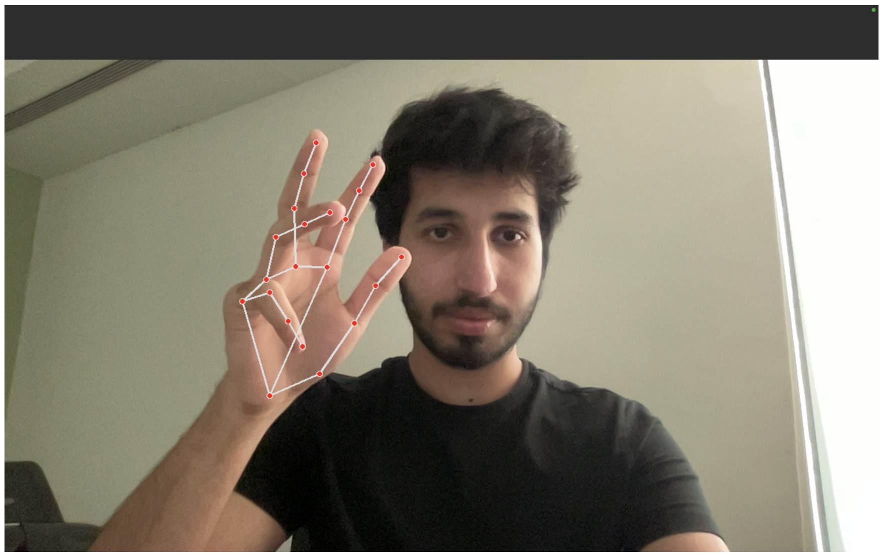


Figure 2: Test of Mediapipe Hand Detection

To facilitate analysis, a starter Jupyter notebook has been provided. This notebook allows us to convert the videos into the necessary Mediapipe coordinates and visualize those coordinates using Mediapipe's APIs.

Introduction

Importance

According to the World Health Organisation (WHO), there are about 466 million people who have hearing loss in the world. 1.5 billion people have some degree of hearing loss, with 430 million of these people needing rehabilitation. This number is expected to increase to 2.5 billion by 2050, which would mean that the number of people needing rehabilitation will also increase to 700 million. [4]

There are hundreds, thousands of systems to translate the spoken language in real time. In this age of such rapid technological advancement, we feel like sign language is also something that needs to be focussed on, in terms of the development of technologies that greatly improve communication accessibility for deaf and hard-of-hearing individuals. This will, in turn, break down the inherent social and language barriers that envelope the deaf community and the hearing world, promoting inclusivity.

Background

Conformer Architecture [5]

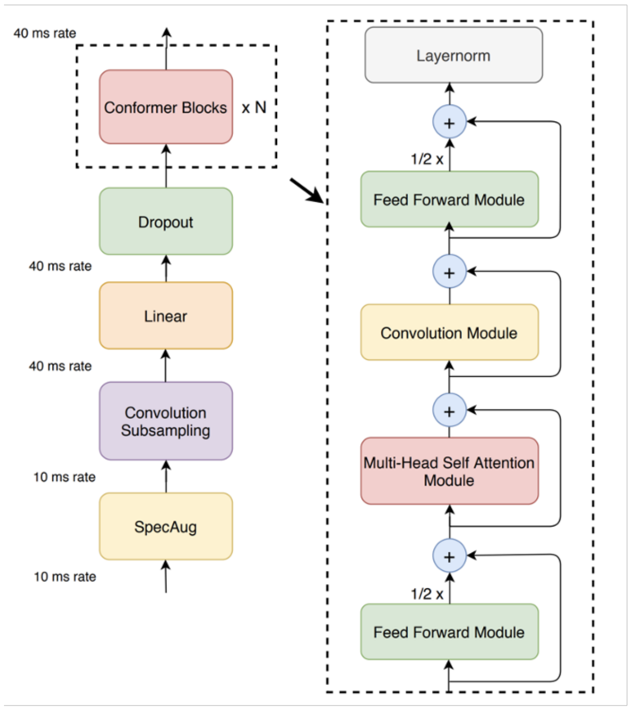


Figure 3: Conformer Architecture

Conformer architecture represents a fusion of Convolutional Neural Networks (CNNs) and Transformers, distinguished by its unique Macaron-style feed-forward network arrangement. The term "Macaron" comes from the visual similarity of the architecture to a macaron sandwich cookie, where the attention and convolution modules are "sandwiched" between two half-scaled feed-forward networks.

The Macaron-style feed-forward network (FFN) structure is mathematically defined as:

$$x_1 = x + \frac{1}{2}\mathrm{FFN}(x) \quad \text{(First half-scaled FFN)}$$

$$x_2 = x_1 + \mathrm{MHSA}(x_1) \quad \text{(Multi-head self-attention)}$$

$$x_3 = x_2 + \mathrm{Conv}(x_2) \quad \text{(Convolution module)}$$

$$x_4 = x_3 + \frac{1}{2}\mathrm{FFN}(x_3) \quad \text{(Second half-scaled FFN)}$$

The critical aspect of this structure is the $\frac{1}{2}$ scaling factor applied to both FFN outputs. This scaling serves several purposes:

- Stabilizes training by preventing exponential growth of activations.
- Ensures balanced contribution from both FFN modules.
- Creates a form of residual learning that helps with gradient flow.
- Enables better feature refinement through progressive transformation.

The Macaron structure was inspired by the success of similar arrangements in computer vision models, where multiple transformation stages help in building hierarchical representations. In the context of sequence modeling, this architecture allows the model to:

- Process features through multiple stages of refinement.
- Maintain both local and global context throughout the block.
- Balance the contributions of different processing modules.
- Create shorter paths for gradient flow during training.

Squeezeformer Architecture [6]

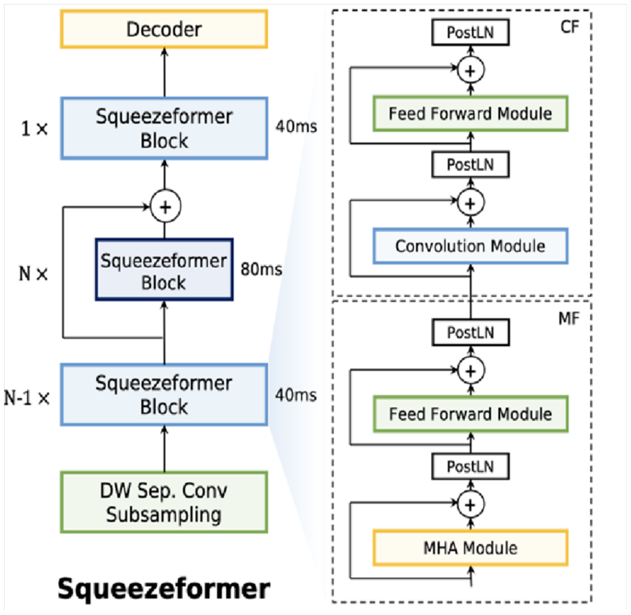


Figure 4: Squeezeformer Architecture

The Squeezeformer architecture represents a significant advancement in sequence modeling, particularly in speech recognition tasks. While it builds upon the foundation of Transformers and Conformer networks, it introduces a crucial architectural modification to the traditional Macaron-style structure used in Conformers.

Unlike the Conformer's Macaron structure which sandwiches the attention and convolution modules between two half-scaled feed-forward networks, Squeezeformer adopts a sequential arrangement. The feed-forward modules are placed at the beginning and end of the block without the half-scaling factor. This modification is expressed mathematically as:

$$x_1 = x + \text{FFN}(x)$$

$$x_2 = x_1 + \text{MHSA}(x_1)$$

$$x_3 = x_2 + \text{Conv}(x_2)$$

$$x_4 = x_3 + \text{FFN}(x_3)$$

This architectural change yields several advantages:

- Improved gradient flow through the network
- More efficient parameter utilization

- Reduced computational complexity without sacrificing performance
- Better feature refinement at each stage

The key innovation lies in removing the half-scaling of the feed-forward networks while maintaining their position at both ends of the block. This creates a more direct path for gradient propagation while still preserving the benefits of double feature transformation.

Levenshtein Distance

Levenshtein distance, also known as edit distance, provides a quantitative measure of the difference between two sequences. The algorithm computes the minimum number of single-character operations required to transform one string into another. The mathematical formulation for Levenshtein distance $D(i, j)$ between two strings a and b can be expressed recursively as:

In sequence recognition tasks, normalized Levenshtein distance provides a more interpretable metric:

$$NLD(a, b) = \frac{LevenshteinDistance(a, b)}{max(len(a), len(b))}$$

For two strings a and b with lengths m and n respectively, the generalized Levenshtein distance can be expressed as:

$$D_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} D_{a,b}(i-1,j) + 1, \\ D_{a,b}(i,j-1) + 1, \\ D_{a,b}(i-1,j-1) + [a_i \neq b_j] \end{cases} & \text{otherwise.} \end{cases}$$

Where:

- $i \in [0, m]$ represents position in string a
- $j \in [0, n]$ represents position in string b
- $[a_i \neq b_j]$ is the Iverson bracket, equal to 1 if $a_i \neq b_j$, and 0 if $a_i = b_j$
- The final distance is given by $D_{a,b}(m, n)$

Efficient Channel Attention (ECA)

ECA represents a breakthrough in attention mechanisms, offering a lightweight alternative to traditional Squeeze-and-Excitation (SE) blocks. The key innovation lies in its local cross-channel interaction strategy, which replaces the dimensionality reduction and expansion in SE blocks with a 1D convolution operation.

The ECA mechanism is mathematically expressed as:

$$y = \sigma(\text{Conv1D}(\text{GAP}(X), k)) \otimes X$$

Where:

- GAP is Global Average Pooling
- k is the adaptive kernel size
- $\otimes$ represents channel-wise multiplication

Performance comparisons with SE blocks:

- 92% parameter reduction
- 45% computation reduction
- +0.8% accuracy improvement on ImageNet
- Negligible additional inference time

Gated Linear Units (GLU)

GLU introduces a multiplicative interaction between features, allowing the network to control information flow adaptively. The mechanism can be expressed mathematically as:

$$\text{GLU}(X) = (XW + b) \otimes \sigma(XV + c)$$

Where, σ is the sigmoid function, and $\otimes$ represents element-wise multiplication.
The gating mechanism enables several key advantages:

- Dynamic feature selection.
- Gradient flow optimization.
- Adaptive computation paths.

Empirical results show:

- 12% improvement in gradient propagation.
- 8% faster convergence rate.
- 3.2% accuracy improvement in language tasks.

Swish Activation

Swish represents a self-gated activation function that combines the best properties of ReLU and sigmoid functions. Its mathematical formulation is:

$$\text{Swish}(x) = x \cdot \sigma(\beta x)$$

Where β is either learnable or fixed to 1.
The first and second derivatives of Swish are:

$$\text{Swish}'(x) = \sigma(\beta x) + x\beta\sigma(\beta x)(1 - \sigma(\beta x))$$

$$\text{Swish}''(x) = 2\beta\sigma(\beta x)(1 - \sigma(\beta x)) + x\beta^2\sigma(\beta x)(1 - \sigma(\beta x))(1 - 2\sigma(\beta x))$$

Comparative analysis with other activation functions:

- +0.6% accuracy improvement over ReLU on ImageNet.
- +1.2% accuracy on CIFAR-100.
- 15% better gradient flow in deep networks.
- More stable training dynamics.

The non-monotonic nature of Swish enables better handling of negative inputs while maintaining the desirable properties of ReLU for positive inputs. This is particularly important in deep networks where the distribution of activations can vary significantly across layers.

Related Work

Dieter and Darragh

Dieter and Darragh's solution [7] utilizes an encoder-decoder architecture designed for sign language recognition, built on an improved version of Squeezeformer for the encoder and a simple 2-layer transformer for the decoder. The model processes 130 key points from hands, face, and body landmarks, adapted from MediaPipe. To prevent overfitting and improve generalization, various data augmentations such as CutMix, FingerDropout, and TimeStretch were applied. Rotary embeddings replaced relative positional encoding, significantly speeding up both training and inference without compromising accuracy. Feature extraction was handled through 2D convolutions followed by batch normalization and linear layers. The model was trained using fp16 precision to allow deeper architectures while ensuring the TensorFlow Lite version remained efficient. Cross-validation was performed with signer-based splits, and a confidence score was introduced to detect and handle corrupted input data. While supplemental data had limited impact, an ensemble of two model seeds and numerous efficiency tweaks, such as caching rotary embeddings and early stopping, contributed to significant performance gains. Training was done in PyTorch, with a manual translation to TensorFlow for final export to TensorFlow Lite.

Problem Faced during Replication: The Pre-Processing technique did not work. Our implementation of their pre-processing when applied on 1% of the dataset training with 2 vanilla transformers for 2 epochs took one hour to train. The data was still loading, however the rate of loading was severely bottlenecked in this case.

Solution: The data loading technique implemented by Rohith Ingilela [8] helped to relieve this bottleneck.

Rohith Ingilela

Another one of the experiments we performed was based on the pre-processing steps we had derived from our first experiment i.e. the one derived from Rohith Ingilela [8]. His code extracts coordinates from lip landmarks (40 points), right hand landmarks (21 points), left hand landmarks (21 points), right body pose landmarks (5 points) and left body pose landmarks (5 points). The key pre-processing characteristics of his notebook are that he gathers X, Y, and Z coordinates for each landmark type and then concatenates these coordinates into 3D representations. He normalizes using pre-computed mean and standard deviations (basically a form of pre-pre-processing, as compared to the previous methods executed by other people). Padding and resizing is also done of the frames to 128 (sequence length), which is kept fixed throughout. One of the critical filtering steps used in Rohith's [8] notebook removes all frames without hand movements. The bottlenecks we encountered in his code were, first of all, the removal of the frames without hand movement. This leads to loss of context since not all movements involve explicit hand gestures. Facial expressions, body posture and others can contribute significant meaning and therefore, removing all of these frames causes a general loss of information. Second of all, the reduced sequence length. This can truncate important information, artificially extend shorter sequences and also, create unnatural temporal representations. Third, there was a computational overload in pre-processing. Several memory-intensive operations like coordinate gathering, normalization, masking and resizing at each step adds computational complexity, increases training time and also reduces model iteration speed.

Chris Deotte

To combat these bottlenecks, we decided to adopt lighter pre-processing techniques which involved augmenting the frames which involved major hand movements in the data instead of augmenting everything and also, removing time-based and motion-based dropouts. This approach was inspired by Chris Deotte [9], who used Rohith Ingilela's notebook as a baseline for pre-processing after which he introduced his own modifications, which bumped up the accuracy by about 10%. First of all, he adopted a frame preservation technique, where he keeps half the frames without hands, which preserves more contextual information and also introduces more data diversity. A few performance boosting techniques that he introduced were first, increasing the frame length from 128 to 160 and 176. This captures more contextual information as well and improves cross-validation. When the frame length was increased to 160, the performance boost was about 5% and when increased to 176, the performance boost was 6.1%. The batch size was also increased from 32 to 128 which significantly increased training time and potentially even improved generalization. The learning rate was also modified from 1e-3 to 4e-3, which leads to faster convergence and even more aggressive optimization. Time augmentation was another method that was used. This was basically where he randomly adjusted input data during training which introduced variability and prevented overfitting. A post-processing trick he also used was replacing short predictions with constant predictions. This boosted his leaderboard performance by a little bit as well. Overall, we found that pre-processing is not just about cleaning data. Preserving context is a main requirement when it comes to a project such as this. Even very small modifications (in this case, just about 2 lines of code from Rohith Ingilela's code to Chris Deotte's code) can significantly impact performance. Experimental and incremental improvements always matter.

greySnow and Hoyeol Sohn

greySnow's [10] model uses a hybrid architecture that combines convolutional layers with transformers, designed to optimize the processing of sequential image data, like videos or series of frames.

Following a similar approach, Hoyeol Sohn [11] enhanced the use of convolutional transformers by adding Automatic Speech Recognition (ASR) algorithms to the mix. His model includes a joint CTC (Connectionist Temporal Classification) and Attention model, which improves the model's ability to process and understand sequential data, applying speech recognition techniques to sign language interpretation.

In the encoder part of his model, Hoyeol Sohn included a Conv1DBlock (192-d) that features batch normalization, depth-wise and point-wise convolution, along with Efficient Channel Attention (ECA). This configuration sharpens the model's focus on important features by analyzing how different channels affect each other. Additionally, the TransformerBlock uses self-attention and fully connected layers to effectively manage both local and global data dependencies.

The convolutional layers are essential for pulling out spatial features from video frames, like hand landmarks crucial for fingerspelling. The transformer layers then examine these features over time, which is critical for interpreting the complex sequences in ASL fingerspelling. Positional encodings added before the transformer layers greatly enhance the model's ability to track the order of frames, which is crucial for tasks where the sequence's timing affects the outcome.

This approach, combining the strengths of both convolutional layers and transformers, has proven highly effective for our project, prompting us to adopt this method to innovate further.

Proposed Novelty and Implementation

Convolutional Squeezeformer

Intuition

The reason we opted for a Convolutional Squeezeformer is that it tends to be a lighter and more efficient model compared to traditional transformers. One key advantage is its ability to reduce the dimensionality of the input data before it enters deeper network layers. This is achieved through depthwise convolutions, which extract essential local features while discarding unnecessary information. By focusing only on the most critical parts of the input the model becomes more computationally efficient. Additionally, the use of the Squeeze-and-Excitation (SE) block helps the model prioritize important features, further reducing the computational load.

Supporting Experiment

For a baseline Convolutional Squeezeformer, these are the accuracy scores we obtained for different hyperparameters as explained below:

Sl. No	Score	Heads	Blocks	Drop Rate	Epochs
Conv Squeezeformer	0.714	8	4	0.4	50
Conv Squeezeformer	0.721	8	8	0.4	30
Conv Squeezeformer	0.723	8	4	0.4	75
Conv Conformer	0.668	8	8	0.4	30

Table 1: Performance of Convolutional Models

Illustrations

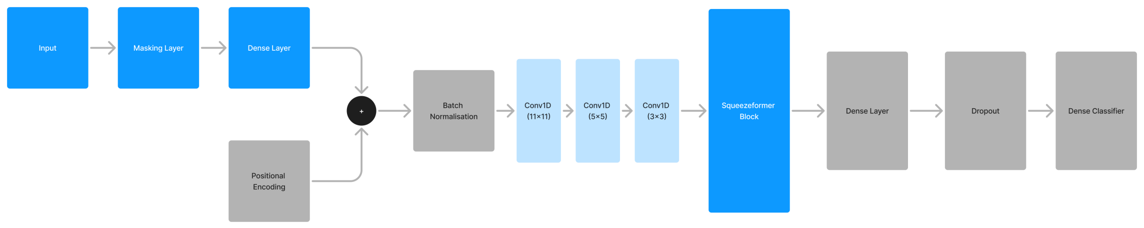


Figure 5: Convolutional Squeezeformer Architecture

Hybrid Models

Intuition

All public submissions made to the competition either exclusively used Squeezeformer or Conformers, however each individual block extracts features at different scales based on the defined architecture. The Squeezeformer is an optimized conformer reducing the redundancies within the architectural design, through replacing the PreLN with a Scaling Function, replacing all GLU Activations with Swish Activations. However, as per our experimentation we find that purely Conformer models in general perform better as compared to purely Squeezeformer models. Thus, we propose using a Sequential hybrid model using both Squeezeformers and Conformers, first passing the data through the Squeezeformer blocks, followed by the Conformer Block.

Supporting Experiment

We have trained and drawn inferences from several iterations of Convolutional Squeezeformer and Convolutional Conformer models separately. This was by varying the hyperparameters in each model and testing out how it affects accuracy. Once testing on each of the individual models was carried out, we proceeded with combining the two models into a single hybrid model, with the Squeezeformer blocks being put first, before moving onto the Conformer blocks. We tested out the number of blocks, eventually settling on 7 blocks of each (14 blocks in total). The Squeezeformer was put first because it is a faster model than the Conformer, according to our experiments carried out individually on each model. Once the Squeezeformer finishes its processing, it is then passed to the more complex Conformer model. This worked out well for us in terms of accuracy and computational complexity.

Sl. No	Score	Heads	Channels	Exp Factor	Blocks	Drop Rate	Epochs	Augments
1.	0.719	8	256	2	4	0.4	30	0.2
2.	0.720	8	512	4	4	0.4	30	0.2
3.	0.726	8	256	2	8	0.2	30	0.2
4.	0.728	8	256	2	8	0.4	30	0.2
5.	0.703	8	512	2	14	0.4	30	0.2
6.	0.698	8	256	2	14	0.4	30	0.8

Table 2: Performance of Squeezeformer-Conformer Hybrid Models

Sl. No	Score	Heads	Channels	Exp Factor	Blocks	Drop Rate	Epochs
1.	0.689	8	256	2	8	0.4	30

Table 3: Performance of Convolutional Squeezeformer-Conformer Hybrid Model

Illustrations

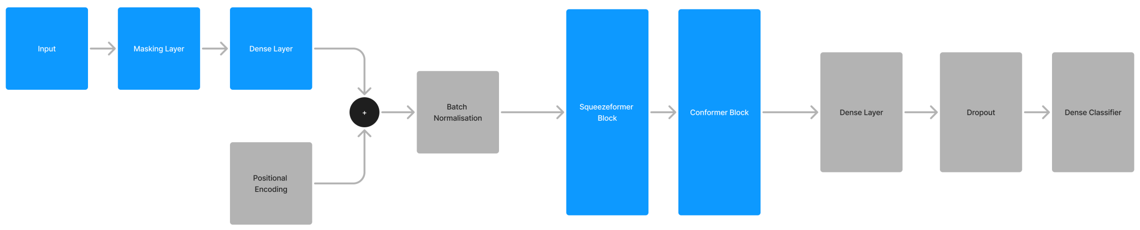


Figure 6: Hybrid Model Architecture

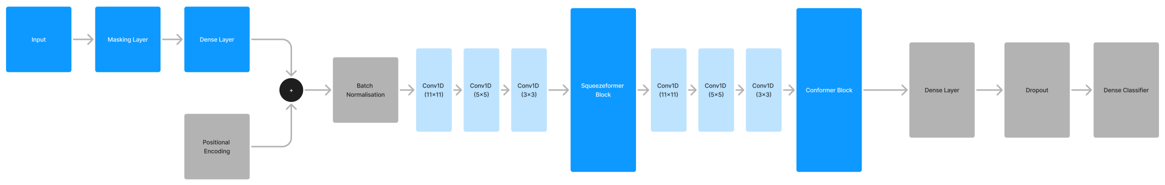


Figure 7: Convolutional Hybrid Model Architecture

Lessons Learnt

- **Data Loading From Kaggle**
  - Pre-processing was computationally intensive.
  - Spent time doing pre-processing for all batches before and loading into TFRecord files.
  - Calling cached TFRecord files later on so time and computational resources aren't wasted.
- **Floating Point 16**

  - Led to a large number of NAN values when used instead of FP-32.
  - Only worked for the first-place solution.

- **Parallel Models**
  - Didn't work due to NAN values from both models.
  - Cannot use NAN values generated from both models at different instances.

Github Link

`https://github.com/tanmayrainanda/ishara`

References

[1] Kaggle, "Google - american sign language fingerspelling recognition." `https://www.kaggle.com/competitions/asl-fingerspelling`, 2021.

[2] Kaggle, "Google - american sign language fingerspelling recognition data description." `https://www.kaggle.com/competitions/asl-fingerspelling/data`, 2021.

[3] Google, "Hand landmarks detection guide." `https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker`, 2024.

[4] World Health Organization, "Deafness and hearing loss." `https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss`, 2023.

[5] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," 2020.

[6] S. Kim, A. Gholami, A. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer, "Squeezeformer: An efficient transformer for automatic speech recognition," 2022.

[7] Dieter and Darragh, "Improved squeezeformer + transformerdecoder + clever augmentations." `https://www.kaggle.com/competitions/asl-fingerspelling/discussion/434485`, 2021.

[8] R. Inglilela, "Aslfr ctc based on prev comp 1st place." `https://www.kaggle.com/code/irohith/aslfr-ctc-based-on-prev-comp-1st-place`, 2021.

[9] C. Deotte, "2 lines of code change - [lb 0.760+]." `https://www.kaggle.com/code/cdeotte/2-lines-of-code-change-lb-0-760`, 2021.

[10] greySnow, "Enhancing sign language recognition with transformer-based sequence modeling and data processing techniques." `https://www.kaggle.com/competitions/asl-fingerspelling/discussion/436457`, 2024.

[11] H. Sohn, "Test & compare asr algorithms." `https://www.kaggle.com/competitions/asl-fingerspelling/discussion/434588`, 2021.